Project Reports

Géraud Le Falher

May 8, 2014

Here is a chronological list of projects I have completed during my first three semesters at Aalto University.

Using naive Bayes, logistic regression and SVM for handwritten characters classification
T-61.3050 Term Project, final report

Géraud Le Falher GERAUD.LEFALHER@AALTO.FI
Student number **336 978**

Abstract

After looking at the dataset of handwritten characters, I examine the effect of supervised (LDA) and unsupervised (PCA) dimensionality reduction. Then I used supervised classification, either with generative (naive Bayes) or not (logistic regression, SVM) model, which give respectively 66.2%, 69.8% and 89.5% of accuracy.

1. Dataset

The training dataset consists of 42,152 handwritten characters, each represented by a binary matrix with 8 columns and 16 rows. Each 1 maps to a white pixel and each 0 to a black one, thus we can visualize some of them[1] to get a sense of what they may look like, like in Figure 1. We can also look at the frequency of each letter in the Table 1. It shows us that the corpus does not follow the relative frequencies of letters in the English language because for instance, there is more *N* than *E*.

After making these observations, I tried to apply advice given in Chapter 11 of Alpaydin's book (Alpaydin, 2009), relative to the preprocessing of characters. But they are already in the center of each picture, and scaled to occupy all the possible space. What could maybe be improved is their rotation but it sounds like a daunting task and apart from that, the dataset is of an excellent quality. So the only thing I did was to convert the text file to MATLAB binary format in order to speed-up subsequents loadings.

[1] I adapt some code of the ml-class courses of Prof. Ng

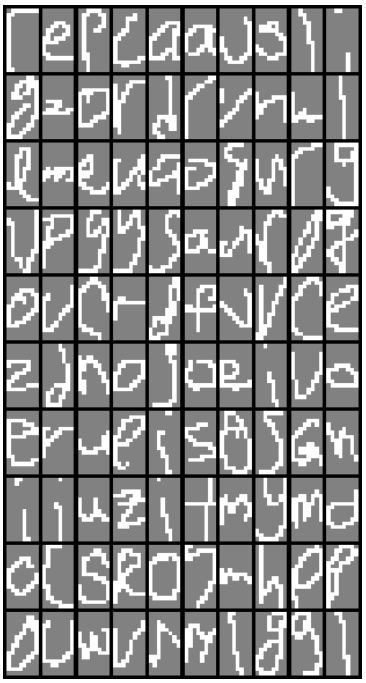


Figure 1: A hundred characters from the training set

2. Dimensionality reduction

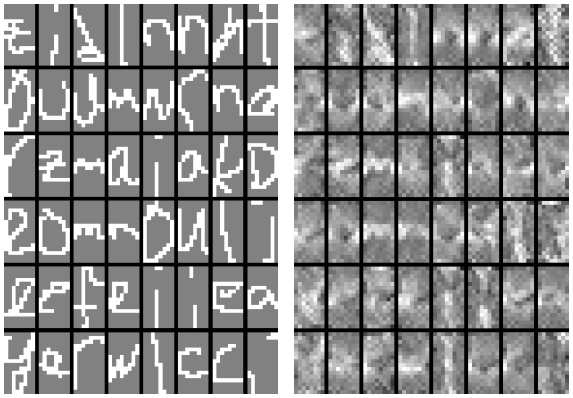
2.1. Linear Discriminant Analysis

I started with linear discriminant analysis because as a supervised method, it seems more promising. For each class, I computed the within class scatter matrix:

$$S_W = \sum_{i=1}^{K} S_i$$

where S_i is the covariance matrix of all the x which belongs to class i and m_i is their mean:

$$S_i = \sum_{i=1}^{K} r_i^t (x^t - m_i)(x^t - m_i)^T$$



(a) The first 48 characters of the training set.

(b) Their reconstruction in a 25 dimension space.

Figure 2: Before/After comparison of LDA reduction

and the between class scatter matrix:

$$S_B = \sum_{i=1}^{K} N_i (m_i - m)(m_i - m)^T$$

where N_i is the size of class i and m the mean of all m_i

Then I get the projection matrix on a subspace of dimension d by taking the first d largest eigenvectors of $S_W^{-1} S_B$ with the constraint that $d < K$, because the rank of S_W is at most K. But the reconstruction was visually blurry (see Figure 2) so I decided to try PCA instead, while later tests show me that it was a wrong assumption.

2.2. Principal Components Analysis

PCA is an unsupervised method which consists of building the projection matrix by taking the first d largest eigenvectors of the covariance matrix S, after having centered the data and normalized the variances, which can be done with the MATLAB function `zscore`. Alpaydin suggests that we can still use the label information by using the sum of the covariance matrix of each class weighted by their estimated probability:

$$S = \sum_{i=1}^{K} \hat{P}(C_i) S_i$$

I did that and it effectively provides a minor improvement in terms of proportion of variance explained, as shown in Figure 3.

A	B	C	D	E	F	G
2958	860	1706	880	3763	723	2239
H	I	J	K	L	M	N
623	3953	189	817	2275	1416	4353
O	P	Q	R	S	T	U
3440	1283	140	2351	1090	1594	2258
	V	W	X	Y	Z	
	664	140	413	1033	991	

Table 1: Letters frequencies in the training dataset

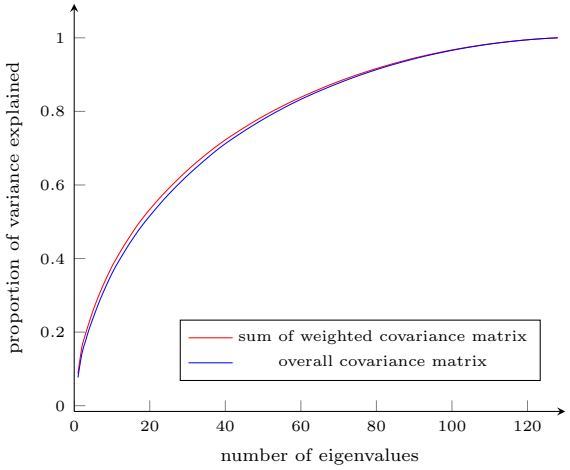


Figure 3: Using label information leads to a (little) more efficient selection of eigenvectors

The reconstruction was also more appealing visually, as shown in Figure 4.

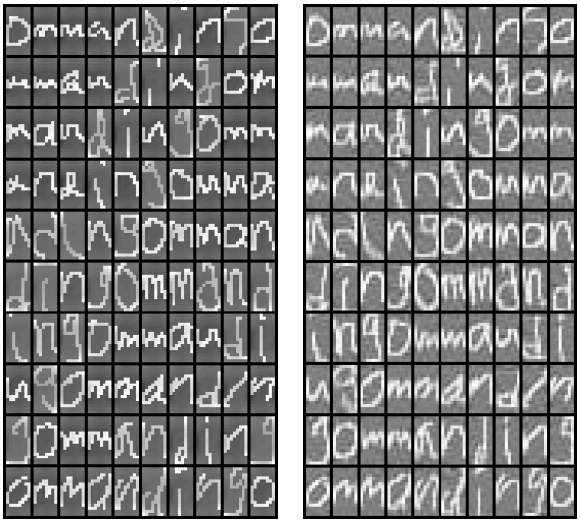
3. Naive Bayes classifier

Because after the dimensionality reduction, each feature seems to approximatively follow a normal distribution (see Figure 5), I decided to start with a naive Bayes classifier. Of course, it requires to verify these assumptions afterward but as it is easy to implement and very fast to train, it sounds like a good baseline.

Basically, we suppose that $p(x|C_i) \sim \mathcal{N}_d(\mu_i, \Sigma_i)$ so the discriminant function for class i is $g_i(x) = \log(p(x|C_i)) + \log(P(C_i))$ and when we plug the usual estimators

$$m_i = \frac{1}{N} \sum_{t=1}^{N} x_i^t \qquad \text{and}$$

$$s_{i,j} = \frac{1}{N} \sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)^T$$



(a) 100 characters after `zscore`.

(b) Their reconstruction in a 63 dimension space.

Figure 4: Before/After comparison of PCA reduction

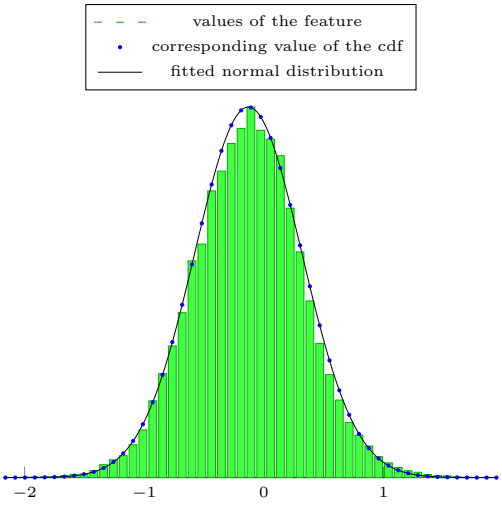


Figure 5: Distribution of the 28$^{\text{th}}$ feature given by PCA.

we get

$$g_i(x) = \mathbf{w_i}^T x + w_{i0} \qquad \text{with}$$
$$\mathbf{w_i} = S^{-1} m_i \qquad \text{and}$$
$$w_{i0} = -\frac{1}{2} m_i^T S^{-1} m_i + \log(\hat{P}(C_i))$$

The result were not very good because this method computes $Kd(d+1)/2 = 54{,}080$ parameters with a dimensionality reduced to 64[2] by PCA whereas there is only 42,152 samples and even less in cross-validation. So I wrote the covariance matrix in the form

$$S_i^* = \alpha\sigma^2\mathbf{I} + \beta S + (1-\alpha-\beta)S_i \qquad \text{with}$$
$$S = \sum_i^K \hat{P}(C_i) S_i$$

and performed a grid search with 4-fold cross validation to find optimal α and β, namely 0 and 1.

To find the best parameter d of PCA reduction, I did a 5-fold cross validation with several values and it turns out that the accuracy reaches a maximum of around 69.1% for $d = 63$ and then becomes somewhat flat (see Figure 6). It appears that this generalization accuracy is rather optimistic because on the test set, the accuracy is only 66.2%.

I earlier mentioned that I discarded LDA whereas it was not as bad as it looks and indeed, we can see in Figure 7 that naive Bayes classifier performs almost as well with only 11 features generated by LDA.

[2]Number chosen because it captures 85% of the variance.

To summarize, while naive Bayes' approach is easy to implement and takes less than a second to be trained and to give a prediction, it does not yield very good results. It is probably because its primary assumption, the "normality" of data, does not hold in a 63-dimensional space with 26 classes. Of course, we could try to use a better generative model in the form of mixture of gaussians but that seems more complicated.

4. Logistic regression

Because doing a full density estimation could be challenging in a high dimensional space and also because it is not required to do the classification, I then tried a linear discrimination classification using the sigmoid function. As seen in exercise session 5, with two class, we could write the log-likelihood as:

$$\mathcal{L} = \sum_{t=1}^{N} \left(r^t \log y^t + (1-r^t)\log(1-y^t) \right) - \lambda \left|\left| \mathbf{w}^T \right|\right|^2$$

with

$$y^t = \text{sigmoid}(\mathbf{w}^T x^t) = \frac{1}{1+\exp(-\mathbf{w}^T x^t)}$$

and where λ is a regularization parameter that ensures that $\mathbf{w}$ does not become too large in order to accommodate for the training set and thus be prone to overfitting.

There is no analytical solution for $\mathbf{w}$ so we need a gradient based method in order to optimize it. Since I found a minimization function on internet[3], I computed

[3]by Carl Edward Rasmussen.

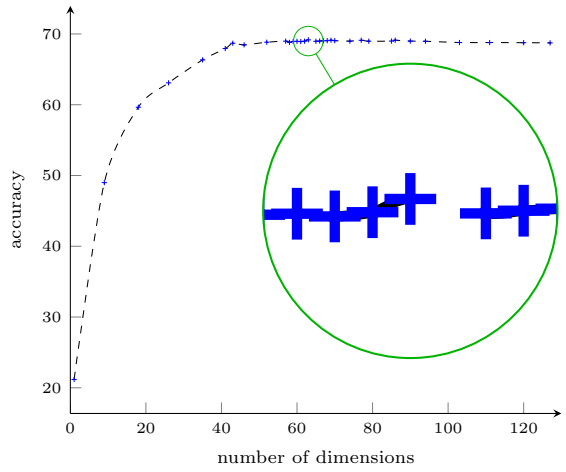


Figure 6: Accuracy of naive Bayes with respect to the number of PCA dimensions retained.

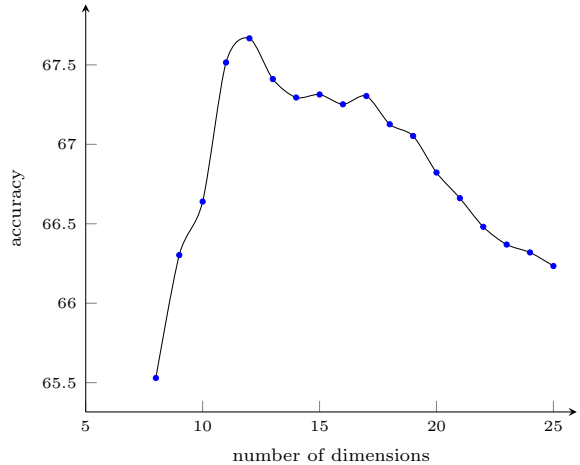


Figure 7: Accuracy of naive Bayes with respect to the number of LDA dimensions retained.

$-\mathcal{L}$ and its gradient:

$$-\frac{\partial \mathcal{L}}{\partial w_i} = -\sum_{t=1}^{N} \left(x_i^t (r^t - \text{sigmoid}(\mathbf{w}^T x)\right) + 2\lambda \mathbf{w}$$

Then I trained a classifier for each class against all the other. At the prediction step, each sample is affected a probability to belong to each class via the sigmoid function and the greatest one is selected.

I have done a 4-fold cross validation to find the parameter λ and it finally give an accuracy of about 76.3% for $\lambda = 10^{-2}$, see Figure 8. But it degrades when I do PCA before so I used the training data untouched. I have not had time to see if training $K(K+1)/2$ pairwise classifiers would give better results, but that sounds promising because it is more likely that two classes are linearly separable compared with one class against twenty five others. Again, this generalization accuracy was optimistic and I only get 69.8% accuracy on the test set.

Compared with naive Bayes' approach, logistic regression has the advantage of making less assumptions on the data. And obviously, its results are somewhat better. Yet, it suffers from longer training time (mainly because of the iterative optimization phase) and it is still a linear model.

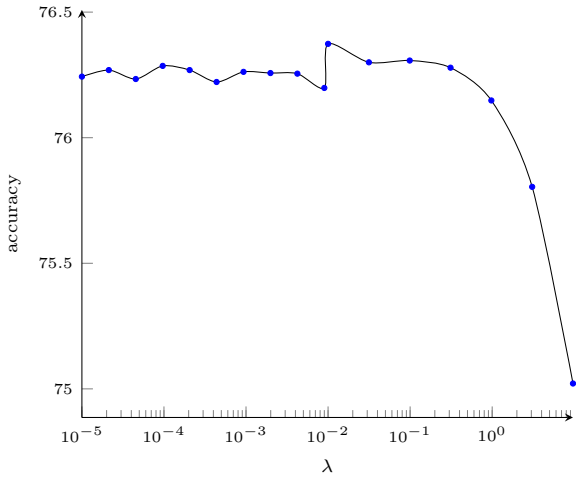


Figure 8: Accuracy of logistic regression with respect to λ.

5. SVM

Assume that we have two classes labeled by $+1$ and -1. The idea in SVM is to find $\mathbf{w}$ and w_0 such that for all points in our set $\mathcal{X} = \{r^t, x^t\}_{t=1\ldots N}$, $r^t(\mathbf{w}^T x^t + w_0) \geq 1$ holds. The distance to discriminant is

$$\frac{r^t(\mathbf{w}^T x^t + w_0)}{||\mathbf{w}||} \geq \rho$$

so we want to maximise ρ s.t. $\rho\,||\mathbf{w}|| = 1$ i.e. minimize $||\mathbf{w}||^2/2$. But to handle non linearly separable cases, we add a "soft error" $C\sum_{t=1}^N \xi^t$ so that the classification condition is now written as

$$r^t(\mathbf{w}^T x^t + w_0) \geq 1 - \xi^t$$

meaning that $\xi^t = 0$ for points far enough from the margin, $\xi^t \in]0, 1]$ for those which are well classified but lied in the margin and $\xi^t > 1$ for the misclassified ones.

So we write the problem in its Lagrangian form

$$L_p = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{t=1}^N \xi^t - \sum_{t=1}^N \mu^t \xi^t - \sum_{t=1}^N \alpha^t \left(r^t(\mathbf{w}^T x^t + w_0) - 1 + \xi^t\right)$$

and we want to minimize L_p w.r.t $\mathbf{w}, w_0$ and maximise it w.r.t α^t, s.t $\alpha^t \geq 0$ and $\mu^t > 0$.

Then we take the dual problem of maximising L_p w.r.t α^t s.t. $\nabla L_p = 0$ and $\alpha^t \geq 0$. Setting the derivative of L_p w.r.t $\mathbf{w}$ to 0 yields

$$\mathbf{w} = \sum_{t=1}^N \alpha^t r^t \alpha^t \tag{1}$$

and w.r.t w_0: $\sum_{t=1}^N \alpha^t r^t = 0$. Finally $\partial L_p/\partial \xi^t = C - \alpha^t - \mu^t = 0 \Leftrightarrow 0 \leq \alpha^t \leq C$. The problem is now to minimize[4]

$$\begin{aligned} L_d &= -\frac{1}{2}\mathbf{w}^t\mathbf{w} + \sum_{t=1}^N \alpha^t \\ &= -\frac{1}{2}\sum_{t=1}^N\sum_{s=1}^N \alpha^t\alpha^s r^t r^s (x^t)^T x^s + \sum_{t=1}^N \alpha^t \end{aligned}$$

s.t. $\sum_{t=1}^N \alpha^t r^t = 0$ and $0 \leq \alpha^t \leq C$

The few samples for which $\alpha^t > 0$ are called support vectors and define $\mathbf{w}$ according to (1). Those satisfying $r^t(\mathbf{w}^T x^t + w_0) = 1$ and $\alpha^t < C$ are on the border and defines w_0. C is a regularisation parameter representing a trade off between a minimal error on the training set and a maximum margin separation. If it is set too large, its high penalty will lead to a lot of support vector, meaning overffiting. Yet, if it is too small, the model is to simple and underfits the training set.

[4]Because other terms cancel out

The flexibility of SVM comes from the fact that we can replace the term $(x^t)^T x^s$ by a custom kernel $K(x, y)$ that should be large when x and y are "similar". One popular choice that I have adopted is the radial basis function

$$K(x, y) = \exp\frac{-||x-y||^2}{2\sigma^2}$$

which introduce a new parameter, σ^2.

By doing a cross-validated grid search to find σ^2 and C, I get $\sigma^2 = 10.8$ and $C = 5$ (see Figure 9, which give a 89.89% generalization accuracy. On the test set, it turns to be 89.50%. Nonetheless, these results are still much better than with the two previous methods. Another advantage of SVM is that they may be improved by using a more appropriate kernel to compare structured images than a simple euclidean distance. But they also come at the price of much longer training and testing times. And more importantly, the optimization problem have analytical solution but they are intractable due to the size of the $N \times N$ Gram matrix $\mathbf{K} = [K(x_i, x_j)]_{i,j}$, which holds more than 1.7 million elements in our case. So one have to rely on sophisticated numerical method described in theses papers (Joachims, 1999; Fan et al., 2005). And because I thought it would be too difficult, I download the libSVM package, which take only two lines of MATLAB but seven minutes of training on my laptop.

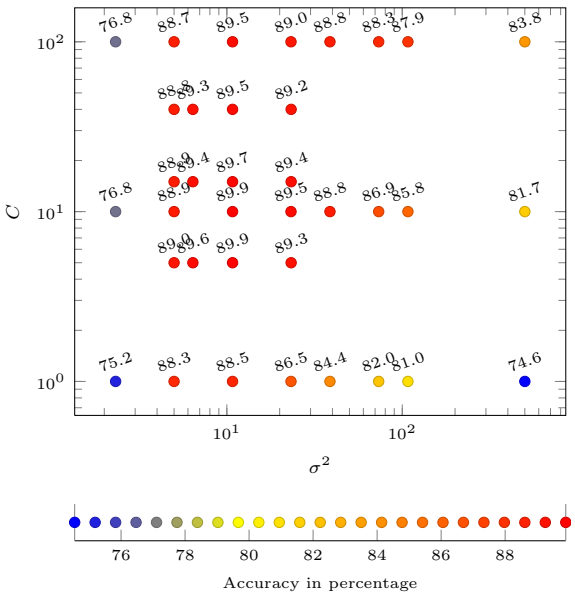


Figure 9: 3-fold cross-validated grid search.

6. Comments on the project

Overall, I found this project stimulating and it was interesting to see a real world application of the material presented in the course. My main difficulty was that during a long time in the beginning, I was a little bit lost at the number of available methods to tackle the problem, so I pondered a lot instead of trying one of them. When I started writing code, it feels better. I have not really kept track of the time spent on the project, but a rough estimate would be twenty hours. I could probably have been faster by spending less time writing this report, but it was a valuable LaTeX experience!

References

Alpaydin, E. *Introduction to machine learning*. MIT press, 2nd edition, 2009.

Fan, R.E., Chen, P.H., and Lin, C.J. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

Joachims, T. Making large-scale svm learning practical. In *Making large-Scale SVM Learning Practical*, chapter 11. MIT-Press, 1999.

T-61.5130 Computer Assignment

ICA for Image Feature Extraction

Géraud Le Falher, **336 978**

January 30, 2013

1 Artificial data

To get acquainted with the fastICA package, I started by mixing 4 simple unidimensional signals ; sinus, sawtooth, Gaussian white noise and Laplacian white noise:

$$
\begin{aligned}
s_1(t) &= \sin(\frac{2\pi t}{0.5}) + \mathcal{N}(0, 0.15) \\
s_2(t) &= 2\left(\frac{t}{a} - \left\lfloor \frac{1}{2} + \frac{t}{a} \right\rfloor\right) + \mathcal{N}(0, 0.15) \\
s_3(t) &= \mathcal{N}(0, 1) \\
s_4(t) &= \text{Laplace}(0, 1)
\end{aligned}
$$

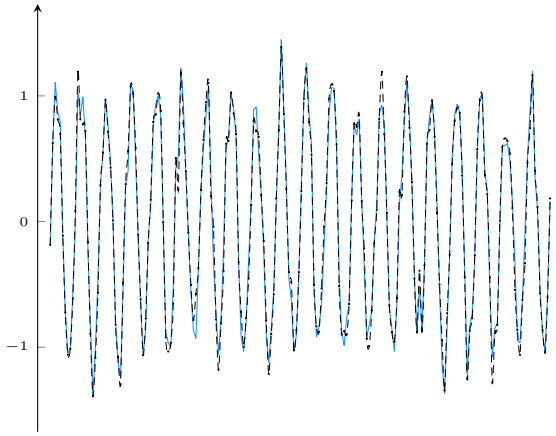
using the following mixing matrix:

$$
A = \begin{pmatrix}
4.72 & 1.13 & 2.16 & 2.53 \\
4.94 & 2.23 & 1.30 & 1.62 \\
2.05 & 1.33 & 0.67 & 3.42 \\
1.86 & 2.29 & 2.10 & 2.21
\end{pmatrix}
$$

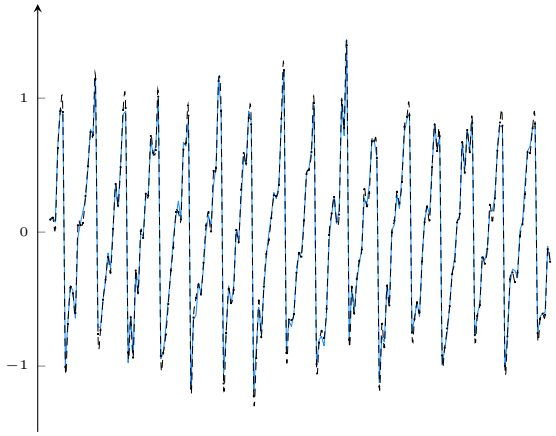
As its name implies, `fastICA` quickly gives good results, once the recovered signals are properly ordered and scaled. To automate this process, I wrote a function that pairs signals $(x, \hat{s})$ based on their root mean squared difference by computing a scaling factor α to minimize it, according to:

$$
\begin{aligned}
RMSE &= \sum_{i=1}^{N} \left(x(t_i) - \alpha \hat{s}(t_i)\right)^2 \\
\frac{\partial RMSE}{\partial \alpha} &= 2 \sum_{i=1}^{N} \hat{s}(t_i)\left(x(t_i) - \alpha \hat{s}(t_i)\right) = 0 \\
\alpha &= \frac{\sum_{i=1}^{N} x(t_i)\hat{s}(t_i)}{\sum_{i=1}^{N} \hat{s}(t_i)\hat{s}(t_i)}
\end{aligned}
$$

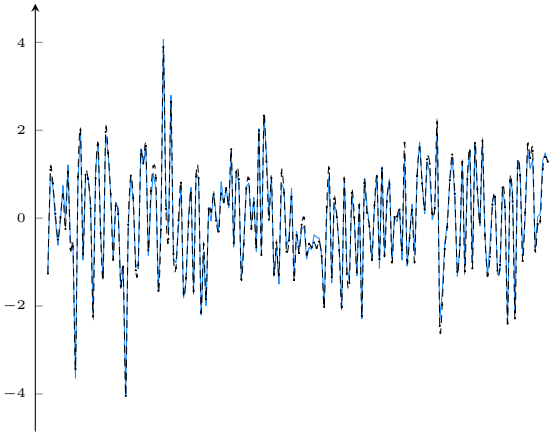
After this step, there is a very good correspondence between original signals and the ones returned by `fastICA()` at least visually, as shown in Figure 1 next page. But of course, we could also make some numerical measurements.



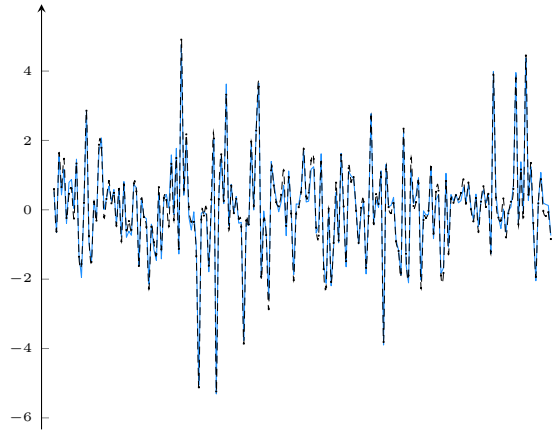
(a) Sinus



(b) Sawtooth



(c) Gaussian



(d) Laplacian

Figure 1: In each case, the original signal is in blue and the recovered points are the black crosses.

option	Sinus	Sawtooth	Normal	Laplace
$default$	$6.140{\cdot}10^{-3}$	$2.360{\cdot}10^{-3}$	$8.459{\cdot}10^{-3}$	$1.201{\cdot}10^{-2}$
$symm$	$1.121{\cdot}10^{-2}$	$4.542{\cdot}10^{-3}$	$1.545{\cdot}10^{-2}$	$1.313{\cdot}10^{-2}$
$stabilization$	$6.140{\cdot}10^{-3}$	$2.361{\cdot}10^{-3}$	$8.457{\cdot}10^{-3}$	$1.201{\cdot}10^{-2}$
$g = tanh$	$2.934{\cdot}10^{-3}$	$2.351{\cdot}10^{-3}$	$8.780{\cdot}10^{-3}$	$1.118{\cdot}10^{-2}$
$g = gauss$	$6.782{\cdot}10^{-3}$	$5.365{\cdot}10^{-3}$	$9.928{\cdot}10^{-3}$	$1.008{\cdot}10^{-2}$

Table 1: Effect of different options on the *rmse* of the four recovered signal.

Distribution	$\mathcal{N}(0,1)$	$Exp(1)$	$F(5,4)$	$Gamma(5,10)$	$\mathcal{U}(0,1)$	$t(5)$
RMSE $\times 10^{-3}$	3.332	13.34	0.2998	1.332	2.744	2.551

Table 2: Error on various distribution

Next, I tried to see how some of the parameters of `fastICA()` affect the result, compared with the default options. First, instead of estimating independent components one by one, one can use a *symmetric* approach. One can also use a *stabilized* version of the algorithm. Finally, instead of the default cubic nonlinearity, which is not robust, I used the hyperbolic tangent and a gaussian function. Overall, the results are of the same order of magnitude, as shown in Table 1. Specifically, *stabilization* seems to have almost no effect over the four chosen signals, *symmetric* is clearly a worse approach, and $tanh()$ appears to be the best nonlinearity. We may also note that the Laplacian distribution is less sensitive to the choice of options than the others.

We may ask ourselves another question: does the default method provide the same accuracy with different kind of signals? For that, we can replace the Laplacian noise with another distribution (namely the Exponential distribution, the F-distribution, the Gamma distribution, the Uniform distribution, and the Student's t-distribution in the Table 2). But overall, that does not seem to be of a paramount effect.

2 Images of natural scenes

Figure 2: A natural scene represented in grayscale as a 256×512 matrix of intensity values.

In this second part, given photos like the one in Figure 2, we want to find independent components in it in order to compress its representation. Indeed, we can approximate the value of each pixel (x, y) as a sum of basis function a_i scaled by a coefficient s_i, as in equation 1, which could also be written under the familiar form $x = As$. In this case, picture's matrix is flatted to fit them in an one dimensional vector, and the i^{th} column of A represents the basis picture a_i.

$$I(x, y) = \sum_{i=1}^{n} s_i a_i(x, y) \tag{1}$$

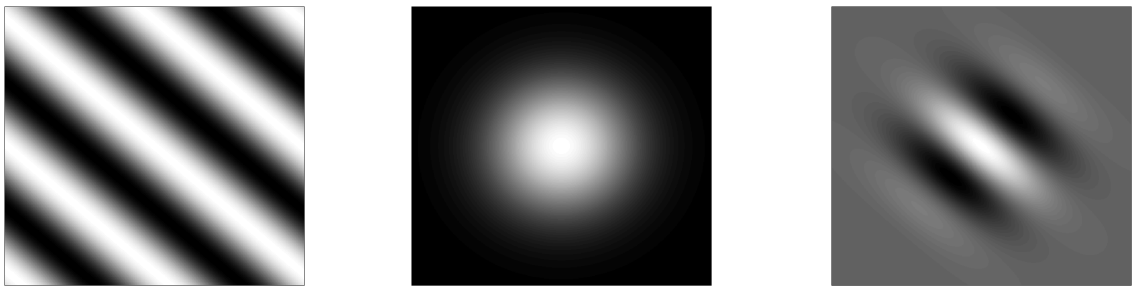
Ideally, only a few basis should be activated at the same time, meaning that a given neuron is activated rarely, which also implies that the distribution of the s_i is sparse. Because sparseness could be seen as super gaussianity, maximizing sparseness is done by maximizing non gaussianity, thus justifying the choice of the ICA method.

On a practical point of view, we start by removing the mean of each picture and divide it by its estimated deviation, using the `zscore` function of MATLAB. Then, because working on the full pictures would be to resource expensive, we extract N square-shaped patches of size $w \times w$ (see Figure 4a p. 5). We also remove their local mean, since it would yield a sub-gaussian component during ICA (see Figure 4b). Finally, we reduce their dimension from $w \times w$ to 169 using PCA to reduce the noise and we run `fastICA` with $tanh$ nonlinearity to get A and s (see listing 1).

Listing 1: Features extraction

```
close all;
clear all;
images = load_images('images/nat*.tif');
% normalize images
for i=1:numel(images)
    images{i}=zscore(single(images{i}));
end
% get patches
window = [16 16];
numpatches = 10000;
patches = sample_patches(images, window, numpatches);
% remove its local mean to each patch
localMeans = reshape(mean(mean(patches)), 1, numpatches);
X = reshape(patches, prod(window), numpatches) - repmat(localMeans, prod(window), 1);
% Try to reduce noise with PCA
C = pca(X','NumComponents', 169);
% Compute basis pictures of the reconstructed version of X
[s, A, W] = fastica(X'*C*C');
```

Theoretically, it is supposed to produce basis that share some characteristics with Gabor wavelet, which consist of a sinusoidal carrier multiplied by a Gaussian envelope, as visualized in Figure 3. But unfortunately, with N =10 000 and $w = 16$, it failed to produced the expected results. Indeed, the basis patches look much more like random noise, like shown in Figure 4c, compared with the expected Figure 4d.

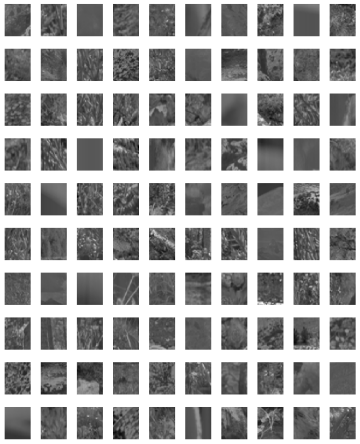


(a) × (b) = (c)

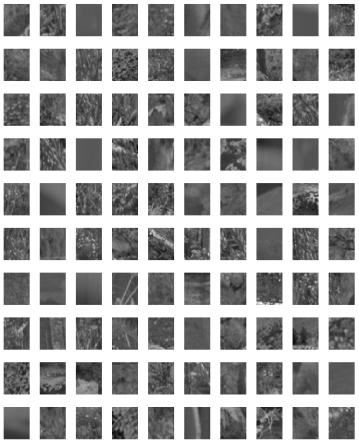
Figure 3: The real part of a Gabor wavelet is the product of a sinusoidal carrier and a Gaussian envelope.

We can also take a look at the recovered coefficients (see Figure 5 p. 6). In the first case (5a), the sparseness of the distribution of the s_i is pretty clear, but is still visible even when using PCA (5b and 5c). Yet, because they can peak at any point, we cannot be sure that there is no higher-order correlation between them. There will then be some "group" of components that will be activated around the same point.

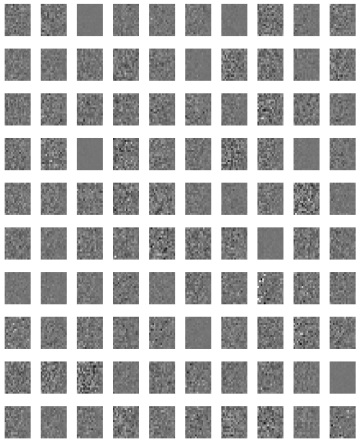
Finally, we can take an interest in the influence of each basis image, by convolving it with a full picture, as in Figure 6a p. 6. The bright areas show where the pattern is activated. For the sake of completeness, I also used one of the columns of A but as one may have expected, it only produced a blurry version that does not convey any meaning (Figure 6b).



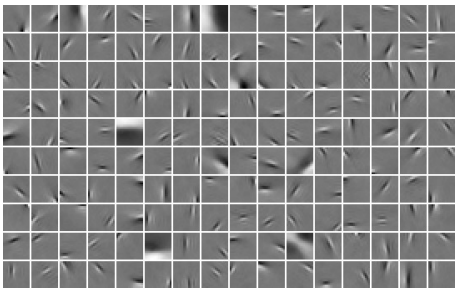
(a) 100 original patches chosen randomly.



(b) The same patches as in Figure 4a after reduction and reconstruction.

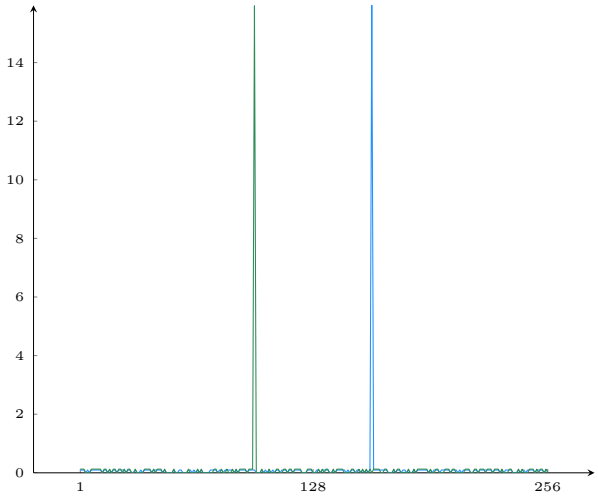


(c) 100 columns of A chosen randomly.

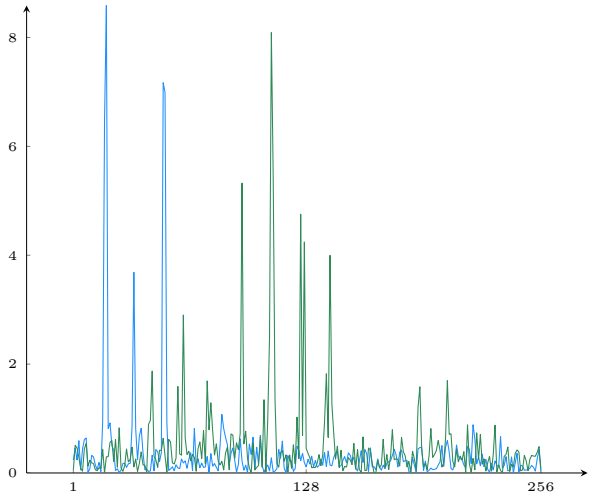


(d) Basis obtained from similar sources as shown in the exercise session.

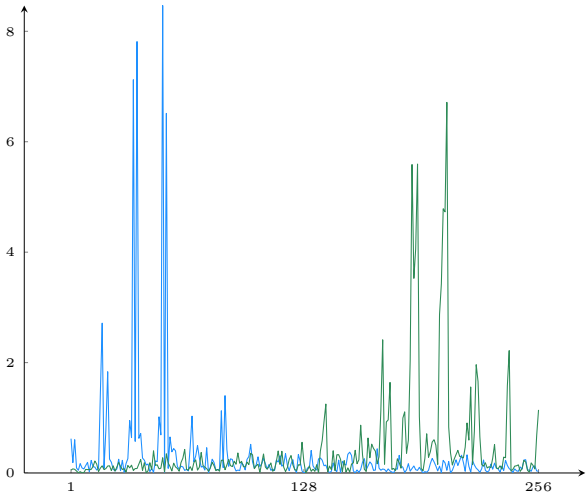
Figure 4: Different kind of picture patches.



(a) $s_{i,89}$ and $s_{i,129}$ with $N = 1000$ and no PCA.

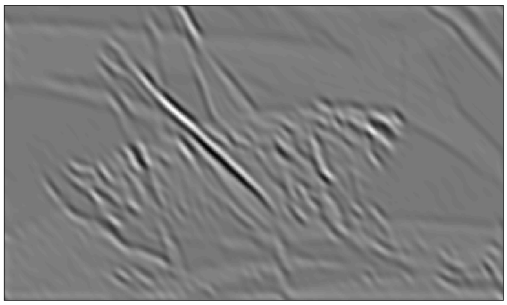


(b) Same with $N = 1000$ and 169-PCA.

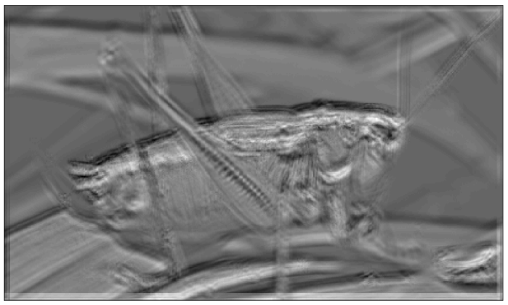


(c) Same with $N = 10,000$ and 169-PCA.

Figure 5: Two randomly chosen independent component plotted under various settings.



(a) The picture of Figure 2 after a convolution with a smaller version of the pattern in Figure 3c.



(b) Same operation using a column of A.

Figure 6: Convolution of image with a basis.

3 Images of building

Figure 7: The main building of the campus.

This time, we use another set of images depicting various building around Otaniemi, like the one in Figure 7. Of course, the mathematics does not change with the semantic of the picture so, without surprise, the same approach gives the same – bad – results.

Signed network analysis

Géraud Le Falher
Sanja Šćepanović

Aalto University

Abstract. We investigate current work on signed network: sociology models, computers science empirical and theoretical research and edge sign prediction methods. Comparison and analysis of the existing models, such as balance and status theory, are provided first. After that, another viewpoint to possible modeling of signed network is given. Finally, we experiment on the task of edge prediction using varety of machine learning methods. To the best of our knowledge, the results using logistic regression based on the number of 4-cycles are better than the result of any of the reported predictions so far: above 98% accuracy with no more than 20% false positive rate.

Keywords: signed network, data mining, sociology theory of balance, link prediction

1 Introduction

Social network analysis is a popular recent research topic among scientists coming from different fields. One aspect of this topic, however, still has not received enough attention – analysis of networks with signed edges. We may not only be connected or not to other people, but we often have positive or negative sentiment attached to our relationship. Similar holds for our attitude towards certain products or services. Examples of online social networks, where such property of relationships is explicitly visible are Epinions, Slashdot, YouTube etc. In those network, people may tag each other as friends or foes, as trustworthy or not, or they my like or dislike a video. In some other online networks, we can infer implicit positive or negative relationship; for instance, Wikipedia admin selection network. In this case, the Wikipedia contributors vote for or against each other being promoted. Thus, it is important to analyse how could the positive/negative relationship in those network be modeled and to find possible explanations coming from sociology or psychology. In this paper, we build on the previous such work, by improving the suggested models first, and then we also present an improved result of the edge sign prediction based on our suggested model.

2 Related work

In this section we make an overview of the current work on signed network. First, we focus on the theories from sociology used to model signed network properties (2.1), and after that we look into computer science research on the topic (2.2).

2.1 Theories from Sociology

After the seminal work in psychology and sociology by Heider [8], Cartwright and Harary [2] have generalized the **structural balance theory** in graph-theoretic language. The structural balance theory suggests that people tend to form relationships in a way that preserve social balance. The social balance requirement can be simply put by these four statements:

1. *"a friend of my friend is my friend"*,
2. *"an enemy of my friend is my enemy"*,
3. *"a friend of my enemy is my enemy"* and
4. *"an enemy of my enemy is my friend"*.

In terms of signed networks, these requirements forbid triangles with exactly one negative or with all three negative edges. Davis [4] has proven, using graph-theoretic language, that, when the balance theory holds in a network, we then observe polarization, i.e., the network clusters into two opposite poles. The people in one pole will have positive relationships among themselves, while all the edges between the two poles will be negative.

The described theory of balance, requiring the above presented four statements to hold in social relationships, is also called **strong structural balance theory**. That is because, in the later years, Davis [4] has shown that if we omit the last statements from the requirements (*"an enemy of my enemy is my friend"*), the theory will better correspond to the situation we meet in real world networks. If we keep only the first three statements, then we talk about Davis' **weak structural balance theory**. On a global level, this theory predicts the formation of a larger number (not only two) of clusters in the network, a situation more often observed in reality. Talking in terms of graphs, the weak structural balance theory just disallows the triangles which have exactly one negative edge. In fact, Davis has proven that such requirement in a network is equivalent with the network having a set of clusters such that all the intra-cluster edges are positive and all the inter-cluster edges are negative.

2.2 Computer Science empirical research

One of the first papers in the field by Kunegis et al. [9], mines Slashdot data in order to give notion of basic network properties in the case of a signed network. The authors present signed variants of clustering coefficient, centrality, distances and similarity measures. The task of predicting sign of edges is as well tackled.

Building on the theories from sociology described above, Leskovec et al. [11] analyse three online social networks with signed edges: Epinions, Slashdot and

Wikipedia admin selection network. Since the structural balance theories are defined in terms of relationships between three people, their empirical evaluation involved counting different types of triads (triangles) in the network, as shown in Figure 1. Out of possible 4 types of triads, the authors find that the triad type with exactly one negative edge (T_2) is constantly underrepresented in the real data. However, the triad type with three negative edges (T_0) is considerably more represented than would be expected if the strong balance theory holds. Thus the authors conclude that the *weak structural balance theory* suggested by Davis models the real networks better than the *strong balance theory*.

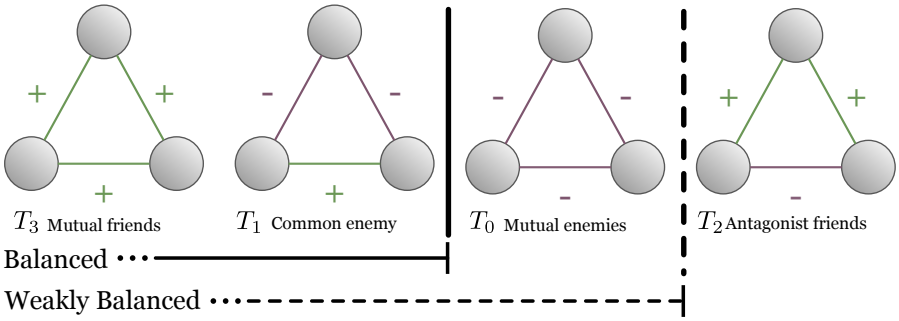


Fig. 1: The four undirected types of triads considered by the structural balance theories.

Since the balance theory does not take into consideration the edge direction, when it comes to directed edge network, the authors turn developing their own idea for a **status theory**, which can explain different level of trust (or status) users enjoy. When working with this theory, the number of different types of triads to count is 16, as shown in Figure 2. The status theory does not correspond to theories from sociology and is considerably less intuitive. It also leaves some level of ambiguity for characterizing certain types of triads as we explain in detail later in section 3.

In the follow up work [10], the authors perform edge sign prediction based on the previously described models of signed network. The accuracy above 90% is achieved on two of the network, and somewhat lower for Wikipedia (80%).

In an earlier work on **trust propagation** [7], Guha et al. have included implicit notion of status theory as formalized later in the described work by Leskovec et al. They approach ultimately the same goal of predicting sign of edges in a network, but using different perspective. Guha et al. report prediction accuracy above 93% in the best case. In this paper, the problem is tackled starting from local atomic propagations, and then analysing possible iterative propagation outcomes. In comparison to the approach by Leskovec et al., where only local properties of the network seem to be considered (types of triangles), Guha et al. look more into the global structure of the network and diffusion of trust as a

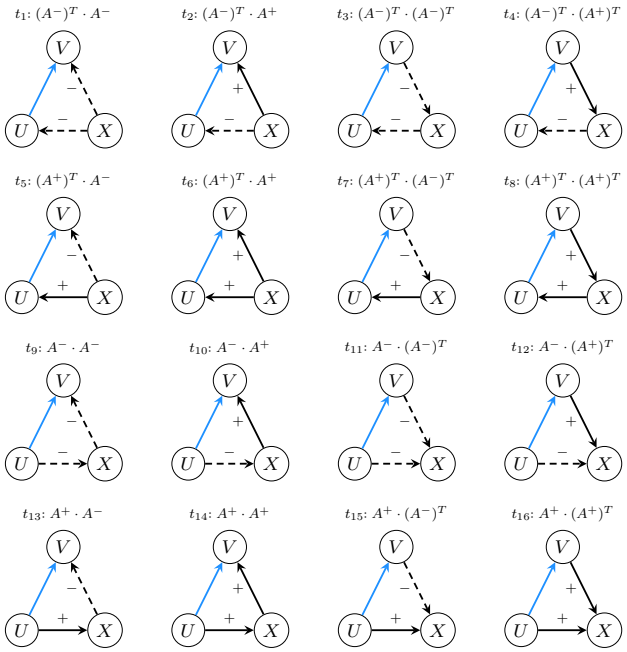


Fig. 2: The 16 types of triads described by the status theory.

process on it. Depending on the trust propagation scheme that is chosen, this paper offers couple of stationary trust states. The authors leave the possibility that a different propagation scheme might be applicable for a different actual network.

Another global approach build upon the k-clustered structure of a weakly balanced network. The adjacency matrix A of such a network can be thought as a partial observation of the underlying adjacency matrix $A^\star$ of the complete graph, which is of rank k [see 3, Theorem 13]. The idea is then to estimate $A^\star$ from A in order to predict the sign of unobserved edges. Because k is assumed to be small compared to the number of nodes, [3] suggest to minimize the rank of X, a binary matrix which must match all non-zero (i.e., observed) entries of A. It is a NP-hard problem in the general case, but it has been extensively studied in the context of collaborative filtering. Therefore, several relaxations and optimization strategies are proposed, that yield good accuracy (88% on the Wikipedia network) despite significantly lower training time compared to local methods [see 3, Tables 6 and 7].

In the paper [13], besides simple signed network properties, the authors exploit homophily between individuals in order to improve trust prediction. The proposed framework is named *hTrust*. For measuring the homophily effect, the rating similarity for two users defined as the cosine between their ranking vectors is used.

dataset	# nodes	# edges	# positive edges	# negative edges	# triads
Wikipedia	7115	103,680	81,696	21,984	790,490
Epinion	131,828	840,799	717,129	123,670	13,317,672
Slashdot	82,140	549,202	425,072	124,130	1,508,105

Table 1: Statistic on 3 datasets.

Finally, this global structure of neighboring individuals striving to reach a state of minimum unbalance trough interactions suggest a physical interpretation. Although the field of sociophysics may sound controversial[1], many articles in prestigious publication have touched the problem, for instance by applying the Ising spin model to efficiently compute global balance of large networks [5] or trying to find local minima of the social energy [1, 12]. Unfortunately, if it provided insights on the structure, it does not give explicit method to perform sign prediction.

3 Models for signed network

In this section, first we analyse and compare the existing models for signed network 3.1, and after that we talk about possible directions for improvement of the models 3.2. During the evaluation, we use the same datasets as in the described related work: Epinions, Slashdot and Wikipedia admin selection dataset. The statistics about all the 3 dataset is given in table 1.

3.1 Comparison and discussion on the existing models

As we mention in the Related Work section 2.2, the task of modeling signed network is tackled from two different perspectives so far:

1. *global perspective*: starting from local atomic propagations, and then analysing possible iterative propagation outcomes through the whole network [7], and
2. *local perspective*: only local properties of an edge are used to describe it's sign [11].

Coming from the knowledge of many existing sociophysics models, where it has been shown that the complex systems behavior and properties can be modeled successfully by starting from a simple set of local rules of interaction, we think that the local perspective possibilities for signed network at least deserve to be fully analysed.

The balance and status theories certainly show a good level of applicability to the signed network datasets taken as a representation of social interaction.

[1] One is quick to see himself as the Dr. Seldon in Isaac Asimov's *Foundation* Series and predict the result of the next election or where the next terrorism attack will take place.

However, none of the theories manages to explain all the aspects of the two network that we find in real data. The balance theory cannot, in its simple form, be applied for the directed network. The theory yields a good prediction for a certain undirected triad type, which can, however, be improved a lot if we take into consideration the 4 possible directed triads that correspond to the same type. We illustrate this through Figure 3 (balanced triad T_3 with all positive edges) and Figure 4 (resulting 4 directed triads corresponding to T_3). Namely, the balanced triad T_3 is indeed more represented in all the datasets, as the balance theory predicts. However, if break down the triads according to edge directions into: t_6, t_8, t_{14} and t_{16} — then in the triad of the type t_8, we are actually more likely to find negative third edge than expected. Thinking in terms of status, the node U is indeed more likely to think of itself having a higher status than V due to transitivity of status from V over X to itself. So, if the edge direction would be as in t_8, the undirected triad with 1 negative and 2 positive edges is actually more likely than the triad with all 3 positive edges. From this simple example, we see the power of status theory and the need for taking edge direction into account.

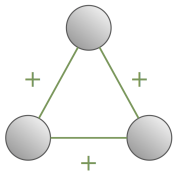


Fig. 3: T_3: The undirected balanced triad with all positive edges.

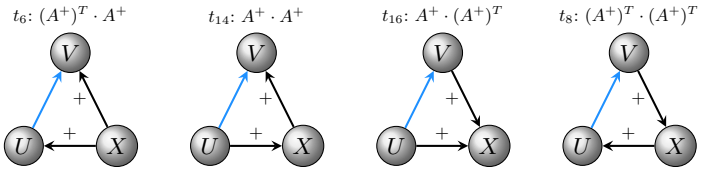


Fig. 4: t_6, t_8, t_{14}, t_{16}: The directed triads of the same balance type as T_3 in Figure 3.

The status theory, on the other hand, is not as intuitive as the balance theory. It also leaves a level of ambiguity, as we present in the next example. For the triad type t_{16} in Figure 4, the status theory described in [11] actually gives different prediction for the edge UV sign, depending whether we take the viewpoint of U (generative surprise), or the viewpoint of V (receptive surprise). We think that a

good theory would need to provide a unique answer in such case, perhaps taking into account more information, that might be needed.

3.2 Analysis on the possible improved models

In order to better understand the principles that lead different directed triad types to be more frequent in real datasets than would be in the random networks, and to seek for information that can give us theory with a better level of certainty, we looked into following statistics. For each of the triad types, we calculate the average popularity of nodes U and V, the directed link between whom we are evaluating. The **popularity** of a node we define to be the difference between positive in degree and negative indegree for the node. As shown in the plots in Figure 5, in the case of certain triad types, node U is more likely to have higher popularity than the node V, and vice versa. Even though the pattern is visible the best in the case of Wikipedia, the similar relationship between the popularity of the nodes U and V in different triad types we find in the other two network as well.

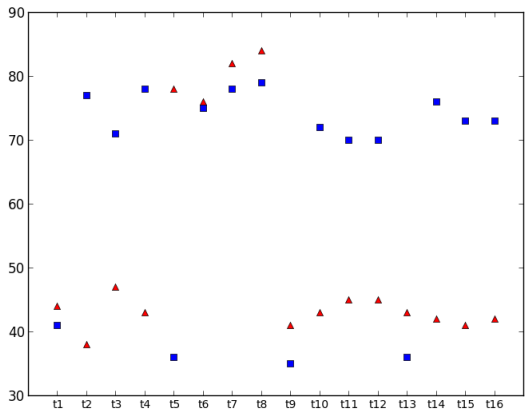


Fig. 5: The average popularity of node U (in red) and node V (in blue) for different directed triad types from Figure 2 (t_1 — t_{15}).

Now we turn to evaluating similar type of statistics for the number of directed 4-cycles (we will refer to those as quads). Namely, in the same way as the triad contexts for each edge may be counted, we can see in how many out possible 64 directed 4-cycles, an edge is involved. It turns out that we find similar, perhaps more clarified situation compared to triads. In figure 6 we show ranks of the nudes involved in different quads (directed 4-cycles). For simplicity, we do not draw the possible 4-cycles. Another statistics that we are interested is to see the

percent of different quad types found in the dataset, and the percent of closing positive edges. Again for the case of Wikipedia, the result is shown in Figure 7. It is clear how certain positive types of quads are many more times found in the dataset. It is also clear that a certain (smaller) number of quad types is much more likely to have negative closing edge sign.

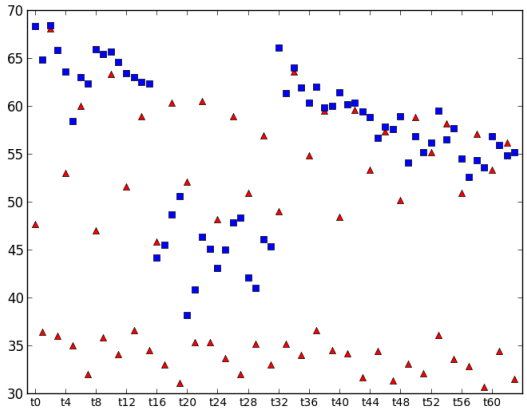


Fig. 6: The average popularity of node U (in red) and node V (in blue) for different directed quad types.

4 Edge sign prediction

In this section, we first describe the machine learning methods that we use in the edge sign prediction task (4.1). After that, we present results of our experiments (4.2).

4.1 Descritpion of the machine learning methods

We use supervised method for which we perform the feature extraction phase as follows. We considered three type of feature for each edge: degree information of its node, embededness and triads count. It can be done by building the graph in memory and iterating over all edges or by looking at the adjacency matrix A, which possibly allow to take longer cycle into account more efficiently. Namely the sign of edge (i, j) is just $A_{i,j}$. Then we construct two adjacency matrix corresponding to the positive and negative subgraph $A^+ = \max(0, A)$ and $A^- = -\min(0, A)$. We use them for the degree's feature $d^+_{in}(j)$, $d^-_{in}(j)$, $d^+_{out}(i)$ and $d^-_{out}(i)$ (e.g. $d^+_{out}(i) = \sum_j A^+_{i,j}$). Then we consider k-cycle of the form

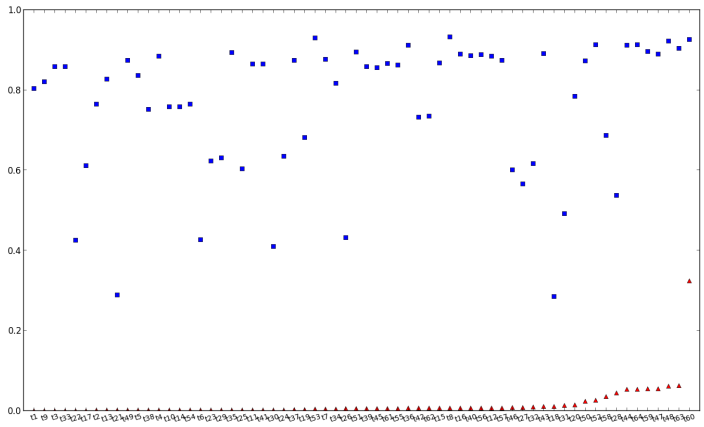


Fig. 7: The (ordered) quads percentage in the dataset (red) and the percent of closing positive edge for each quad (blue).

$i \overset{\pm}{\longleftrightarrow} x_1 \ldots x_{k-2} \overset{\pm}{\longleftrightarrow} j$. For each of the 4^{k-1} of them, we want to count in how many the edge (i, j) is involved. It is done by considering the systematic power of the adjacency matrices of the form $C = (A^{s_1})^{d_1} \times \cdots \times (A^{s_k})^{d_k}$, where the sign vector is $\boldsymbol{s} \in \{-, +\}^k$ and the direction vector is $\boldsymbol{d} \in \{1, T\}^k$ (here T is the transposition operation). The seeked number of cycle is then $C_{i,j}$. For instance, when $k = 3$ and we are looking for triad t_7 ($i \overset{+}{\longleftarrow} x_1 \overset{-}{\longleftarrow} j$), we compute $C = (A^+)^T(A^-)^T$. This method can also deals with undirected cycle. We simply do not consider $\boldsymbol{d}$ anymore and for symmetry reason, paired matrices like $A^{s_1} \times A^{s_2} \ldots A^{s_k}$ and $A^{s_{k-1}} \times A^{s_{k-2}} \ldots A^{s_1}$.

We then use theses features to train two kind of binary classifiers, logistic regression, which is classically defined as

$$\Pr(\text{sign}(x^{(e)}) = +|x^{(e)}, \theta) = \frac{1}{1 + e^{-\left(\theta_0 + \sum_{i=1}^{d} x_i^{(e)} \theta_i\right)}}$$

and radial basis support vector machine.

As mentioned in section 2.2 p. 2, one can also rely directly on the adjacency matrix A without extracting features per se. More precisely, the first relaxation to this NP-hard problem is to minimize, not a discrete value as the rank, but $||X||_F = \sqrt{\text{Tr}(X^*X)}$, the Frobenius norm of X. In this case, $A^\star$ can be perfectly recovered with high probability, provided that the observed A is a uniform sampling of $A^\star$ with enough values [see 3, Theorem 18 for more details]. But finding this global minima is computationally expensive, which explain a further relaxation, where we try to minimize the distance between A and X where A is

non zero. Formally

$$\begin{aligned} &\text{minimize } ||\mathcal{P}(X) - A||_F^2 \\ &\text{s.t. } \operatorname{rank} X \leq k \end{aligned}$$

where the projection operator $\mathcal{P}$ is defined as:

$$(\mathcal{P}(X))_{ij} = \begin{cases} A_{ij} & \text{if } A_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

The problem is now than the uniform sampling assumption does not hold for real network. So the authors propose yet another method, in which the rank k of $A^\star$ is assumed to be a parameter known beforehand (and thus an hyperparameter of the model, one that is arbitrary fixed and not learned from the data), that is $A = WH^T$, with $W, H \in \mathbb{R}^{n \times k}$. They also use loss functions $\mathcal{L}$ more adapted to signed network and a regularization parameter λ to avoid overfitting.

$$\min_{W,H \in \mathbb{R}^{n \times k}} \sum_{(i,j):A_{ij} \neq 0} \mathcal{L}(A_{ij}, (WH^T)_{ij}) + \lambda ||W||_F^2 + \lambda ||H||_F^2$$

Finally, they propose to use stochastic gradient descent for the optimization. At each step t, a edge (i, j) is randomly selected and the respective row of W and H are updated according to these rules (when using a sigmoidal loss function g, as in our case)

$$\begin{aligned} \mathbf{w}_i &\leftarrow \mathbf{w}_i - \eta_t \left(\frac{\partial \mathcal{L}(A_{ij}, (WH^T)_{ij})}{\mathbf{w}_i^T} + \lambda \mathbf{w}_i^T \right) \\ &\leftarrow (1 - \lambda \eta_t) \mathbf{w}_i^T - \eta_t \left[A_{ij} g(A_{ij}, (WH^T)_{ij})(1 - g(A_{ij}, (WH^T)_{ij})) \right] \mathbf{h}_j^T \\ \mathbf{h}_j &\leftarrow \mathbf{h}_j - \eta_t \left(\frac{\partial \mathcal{L}(A_{ij}, (WH^T)_{ij})}{\mathbf{h}_j^T} + \lambda \mathbf{h}_j^T \right) \\ &\leftarrow (1 - \lambda \eta_t) \mathbf{h}_j^T - \eta_t \left[A_{ij} g(A_{ij}, (WH^T)_{ij})(1 - g(A_{ij}, (WH^T)_{ij})) \right] \mathbf{w}_i^T \end{aligned}$$

Although the complete matrix is $-1, +1$ valued, because W and H are initialized at random, they can have any real value. We take advantage of this to consider such values as score (instead of binary decision), bring them to $[0, 1]$ using a sigmoid and classifies using a 0.5 threshold (so that positive value still result in positive prediction). One shortcoming of the low rank completion approach is that it does not really generalize. If every edge between two existing nodes can be predicted, as soon as a node is added, A can no longer be considered as an approximation of the adjacency matrix of the network.

4.2 Experimental results

To asses the performance of our model, we use 10-fold cross validation as in [10]: we randomly split all the edges into 10 bins, hide one of the bin in the

graph (by setting them to 0), extract the features and train the models on the remaining 90% edges and test its prediction on the last 10% using the full graph again. For each method, training time, we report accuracy, the proportion of correct predictions, and F_1 score, the harmonic mean of precision and recall. Because these last two measures are biased by the skewed distribution of classes, we also consider: false positive rate, the proportion of negative edges incorrectly classified as positive and AUC, the area under the receptor operative curve. The interested reader could refer to [6] but briefly, all our classifiers output for each edge a probability to belong to the positive class and then, a class with a simple threshold rule: if $\Pr(\text{class}(x) = +) \geq t$, output $+$, otherwise output $-$. All these decisions can be summarized by a point in a graph with false positive on the x axis and true positive on the y axis. A classifier always outputting $-$ is in $(0, 0)$, in $(1, 1)$ if it always output $+$, in $(1, 0)$ if it always take the correct decision and in $(1 - t, 1 - t)$ if the probabilities are random. By varying t, one can build a curve in this graph and compute the area below it, which is over 0.5 if the classifier outperforms random predictions. We also compare our three methods with random prediction and always positive prediction.

We start with Wikipedia (see Table 2), because its smaller size allows faster experimentation. Some comments can already be made at this point. First, in the case of directed 3-cycle (or triad), despite significantly larger training times, SVM does not bring any improvements over logistic regression. It explains why we have discard it in the following test. These two methods are also dependent of a higher embedness to provide good accuracy. But even in the case of 2c, they both perform rather poorly with worse AUC than random prediction and astronomically high false positive rate. On the other hand, low rank modeling approach outperforms random and always positive predictions. It may seems paradoxical that performances degrade when embedness increases, but there is no causal relation; it just that as the training set size decreases, the adjacency matrix A becomes sparser and it is more difficult to infer the complete matrix $A^\star$. Likewise, when we consider longer cycle, embedness is no longer relevant because two nodes can be involved in such cycles without sharing any common neighbors. Finally, we see that the longer the cycle, the better the result and that directed version is superior to the undirected one. Except of course for the strange case of directed 4-cycle, that achieve "hard-to-believe" results. Our first thought was that information about the sign was somehow present in the features but we were not able to find where. At least no columns is more that 0.49 correlated with the sign column. To add to our confusion, the exact same feature extraction code was use for all cycle length. What is also surprising is that when using five features corresponding to the largest component of the complete θ vector, we still get 91.4% accuracy, 30% false positive and 0.953 AUC.

It is worth noting our analysis on which are the 4-cycle (quad) types that are the most important features in our regression model. In all the 3 datasets, we find that those features are the most numerous quad types which are most likely to be closed with a negative edge (see Figure 7).

The results on Slashdot (Table 3) and Epinions (Table 4) follow a similar pattern, with logistic regression on 3-cycle giving a lot of false positive, low rank performing decently (although the parameter $k = 7$ was chosen based solely on Wikipedia performance) but at the cost of larger training time and 4-cycle features achieving uncomfortable results.

To see if these results can be generalized, we train our model on one dataset and test it on the others (Table 5) as in [10]. Epinions can be considered as "the best teacher" and Slashdot as the worst but overall, the results are rather satisfying, meaning that these networks share a similar structure of directed 4-cycle.

4.3 Conclusion

Based on theories from Sociology and Computer Science, we tried several model to predict edge sign in social directed graph, either global (low rank completion) or local (by counting different type of cycle in which each edge is involved). We test these model on three dataset and get consistent result with the literature, except in one case (directed 4-cycle). It remains to find either the mistake we made or a valid explanation for these good performances.

method	accuracy	F_1 score	false positive	time [s]	AUC
random	0.498	0.610	0.503	—	0.503
only +1	0.788	0.881	1	—	0.5
logistic regression (directed 3-cycle)	0.745	0.853	0.965	0.837	0.408
SVM (directed 3-cycle)	0.742	0.850	0.956	568.4	0.362
logistic regression (directed 4-cycle)	0.995	0.997	0.017	0.864	0.999
logistic regression (directed 5-cycle)	0.865	0.919	0.537	2.001	0.918
logistic regression (undirected 5-cycle)	0.828	0.899	0.716	0.500	0.878
logistic regression (undirected 6-cycle)	0.825	0.898	0.730	0.665	0.876
low rank ($k = 7$)	**0.856**	**0.910**	**0.391**	44.60	**0.766**

(a) 103,680 edges with embedness at least 0

method	accuracy	F_1 score	false positive	time [s]	AUC
random	0.500	0.631	0.500	—	0.499
only +1	0.854	0.921	1	—	0.5
logistic regression (directed 3-cycle)	0.812	0.895	0.945	**0.507**	0.458
SVM (directed 3-cycle)	0.805	0.891	0.918	122.8	0.458
low rank ($k = 7$)	**0.835**	**0.900**	**0.533**	46.92	**0.694**

(b) 58,383 edges with embedness at least 10

method	accuracy	F_1 score	false positive	time [s]	AUC
random	0.493	0.632	0.504	—	0.503
only +1	0.885	0.939	1	—	0.5
logistic regression (directed 3-cycle)	0.852	0.919	0.927	**0.890**	0.503
SVM (directed 3-cycle)	**0.856**	**0.922**	0.894	23.80	0.515
low rank ($k = 7$)	0.844	0.905	**0.501**	47.97	**0.712**

(c) 27,501 edges with embedness at least 25

Table 2: Result on Wikipedia election dataset. The embedness of a edge is the number of common neighbors of its two endpoint.

method	accuracy	F_1 score	false positive	time [s]	AUC
random	0.500	0.607	0.500	—	0.500
only +1	0.774	0.873	1.000	—	0.500
logistic regression (directed 3-cycle)	0.807	0.888	0.801	2.350	0.761
logistic regression (directed 4-cycle)	0.997	0.998	0.013	5.480	0.999
low rank ($k = 7$)	0.857	0.911	0.441	610	0.894

Table 3: Result on Slashdot dataset.

method	accuracy	F_1 score	false positive	time [s]	AUC
random	0.500	0.631	0.501	—	0.503
only +1	0.853	0.921	1.000	—	0.500
logistic regression (directed 3-cycle)	0.901	0.945	0.643	4.032	0.919
logistic regression (directed 4-cycle)	0.981	0.989	0.119	8.259	0.997
low rank ($k = 7$)	0.935	0.963	0.359	1194	0.949

Table 4: Result on Epinions dataset.

trained on:	Epinions	Slashdot	Wikipedia
tested on Epinions	—	0.766 0.007 0.994	0.965 0.233 0.992
tested on Slashdot	0.980 0.084 0.999	—	0.977 0.096 0.998
tested on Wikipedia	0.987 0.017 0.997	0.922 0.004 0.996	—

Table 5: Training a logistic model on each full dataset result in three vectors $\theta_{Epinions}$, $\theta_{Slashdot}$ and $\theta_{Wikipedia}$. These vectors were then used to predict the sign of the edges of the two other networks. Here are reported the accuracy, the false positive rate and the AUC.

Bibliography

[1] Antal, T., Krapivsky, P., Redner, S.: Dynamics of social balance on networks. Physical Review E 72(3), 036121 (Sep 2005)

[2] Cartwright, D., Harary, F.: Structural balance: a generalization of heider's theory. Psychological review 63(5), 277 (1956)

[3] Chiang, K.y., Hsieh, C.j., Natarajan, N., Tewari, A., Dhillon, I.S.: Prediction and Clustering in Signed Networks: A Local to Global Perspective. CoRR pp. 1–37 (Feb 2013), http://arxiv.org/abs/1302.5145

[4] Davis, J.A.: Clustering and structural balance in graphs. Human relations (1967)

[5] Facchetti, G., Iacono, G., Altafini, C.: Computing global structural balance in large-scale signed social networks. Proceedings of the National Academy of Sciences of the United States of America 108(52), 20953–8 (Dec 2011)

[6] Fawcett, T.: Roc graphs: Notes and practical considerations for researchers. Machine Learning 31, 1–38 (2004)

[7] Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: Proceedings of the 13th international conference on World Wide Web. pp. 403–412. ACM (2004)

[8] Heider, F.: The psychology of interpersonal relations. Psychology Press (1958)

[9] Kunegis, J., Lommatzsch, A., Bauckhage, C.: The slashdot zoo: mining a social network with negative edges. In: Proceedings of the 18th international conference on World wide web. pp. 741–750. ACM (2009)

[10] Leskovec, J., Huttenlocher, D., Kleinberg, J.: Predicting positive and negative links in online social networks. In: Proceedings of the 19th international conference on World wide web. pp. 641–650. ACM (2010)

[11] Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: Proceedings of the 28th international conference on Human factors in computing systems. pp. 1361–1370. ACM (2010)

[12] Marvel, S., Strogatz, S., Kleinberg, J.: Energy Landscape of Social Balance. Physical Review Letters 103(19), 198701 (Nov 2009), http://arxiv.org/abs/0906.2893http://link.aps.org/doi/10.1103/PhysRevLett.103.198701

[13] Tang, J., Gao, H., Hu, X., Liu, H.: Exploiting homophily effect for trust prediction. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. pp. 53–62. WSDM '13, ACM, New York, NY, USA (2013)

T-61.5020 Statistical Natural Language: Course assignment

Comparing methods to classify texts according to the personality of its author

Géraud Le Falher—336 978, GERAUD.LEFALHER@AALTO.FI

May 31, 2013

Abstract

In this project, I studied how free texts can be analyzed to determine the five major traits of the author's personality. After carefully extracting a term document matrix using stemming, I compared the classification performance of three methods, Support Vector Machine, MultiNomial Naive Bayes and k-Nearest Neighbors before concluding that the task is difficult. Indeed, the best accuracy achieved overall is 62.9%.

1 Introduction

Human personality can be described in terms of five *traits*[1], presented in [4] as follow:

- Extroversion vs. Introversion (sociable, assertive, playful vs. aloof, reserved, shy)
- Emotional stability vs. Neuroticism (calm, unemotional vs. insecure, anxious)
- Agreeableness vs. Disagreeable (friendly, cooperative vs. antagonistic, faultfinding)
- Conscientiousness vs. Unconscientious (self-disciplined, organized vs. inefficient, careless)
- Openness to experience (intellectual, insightful vs. shallow, unimaginative)

In this project, we will consider that each of these dimensions is binary (whereas one may argue that no individual can be perfectly extroverted or introverted) and we will try to classify people in each dimension based on their writing. More specifically, some students were asked to produce a so called *stream of consciousness* essay, meaning that they wrote their current thoughts freely for twenty minutes. Pennebaker, King, *et al.* [6] have collected 2468 such essays which account for about 1.6 million words.[2] Furthermore, they have labeled this dataset by assessing the personality of each author with a standard questionnaire.

Psychology has showed that the way we express ourselves (for instance by writing) reflects our personality. A summary of these findings in the case of extroversion is given in[4, Table 1]. For instance, introverts tend to use a more diverse lexicon, more elaborated constructions but less positive emotion words. In Bayesian terms, we would say that the text is an observed variable which is conditioned by a hidden one, the personality. It is then quite natural to use a statistical natural language processing approach, first by extracting relevant features (described in Section 2 on the following page) and then training classifiers of a different kind like MultiNomial Naive Bayes (MNNB), k Nearest Neighbors (kNN) and Support Vector Machine (SVM) (as explained in Section 3 on page 5).

In itself, this problem is not of much interest, first because it is not a very convenient for the people being assessed to spend twenty minutes writing a text and also because this information alone is of little use. Yet it is interesting as a sub routine for a higher level task like matching users in a dating site or forming team of workers. It may also affect the choice of a more refined model for further mining of an individual's texts. Finally, it can be thought as a generalization of sentiment analysis, where instead of deciding between rather straightforward and factual classes (positive or negative), more complex facets of expression are analyzed.

[1] Although others are listed on the relevant Wikipedia page.

[2] It is surprisingly difficult to come with a precise figure because *word* is loosely defined.

2 Pre processing

Before doing any classification, I performed several operations to transform the raw data contained in the file `essays.csv` into a document-term matrix, which is a more suitable representation.

- First, in each line, I separated the text itself from the five labels, which posed no difficulties but is mentioned here for the sake of completeness. Another "easy but annoying" issue was that some characters were generating encoding errors (which was solved not elegantly by removing them, since there was only a few of them).
- I then wondered what to do about numbers not written in full. I changed single digit into equivalent word (as shown in Table 1) and replaced all the others by a single unique token (`xnumx`).

	zero	one	two	three	four	five	six	seven	eight	nine	*total*
raw	18	4,816	1,193	518	287	276	126	95	64	60	7,453
converted	134	5,090	1,814	1,069	737	748	375	297	309	262	10,835

Table 1: Counts of the ten words representing digits. The first line refers to the raw data, whereas in the second, single digit number have been converted to the corresponding word. Although it is probably irrelevant, it is amusing to note most people in this informal setting write numbers with digit and not letters, especially if the number is not 1 (and to some extent, 2 and 3).

- To reduce the sparsity of data, I decided to stem all the words, even though it was not such a severe problem because the texts were all in American English, which is a rather analytical language. Because I used the python language, I first looked at various algorithms offered by the Natural Language ToolKit (NLTK) library[1]. `nltk.WordNet` is based on the morphy function[3] that applies some suffix-suppression rules before looking up in a database of base forms. In my case, it was rather slow and for some reasons, it only removed plural endings (like *s*) but did nothing about past tense verbs. NLTK also implements Porter[8] and Snowball[7] algorithms but although it is highly subjective, I found them somewhat too "aggressive", for instance transforming *was* to *wa*. Therefore I finally chose hunspell[4], which nonetheless came with its own issues such as totally discarding punctuation, segmenting differently (for instance, *mid-day* to *mid* and *day*) and generating some irrelevant alternatives (*thing* is transformed into *thing* but also into *the+ING*) or missing one (*woke* was not changed to *wake*). For the last two problems, Part Of Speech (POS) tagging could have helped but because of the

[3] http://wordnet.princeton.edu/man/morphy.7WN.html
[4] http://hunspell.sourceforge.net/

other issues, the two version of the text were no more aligned. Still in regard with sparsity, it reduces the number of unique tokens from 29,535 to 13,407.

- I also collected some general characteristics of the text, namely: the number of sentences, words and characters; the proportion of punctuation marks and capitalized letters; and the number of words per sentences.

- The use of POS can be indication of the personality. For instance, introverts use more nouns while extroverts favor verbs. Thus I tagged every word with the default tagger of NLTK (based on a maximum entropy model) (I also considered using a Conditonal Random Fields (CRF) implementation [for instance 5] but it required too much training for my goal). It turned out it was the most time consuming operation (more than 23 minutes on my laptop). After that, I counted in every text how many times each of the 27 tags appears. A sample is shown in Table 2.

	PRO	N	V	P	ADV	.	DET	ADJ
total	263,748	260,182	255,939	169,896	154,300	121,431	114,085	90,769
text 1515	189	85	112	135	78	149	194	87
text 1789	108	131	57	86	130	31	69	81
text 1804	38	21	68	35	58	57	24	44

Table 2: Repartition of the eight most represented tag in total and for three random texts.

- One thing that I have not had time to consider is Named Entity Recognition. Yet it would have been interesting to see if this kind of information provide some insight about personality. For instance, we may imagine that someone citing a lot of different places is more likely to be classified as opened to experience.

- The last step was to go through all the texts to find all the distinct tokens along their counts. A sample of that is shown in Table 3 on the following page. Using this list, I processed every text individually to compute its feature vector, that consists of:
  - the five class label
  - the six general characteristics
  - the count of each of the 27 part of speech
  - the count of each of the 13,407 token

i	to	the	and	that	my	a	it	is	of	t
122,593	56,646	40,466	38,077	31,740	29,830	29,153	27,560	25,299	23,177	20,466

Table 3: The first 11 of the 13,407 unique tokens. As expected, most of them are stop words, expect for *I*, which is explained by the nature of a stream of consciousness essays and *t* which comes from negation's contraction, as the texts are written rather informally.

3 Classification

Another way to alleviate the sparsity problem is to resort to general methods of dimensionality reduction such as Self-Organizing Map (SOM) or Independent Component Analysis (ICA). Due to lack of time, I only experimented with Latent Semantic Indexing (LSI), which consist of computing a Singular Value Decomposition (SVD) of the document matrix W and keeping only the r largest values.

MultiNomial Naive Bayes A simple model which assume that all features are mutually independent and only depend of the class, i.e. $P(c \mid w) \propto P(c) \prod_i P(w_i \mid c)$. Because I use a bag-of-word model, $P(w_i \mid c)$ are supposed to follow a multinomial distribution whose parameters are simply the count in the data, according to the maximum likelihood estimation. I use the off-the-shelf implementation of MATLAB[5], whose interface is very simple.

Support Vector Machine A maximum margin separator that operates in a high dimensional space derived from the original one by the use of a kernel. There, the two classes are linearly separated by a hyperplane described by the so called "support vector" instances, selected by the optimization procedure. It works because kernel is a function $K(x, y)$ which measure the similarity between two instances x and y: the smaller it is, the more related they are. The easiest choice was to use a kernel based on euclidean distance (namely, a radial basis function $K(x, y) = e^{-\frac{||x-y||^2}{\sigma^2}}$) but it would have been more interesting to implement a common subsequence kernel[3], $K_n(x, y) = \sum_{s_x \in \mathcal{S}_n(x)} \sum_{s_y \in \mathcal{S}_n(y)} \mathbb{1}_{s_x = s_y}$, that, as its name suggest, keep track of how many subsequences of length n are shared by x and y. On a practical note, I used a C++ implementation written by Chang and Lin [2] with a MATLAB interface. One flaw of my approach was that although SVM requires careful tuning of its two parameters σ^2 and C (for regularization), it is very time consuming so I did it once for one dimension and then use the same parameters for all subsequent runs.

[5] http://www.mathworks.se/help/stats/naivebayes.fit.html

k Nearest Neighbors Contrary to the two other methods, kNN has no training phase but when given a new sample to classify, it finds the k closest points in the training set and assign their majority class. The hyperparameters of the model are k (which I set to 5) and again, the distance used. Using the MATLAB implementation[6], I had the choice between ten of them,[7] and after some preliminary comparison, I chose correlation distance, which is one minus the linear correlation coefficient between two documents seen as a sequence of numerical value. Because I found out it did not hurt accuracy yet it was beneficial to the speed, I used only tokens that appeared in more than 3% and less than 90% of all documents, which gave 1,052 features.

Since the dataset was almost perfectly balanced for every five traits, I decided that accuracy was a sufficient measure of assessing performances of the various methods and distinguishing them from random guessing. The other noticeable methodology point is that all measures reported have been computed from 5-fold cross validation.[8]

Because SVM performs better when all the features are in the same range, I first planned to use tf-idf weighting with a `llc` scheme. Namely, if $N = 2468$ is the number of documents, $a_{i,j}$ the number of times that word j occurs in document j and $d_j = \sum_{i=1}^{N} \mathbb{1}_{a_{i,j}>0}$ the number of documents that contain the word j appear, I replaced $a_{i,j}$ by $b_{i,j} = (1 + \log(a_{i,j})) \log \frac{N}{d_j}$ and then I normalized the column of B to unit vector (or set it to zero if the word appears in all document[9]). But after looking at the initial results, I simply stuck with *standardization*, that is subtracting the mean and dividing by the standard deviation of each column.

	Extroversion	Neuroticism	Agreeableness	Conscientiousness	Openness
	Full matrix				
MNNB	0.564	0.587	0.563	0.552	0.629
SVM	0.557	0.532	0.524	0.528	0.597
kNN	0.533	0.521	0.481	0.514	0.557
	250 most relevant dimensions				
MNNB	—	—	—	—	—
SVM	0.550	0.551	0.542	0.517	0.573
kNN	0.518	0.522	0.509	0.509	0.526

Table 4: Accuracy of the three methods for the five traits, using or not dimensionality reduction (the multinomial assumption does not hold after the SVD reconstruction).

[6] http://www.mathworks.se/help/stats/classificationknn.fit.html
[7] I discard the Mahalanobis distance because computing the covariance matrix took too much time.
[8] For even more details, the code is available on github: https://github.com/daureg/wcpr13
[9] Although it did not happen in this dataset.

The first comment that the results presented in Table 4 on the previous page calls is that this kind of classification is "difficult" in the sense that in most cases, the accuracy is only slightly above what random guessing would have generated. The second is that dimensionality reduction does not produce a very noticeable impact on the results. Then, *Openness* is significantly easier to predict than the other traits for all the methods. Finally, despite its conceptual simplicity, MNNB consistently outperforms SVM while kNN is the worst of the three, probably because the metric used is not well adapted to text.

4 Conclusion

Predicting author's personality from one of their stream of consciousness essays is a difficult task that is naturally suited to Statistical Natural Language Processing (SNLP). After some pre-processing steps (text extraction and cleaning, stemming, POS tagging and term-document matrix building), I tried three machine learning methods to perform classification. Naïve Bayes was the most successful although the others may have benefit from the use of a distance metric tailored to text analysis.

References

[1] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python*. O'Reilly Media, 2009.

[2] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines", *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 27:1–27:27, 3 2011, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[3] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels", *Journal of Machine Learning Research*, vol. 2, pp. 419–444, Mar. 2002, ISSN: 1532-4435. [Online]. Available: http://jmlr.org/papers/v2/lodhi02a.html.

[4] F. Mairesse, M. A. Walker, M. R. Mehl, and R. K. Moore, "Using linguistic cues for the automatic recognition of personality in conversation and text", *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 457–500, 2007.

[5] N. Okazaki, *Crfsuite: a fast implementation of conditional random fields (crfs)*, http://www.chokkan.org/software/crfsuite/ [Last accessed: 2013-05-27], 2007.

[6] J. W. Pennebaker, L. A. King, *et al.*, "Linguistic styles: language use as an individual difference", *Journal of personality and social psychology*, vol. 77, no. 6, pp. 1296–1312, 1999. DOI: 10.1037/0022-3514.77.6.1296.

[7] M. Porter, *Snowball: a language for stemming algorithms*, http://snowball.tartarus.org/ [Last accessed: 2013-05-27], 2001.

[8] M. F. Porter, "An algorithm for suffix stripping", *Program: electronic library and information systems*, vol. 14, no. 3, pp. 130–137, 1980. DOI: 10.1108/eb046814.

List of Acronyms

CRF	Conditonal Random Fields. 4
ICA	Independent Component Analysis. 5
κNN	k Nearest Neighbors. 2, 6, 7
LSI	Latent Semantic Indexing. 5
MNNB	MultiNomial Naive Bayes. 2, 6, 7
NLTK	Natural Language ToolKit. 3, 4
POS	Part Of Speech. 3, 4, 7
SNLP	Statistical Natural Language Processing. 7
SOM	Self-Organizing Map. 5
SVD	Singular Value Decomposition. 5
SVM	Support Vector Machine. 2, 5–7

T-61.5070 Computer Vision: Course assignment

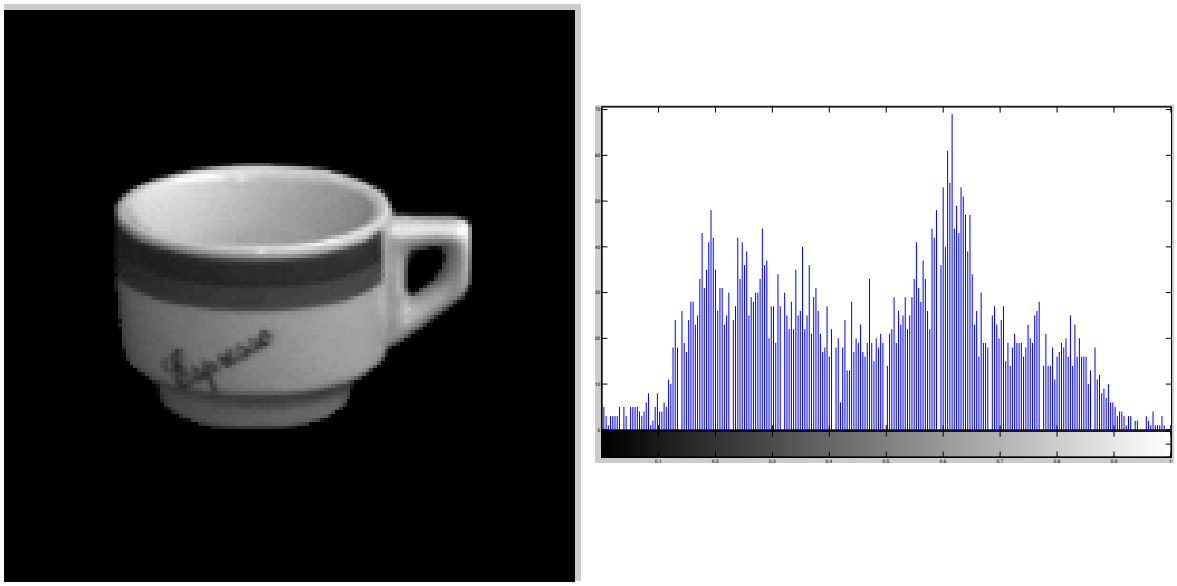
Simple contour-based object detection

Géraud Le Falher — 336 978, GERAUD.LEFALHER@AALTO.FI

May 13, 2013

GETTING A BINARY IMAGE OF THE OBJECT

I first looked at the histogram (Figure 1b) of the original cup (Figure 1a) to select an appropriate threshold value to separate the object from the background. The most common value is 0, corresponding to the black background. But because of the quantization, the border of the cup is a bit blurry and there is no clear cut until the first non zero value, so I chosen 0.1, but it is pretty arbitrary. To have a smoother contour, I then compare opening and closing by square and cross shaped structural elements of various size, as shown in Figure 2 on the following page before opting for 2e, an opening by a 5x5 square.



(a) The gray scale cup. **(b)** Its histogram, omitting the 0 pixel.

Figure 1: Original object.

FOURIER DESCRIPTORS OF THE INNER BORDER

Given the binary image of the object, I used inner boundary algorithm—described in listing 1—to obtained the result shown in Figure 3 on page 4. I then used this list of coordinates $\boldsymbol{B} = \left((B_{x_i})_{i=1\ldots N}, (B_{y_i})_{i=1\ldots N}\right)$ to compute its Fourier transform $\mathcal{F} = \text{abs}(\text{fft}(\boldsymbol{B}_x + i\boldsymbol{B}_y))$, discarding its first component and dividing it by its second to get a position and scale independent descriptor.

(a) Opening by a 3x3 cross **(b)** Opening by a 5x5 cross **(c)** Opening by a 7x7 cross

(d) Opening by a 3x3 square **(e)** Opening by a 5x5 square **(f)** Opening by a 7x7 square

(g) Closing by a 3x3 cross **(h)** Closing by a 5x5 cross **(i)** Closing by a 7x7 cross

(j) Closing by a 3x3 square **(k)** Closing by a 5x5 square **(l)** Closing by a 7x7 square

Figure 2: Various morphological operator tried to get a cleaner contour. Red crosses denote pixel removed from the original thresholded object while green ones stand for added pixel.

```
function B = inner_boundary(picture, label)
    % INNER_BOUNDARY return the inner boundary of the object
    % described by 'label' in 'picture' using 8 connectivity.

    % we first look for the first row from top where the object appear
    j = find(any(picture == label, 2), 1);
    % and in this row, the first column
    i = find(any(picture(j, :) == label, 1), 1);
    % it is the first pixel of the boundary
    B = [i j];
    t = 1;
    dir = 7;
    closed = false;
    % then until we close the border
    while not(closed)
        % we choose the next direction according to the formula
        dir = mod(dir + 6 + (1-mod(dir, 2)), 8);
        % and start looking for neighboring pixel in this direction
        while true
            [ni, nj] = get_dir_pixel(i, j, dir);
            % if we are in the background
            if picture(nj, ni) ~= label
                % we continue in anti clockwise direction
                dir = mod(dir + 1, 8);
            else
                % otherwise...
                break;
            end
        end
        % ...we add this new pixel to the border
        t = t +1;
        i = ni; j = nj;
        B(t, :) = [i, j];
        % and finally check if we are back to our starting point
        closed = t > 3 && all(all(B(1:2, :) == B(t-1:t, :)));
    end
    % remove the last two overlapping points in the border
    B(end-1:end,:) = [];
end
```

Listing 1: Inner boundary

COMPLETE SCENE

I followed similar steps for the whole scene (as shown in Figure 4 on page 5), but I used the value 0.085 as a segmentation threshold.

OBJECT DETECTION

Now that I have the border of these eight objects, I can also compute their Fourier descriptors. The smallest boundary has 211 coordinates so I considered between 2 and 211 of them (since the first one is always one) and look at their distance with the one of the original cup. Except for 210 and 211, it is always possible to find a threshold value that separate the three cups from the other objects. Considering the margin between the

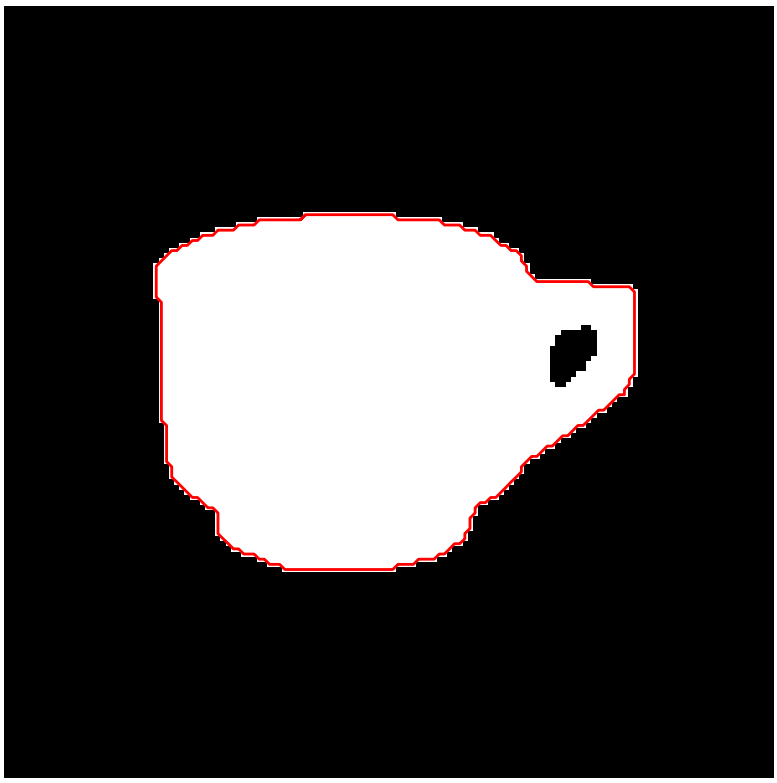


Figure 3: The border of the cup drawn in red.

farthest cup and the closest other kind of object, it is maximised for 31 components although it is not very significant (see Figure 5 on page 6).

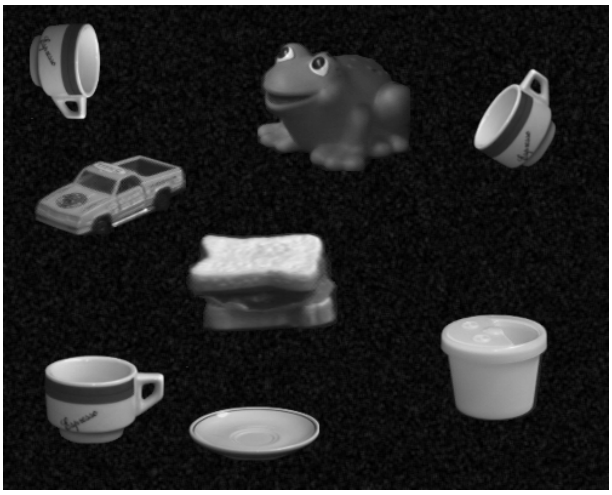
ANALYSIS OF RESULTS

I we consider only relevant number of coordinates (i.e., those for which the margin is more than 0.21), we find that results are rather consistent, as demonstrated in Table 1: the standard deviation is small and the various cups are clearly closer to the original one. The cup with original orientation and the one rotated by an angle of 45° have almost the same distance, which agree with the rotational invariance of Fourier descriptor. Yet the 45° rotated is more than two times farther. I think it may be because discrete low-resolution do not provide very sharp result. In non cup objects, the closest one is the frog and the farthest one the cylinder, although I would have expect quite the opposite.

Fourier descriptors are a good first step to discard objects whose boundary obviously make them non cup. Yet to be more precise, one could sample texture of the candidate object and see if it matches with the original cup.

	90° cup	car	cup	sandwich	saucer	frog	cylinder	45° cup
mean	0.056	0.677	0.057	0.673	0.658	0.424	2.185	0.138
std	0.010	0.002	0.004	0.006	0.005	0.012	0.073	0.011

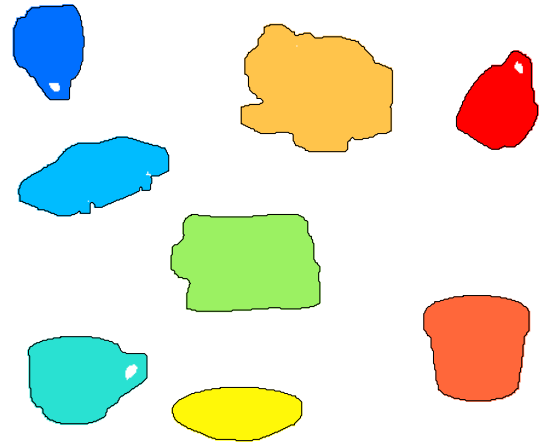
Table 1: Given the distance of each object's boundary to the original one for a large range of coordinates number, we can compute the mean and the standard deviation.



(a) Original gray scene with noise

(b) The noise is still there after thresholding

(c) After opening by a 5x5 square, there is no more noise and object's contour are a little bit better



(d) Connected component are labeled by MATLAB `bwlabel` function which allow me to trace their border in black.

Figure 4: Extracting objects from the scene.

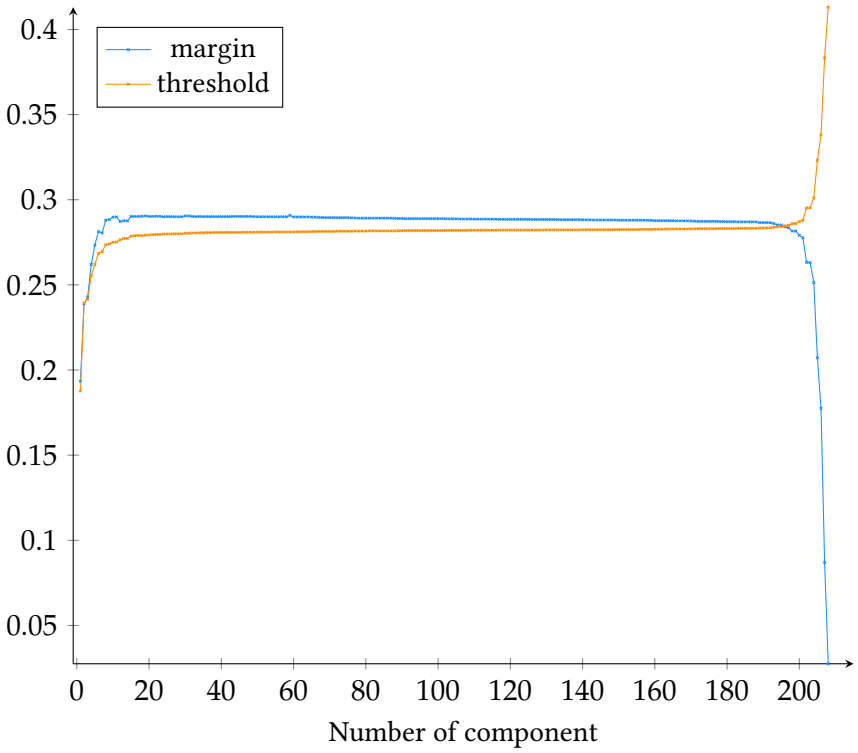


Figure 5: Margin and threshold value between cup-like and other objects.

Visualizing US Census data

Is education worth it?

Géraud Le Falher*—336978

March 14, 2013

1 Motivation and dataset

I chose to analyze the *UCI Adult* dataset, which contains basic demographic information about 32 561 US citizen aged over 16. I describe it with more precision in Table 1, as well as the preprocessing that I applied to it. I then decided to focus on the impact of education regarding several other variables, and especially the income. Indeed, in these times of economic struggles, education is often seen as protection against unemployment and can even be considered an investment, in terms of time and money. On the other hand, some economists have raised concerns about a *Higher education bubble*, describing rising tuition fee and decreasing rate of return. Unfortunately, it will be difficult to answer this question, given that the data are from 1994. We can still generate some visualizations that may give—in principle—an insight about this theory.

2 What is the level of education?

One of the first question I asked to myself is how much the education situation differs from 2012. As a corollary, is the sample of 1994 representative of this year? I first thought of doing a bar chart, but to compel the reader to look only at the numbers I consider relevant, I chose to present it differently in Figure 1. Namely, for each level of education, I put the proportion of individuals in the sample that have reached it, and the difference with the two reference years[1]. It agrees with Tufte's advice because it avoids a lot of redundant ink that would have made up the bar, yet one can still figure out all the raw numbers. I used three colors, black for sample's values, blue and red for 1994 and 2012. It is not really complying with the opponent color theory that would have suggested to pair red with green or blue with yellow but I think it is nonetheless a widely accepted choice, and it may be better suited to black and white printing. In the dataset, there are sixteen different education levels, from preschool to doctorate. To improve readability, I regrouped all of those prior to high school graduation and all of those whose got a higher education diploma. This way, all the values are in the same range and the overall picture is still relevant. You can compare with Figure 5 that tries to encompass all the values.[2]

* geraud.lefalher@aalto.fi

1 using raw data from the US census website.

2 it is situated over the fifth page but I only put it for the sake of completeness.

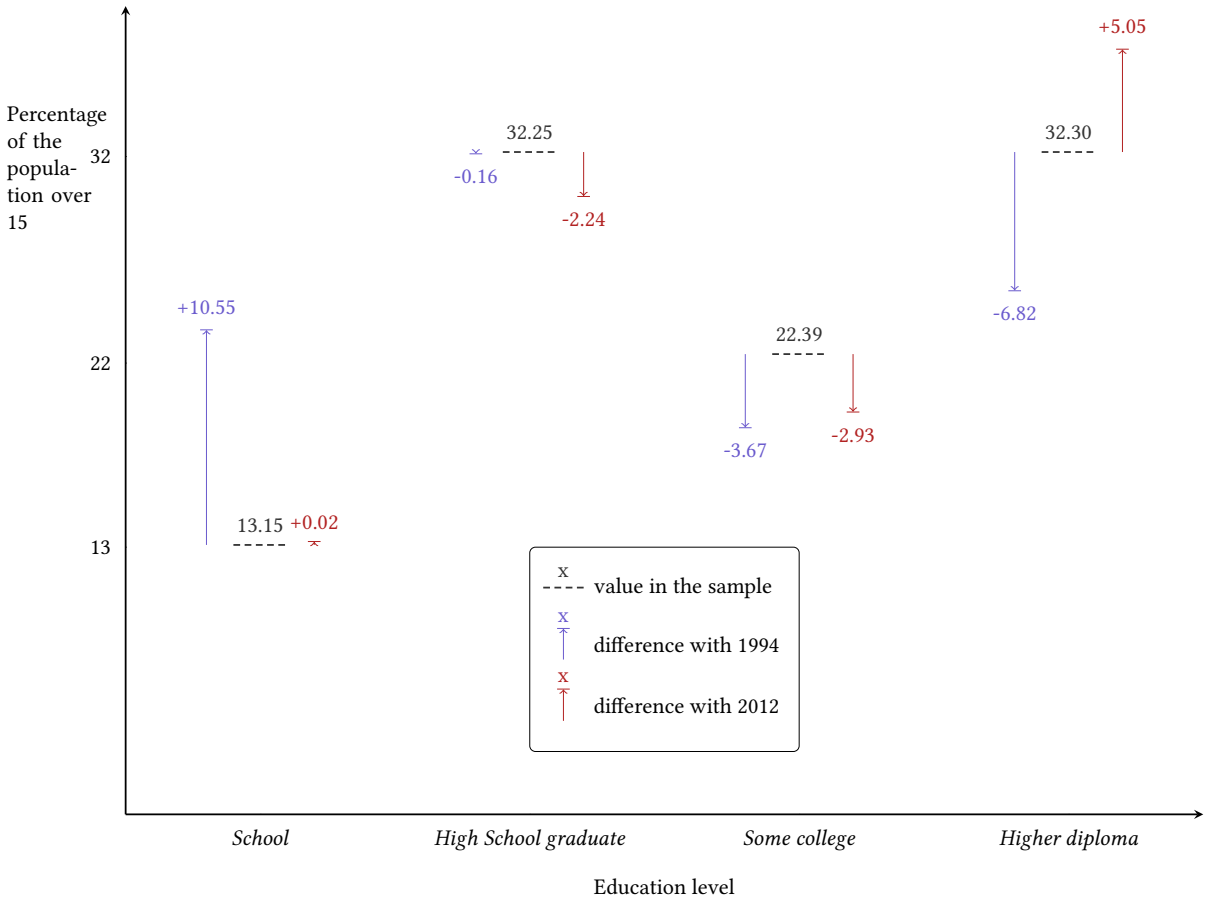


Figure 1: Difference of education's level in the sample compared with 1994 and 2012

3 ARE MEN COMING FROM MARS AND WOMEN FROM VENUS?

Girls allegedly perform better in school before college, when they somewhat fall behind boys. To find if we can see this fact in the dataset, I put in Figure 2 the proportion of people[3] that stop their education at any level and I plot how it differs if we consider only the female individuals[4]. On the substance, we can conclude that is there not so much difference until the high school graduation. But then, women are more likely to leave college without getting a diploma. On the form, the plot is rather simple. I tried to make a grid with relevant (but rounded) values and keep it rather discrete[5]. Yet many things can be improved: some labels—coming directly from dataset—are cryptic, the whole graph is barely understandable without further explanation and the size of percentage in the middle can have been proportional to their value to make them more visual. The main problem of these suggestions is that they would have increase the already critical width of the figure. One may also argue that it only shows 32 values, a low number suggesting that a table would have been enough. Yet I think the plot allows to see these numbers more quickly.

4 DOES STUDYING MAKE ONE DESIRABLE?

I further explored this fundamental duality between men and women using another variable of the dataset: marital status. Again in Figure 3, I regrouped similar situations into only three for legibility

3 i.e., the average of both sexes.
4 who make up one third of the dataset.
5 maybe too much for printing.

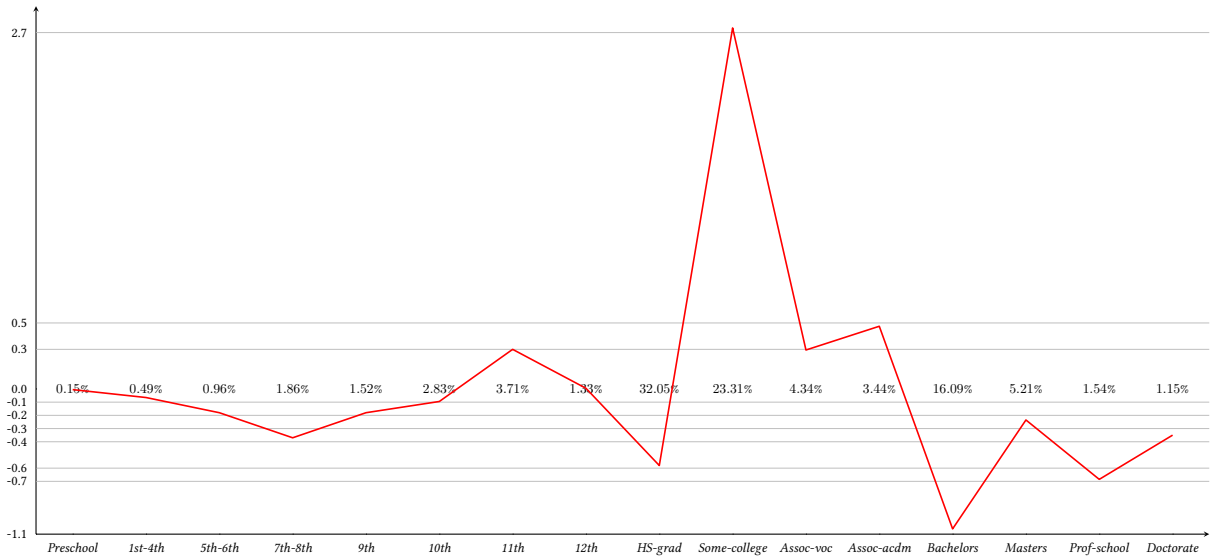


Figure 2: Gender's bias in education.

purpose. It is a bubble chart that uses red/blue opposition to depict a third dimension: is the income greater than $50 000. We can draw several conclusions from it. In general, less than half of the people is married, but if they are, they are more likely to have fully completed their study. The trouble is that we cannot say what is the direction of the correlation, if there is any. It also seems that married people have higher income, but that may be simply because it is household's income. From a visualization perspective, the similarity of bubbles is an incarnation of the small multiples' principle. The area of each circle is directly proportional to the number of people it represents, thus the lie factor is one. The horizontal percentages tell, for a given marital status, how it is distributed among the four education levels. It conversely holds for vertical percentages although I am not convinced it is very clear.

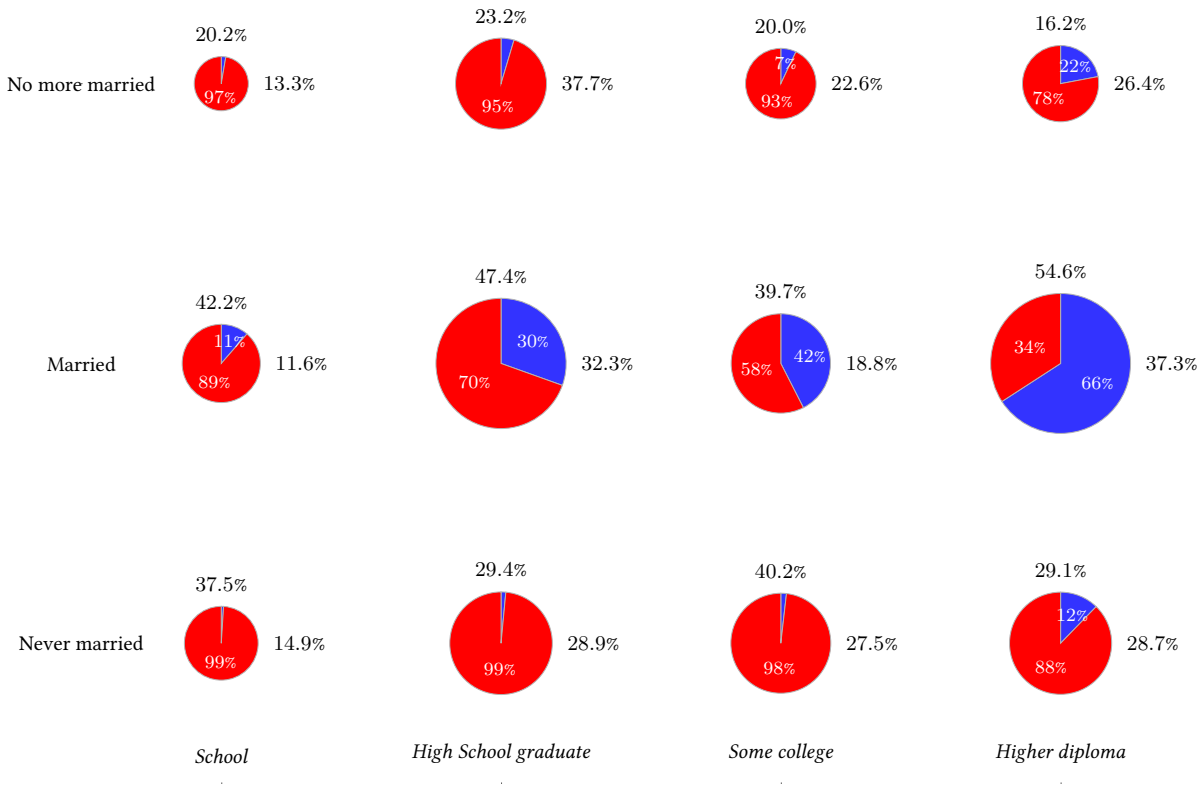


Figure 3: Marital status and education. The blue part of each circle is the fraction with high income.

5 WILL ALL THESE LIBRARY'S HOURS TURNS INTO GOLD?

Finally, let's take a look at the original question, is education a sound investment? I plot binary class (high income or not) against age and education level in Figure 4. In my opinion, this is the most interesting one because of its density. Indeed, it depicts $(90 - 17 + 1) \times 2 \times 4 = 592$ values, which would be quite unbearable in a table. One nice thing about it is that even if we cannot read specific value, because of the Gestalt law of similarity, we associate all the bars of the same color and that allow us to perceive a kind of envelope. It teaches us unsurprisingly that if anyone can earn more than \$50 000, the higher the education, the faster and the more likely it is. Again, it would have been interesting to distinguish between say, Bachelor and Master, but there is not enough sample in the dataset for that to be meaningful.

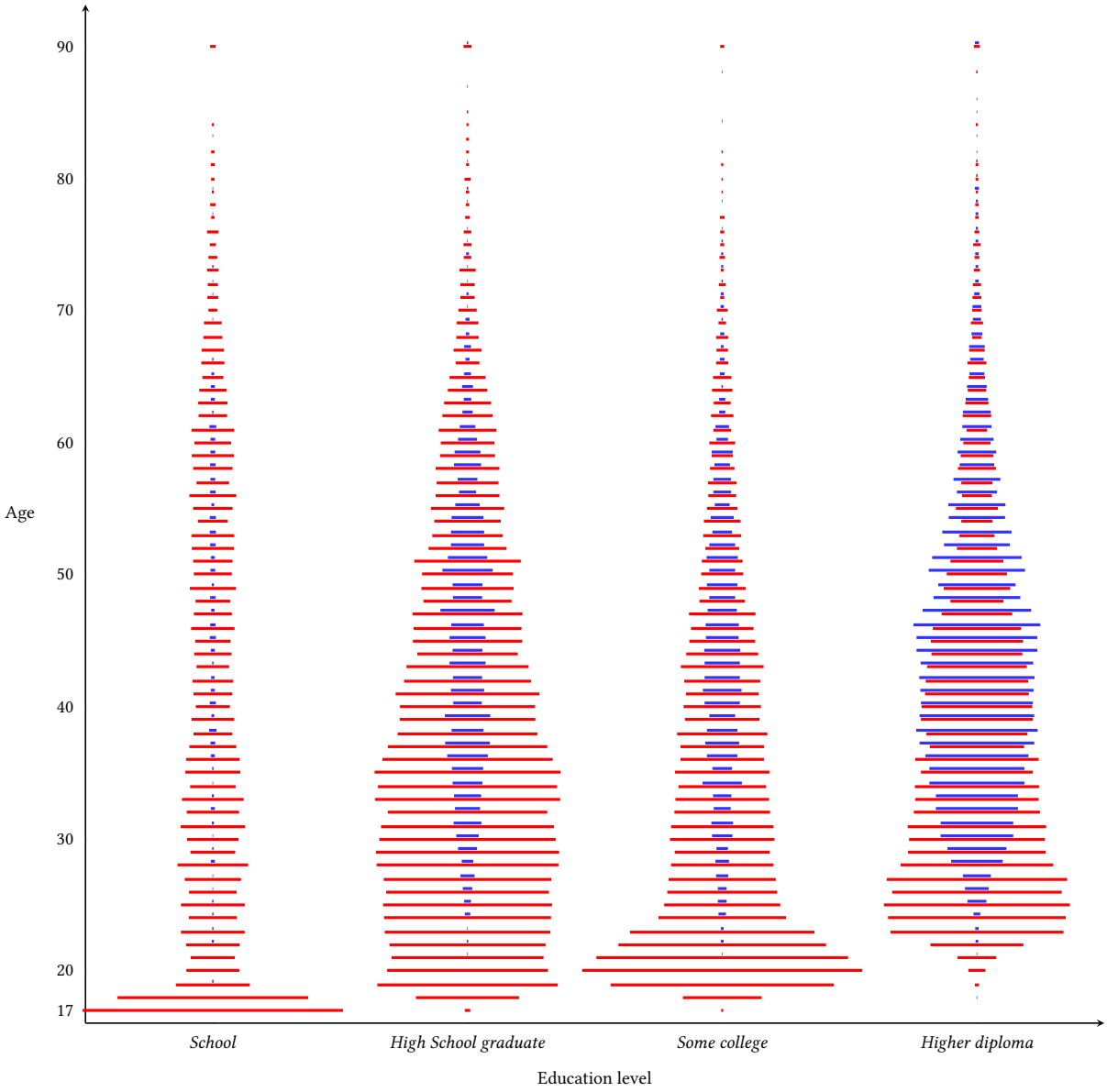


Figure 4: Income given age and education background.

APPENDIX

Dataset in details

Variable	Description	Preprocessing
age	numeric variable ranging from 17 to 90	—
workclass	categorical variable of 8 employment situations	—
fnlwgt	continuous pre computed weight	removed
education	categorical variable of 16 education levels	—
education-num	same as `education` but numerical	removed
marital	categorical variable of 7 marital status	reduced to 3 status
occupation	categorical variable of 14 job kind	—
relationship	categorical variable of 6 familial relation types	—
race	categorical variable of 5 races	—
sex	categorical variable of 2 genders	—
capital-gain	continuous variable	discretized into 3 bins
capital-loss	continuous variable	discretized into 9 bins
hours-per-week	continuous variable ranging from 0 to 99	discretized into 20 bins
native-country	categorical variable of 41 countries	removed
class	binary variable indicating an income over $50 000 or not	—

Table 1: The 15 dimensions of the original dataset, and the preprocessing applied. Some of it was done because I was hoping to get a good 2D reduction using t-statistical nearest neighbor but it did not happen.

Feedback

After choosing the dataset on the noppa page, I spent one or two hours looking at it with Weka to find an interesting phenomenon and how to visualize it. I then made the figures using the Ti*k*Z package of LaTeX, which gives customizable and good-looking results in a reproducible but time-consuming way. Finally, I wrote this report, which took me more time than I expected because as you may have noticed, English is not my native language. Overall, I think a fair estimation would be something around nine hours[6]. Apart from some technical frustration, I think the main difficulty comes from the fact that both visualization and datasets are very rich and open-ended, and this freedom may be a little overwhelming.

6 without counting the 150 minutes that the poor Matlab spent in vain trying to reduce dimensionality.

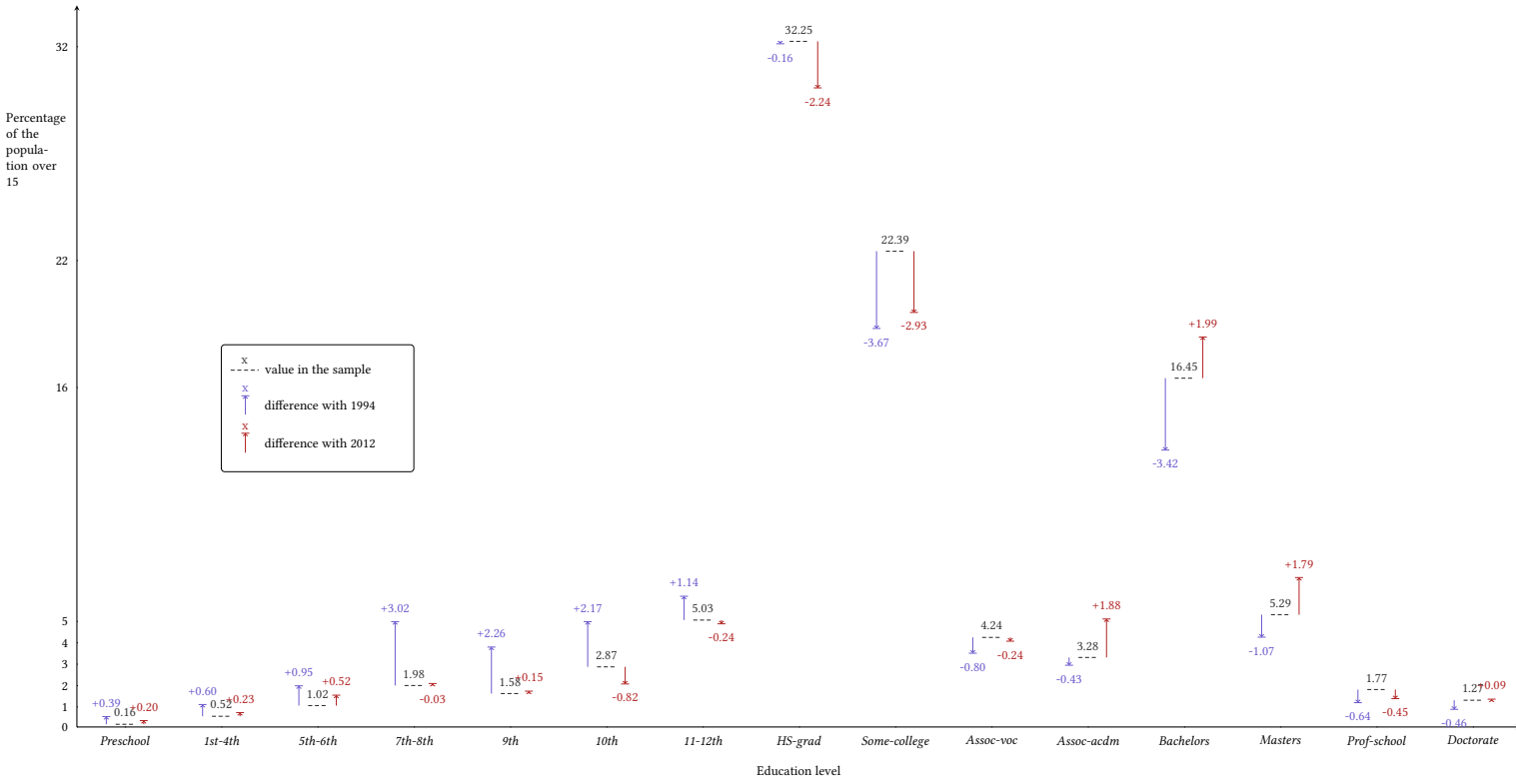


Figure 5: Difference of all education's level in the sample from 1994 and 2012

T-79.5207 Programming Assignment

Spring 2013 — The k-Path Motif Problem

In this programming assignment you will design, implement, and analyse an algorithm of your choice for the following problem.

---

k-PATH MOTIF

Input:
(1) a graph G with n vertices and m edges;
(2) a colouring of the vertices of G with c colours; and
(3) a multiset ("the motif") F of size k consisting of colours.

Task:
Find a path in G on k distinct vertices whose colours agree with F, or conclude that no such path exists.

Example. Fig. 1 displays a vertex-coloured G with $n = 10$, $m = 15$, and $c = 4$. Suppose now the "motif" F is as depicted in Fig. 2, with $k = 5$. In this case G has a path on k distinct vertices whose colours agree with F, as Fig. 3 illustrates. On the other hand, the graph in Fig. 1 has no path that agrees with the motif in Fig. 4.

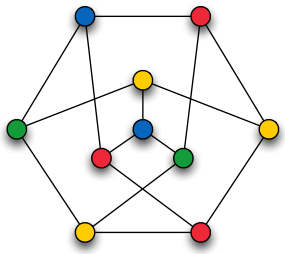


Fig. 1: Coloured graph.

Fig. 2: A motif.

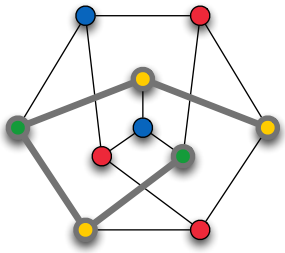


Fig. 3: A path that agrees with motif.

Fig. 4: A motif with no solution.

Complexity. The k-PATH MOTIF problem is (a) known to be NP-hard already when $c = 1$ by reduction from HAMILTONIAN PATH, and (b) known to be fixed-parameter tractable with respect to the parameter k via, for example, colour coding. Thus, an algorithm with worst-case running time polynomial in m is unlikely to exist, but algorithms with worst-case running time $O(f(k)m \log n)$ do exist for $f(k) = \exp(O(k))$. (Here we tacitly assume $m \geq n \geq c$.)

Assignment. But how hard is k-PATH MOTIF in practice? Your task is to explore this question, by

(i) designing, implementing, and analysing an algorithm of your choice;
(ii) constructing small, difficult benchmark inputs to demonstrate the performance of your program; and
(iii) evaluating a solution to (i) and (ii) made by your peer(s).

More detailed instructions are given on the next two pages.

Evaluation criteria. First and foremost, your algorithm and its implementation should be *correct*. For all correctly formatted small inputs (say, $n \leq 12$), your program must terminate in reasonable time, and must not give an incorrect output when tested by a friendly but firm peer. That is to say, randomized algorithm designs are accepted and encouraged – just make sure that the probability of error is small! Second, your algorithm should be *clearly documented* and its implementation *easy to use*, with peer reporting in mind. Third, your program should be *fast* and tuned towards worst-case performance, with an emphasis towards good performance when k is small.

Grading. Max 12 points. Solution to (i) and (ii) max 8 points, peer report(s) written for (iii) max 4 points. Points awarded based on subjective assessment by course staff, utilising peer reports as input.

Instructions (1/2)

Programming language. You may use any programming language you like and enjoy to use, but please do take into account your peers and the performance aspects. Perhaps your programming language of choice should be easily available on most computing platforms, including any libraries that you may decide to use – your peer should be able to compile and execute your program from the source code and instructions supplied by you.

Documentation. Please remember that your submission to (i) and (ii) will be studied by your peer. Make sure to document your submission so that your peer understands what has been done.

Your code versus code written by others. You must indicate if you have used code or libraries written by other people in your program.

Input and output conventions. Your program must read the input from the standard input stream ("stdin") and write the solution to the standard output stream ("stdout"). Additional non-mandatory output, such as program status reports during execution, may be written to the standard error stream ("stderr").

Input format. Input must be accepted in the following textual format. Lines starting with "#" may occur anywhere in the input and should be ignored. The input consists of zero or more consecutive instances of the following form. An instance starts with a single line "`p` n m c k" that gives the parameters n, m, c, and k, which must satisfy $1 \leq n \leq 10^3$, $1 \leq m \leq \binom{n}{2}$, $1 \leq c \leq n$, and $2 \leq k \leq 30$. The parameter line is followed by m lines that specify the edges of the graph G. Each line that specifies an edge is of the form "`e` i j", indicating that vertex $i \in \{1, 2, \ldots, n\}$ and vertex $j \in \{1, 2, \ldots, n\}$ are joined by an edge. Duplicate edges and/or self-loops with $i = j$ must not occur. This is followed by n lines that specify the colours of the vertices of G. Each line that specifies a colour is of the form "`c` i d", indicating that vertex $i \in \{1, 2, \ldots, n\}$ has colour $d \in \{1, 2, \ldots, c\}$. Every vertex $i \in \{1, 2, \ldots, n\}$ must be assigned a colour exactly once. This is followed by one line of the form "`f` d_1 d_2 $\ldots$ d_k" that specifies the motif F, with $d_1, d_2, \ldots, d_k \in \{1, 2, \ldots, c\}$.

Output format. The solution of each instance in the input must be given in the following textual format. If the graph G has a path i_1—i_2—$\ldots$—i_k that agrees with the motif F, with $i_1 < i_k$, write the line "`yes` i_1 i_2 $\ldots$ i_k" to the output. In case there is more than one such path, any one path will do. If the graph G has no path that agrees with the motif F, write the line "no" to the output.

Example. Fig. 5 displays an input that is represented in our input format in Fig. 6. We observe that there is a unique solution, namely the path 5—4—1—3—2, which we must output in our output format as displayed in Fig. 7.

Further examples and test inputs. Further examples and benchmark families for testing and measuring the performance of your program are available in Noppa.

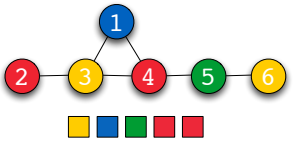


Fig. 5: Coloured graph and a motif.

```
# Example input
# (cf. Fig. 5)
p 6 6 4 5
e 1 3
e 1 4
e 2 3
e 3 4
e 4 5
e 5 6
c 1 1
c 2 2
c 3 3
c 4 2
c 5 4
c 6 3
f 3 1 4 2 2
```

Fig. 6: A valid input.

```
yes 2 3 1 4 5
```

Fig. 7: The solution.

Instructions (2/2)

Timeline. All deadlines are "no later than" (Finnish time).

15 Jan	Programming assignment and supporting materials handed out in Noppa
	(Individual work on the programming assignment)
5 April	Submission of your programming assignment to course staff
8 April	Assignment of peers
	(Reviewing period)
22 April	Submission of peer report(s) to course staff
29 April	Grading announced for programming assignment

Submission instructions for programming assignment. Please submit the assignment to the course staff no later than 5 April (Finnish time). The submission is by email to the course email address "`t795207@ics.aalto.fi`". Please format the title of your email as "`T-79.5207 Programming Assignment: SID Surname, Givennames`". That is to say, if my student ID is 123456 and my name is James Ulysses Bond, I would use the title "`T-79.5207 Programming Assignment: 123456 Bond, James Ulysses`". The email should have a single attached zip archive that contains all the files relevant to the submission.

The submission archive. The archive should be named "`SID_Surname.zip`" and have no more than 2MB compressed size. That is to say, "`123456_Bond.zip`" and no more than 2097152 bytes in size. The following files and subdirectory structure must be present in the archive:

`./`	empty root directory
`SID_Surname/`	submission subdirectory
`SID_Surname/README.pdf`	PDF file containing a short description of your submission (algorithm, implementation, and benchmarks), including your contact information
`SID_Surname/USING.pdf`	PDF file containing a short description on how to compile and use your program
`SID_Surname/PERFORMANCE.pdf`	PDF file containing a performance report on your program
`SID_Surname/source/`	subdirectory with all source code (may have subdirectories)
`SID_Surname/benchmarks/`	subdirectory with your benchmarks (may have subdirectories)

Submission instructions for peer reports. Will be given upon peer assignment.

Questions? First, check Noppa for further information and instructions (code, examples). Second, ask after the lectures and/or tutorials. Third, use the course email address.

Background on motif problems

[1] Nir Atias and Roded Sharan. Comparative analysis of protein networks: hard problems, practical solutions. *Communications of the ACM* 55(5) (2012) 87–97. `http://dx.doi.org/10.1145/2160718.2160738`

[2] Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Probably optimal graph motifs. Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013, Kiel, Germany, 27 February–2 March, 2013), to appear. `http://arxiv.org/abs/1209.1082` (preprint)

[3] Vincent Lacroix, Cristina G. Fernandes, and Marie-France Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM Trans. Comput. Biology Bioinform.* 3(4) (2006) 360–368. `http://dx.doi.org/10.1109/TCBB.2006.55`

A branching algorithm to solve the k-path-motif problem

Géraud Le Falher[*]—336 978

April 5, 2013

This branching algorithm builds its search tree using the `branch` function. This function take as argument:

- the part of the motif that is still unmatched.
- the path so far.
- the nodes that have not yet been considered.

If it can abort, it does it (that is the case if the length of the path is k, if the length of the motif if larger than the number of remaining nodes or if there is no more nodes). Then, it chooses some node to continue the path. If the path is empty, it is just a random node. Otherwise, it considers all possible nodes (that is, the neighbors of the two extreme nodes of the path) which are consistent with the motif—let's call them path's neighbors in the following. But it only keeps a random α fraction of them. For each of these successor nodes v, it generates two branches: one where v is added to the path and another where it is not added. More formally: $\{\text{motif}, \text{path}, \text{nodes} \setminus \{v\}\}$ and $\{\text{motif} \setminus \text{color}(v), \text{path} \cup v, \text{nodes} \setminus v\}$. For each of these branches, the function returns its corresponding recursive invocation along with the new path, the path's length[1] and the nodes still in the graph.

To solve a particular instance, `find_k_motif` first simplifies the graph. After that, it contains only the nodes whose color is in the motif and for each of them, the list of its neighbors. Next, the first call of `branch` (with the complete motif, an empty path, and all the graph's nodes) is put in a priority queue indexed by the length of the path. Then, it recursively push the subtree of the current head of the queue into the queue in a depth first manner, keeping track of the longest

[*]geraud.lefalher@aalto.fi

[1]actually, for implementation reason, it is the opposite of the length

path found so far. It continues until either the queue is empty or a path of length k is found.

To use parallelism, the main function make a first call to `find_k_motif` which returns a list of subtree whose root is at level 2. A new process is spawned to search this subtree as described in the previous paragraph using `find_sub_path` (which starts at a root given in argument). One obvious flaw of this clumsy approach is that the number of subprocess is determined by the input and cannot be specified beforehand. It is also not well balanced, since subtrees that do not contain solution are much deeper. If there is a solution, it is less bad, because as soon as one process found it, they are all terminated.

Because the function branch stops when it is called with zero nodes and because for each recursive call, the number of available nodes is decrease by one, the algorithm halts. Furthermore, by construction, the paths are consistent with the motif. So when the algorithm returns a path, it is correct. Two questions remain. What is the size of the search tree? When α, the fraction of neighbors considered to pursue the path, is strictly smaller that 1, what is the probability of false negatives.

Let's $T(n)$ denote the number of tree nodes at level n, δ_{max} the maximum node degree and $N_i(n-1)$ the number of path's neighbors for the i^{th} nodes of level $n-1$. Then

$$T(n) = \sum_{i=1}^{T(n-1)} 2\alpha N_i(n-1) \leq 2T(n-1)\alpha 2(\delta_{max}-1) \leq (4\alpha(\delta_{max}-1))^n$$

which is pretty bad. For instance, on a complete graph, is it $\mathcal{O}(n!)$. Moreover, there is also the cost of maintaining the priority queue and each of the nodes—containing a partial list of graph's nodes—takes $\mathcal{O}(n)$ space. But it is a rough estimate because we can hope that (1) not all nodes have δ_{max} neighbors, especially with acceptable color and (2) the depth first search will ends before exploring the whole tree. It even turns out that the unicolor complete graph is a favorable case because by setting $\alpha = 1/n$, a path is found in linear time. Another disappointing feature of this complexity is that it does not depend of k. Indeed, because the algorithm do not add a node in the path in every branch, it may need much more that k steps to terminate (contrary to the branching algorithm for minimum vertex cover page 14 lecture 9).

For the graph $G = (V, E)$, let's partition V as $V = \mathcal{P} \cup \overline{\mathcal{P}}$ with $\mathcal{P} = \{v \in V : v \text{ belong to a } k\text{-path}\}$. Denote by X_v the random variable that indicate if the algorithm, starting from v, has found a path. Because the algorithm never drops node from the path, if $v \notin \mathcal{P}$, $Pr(X_v = 1) = 0$. Here we consider the worst case where there is an unique path, hence $|\mathcal{P}| = k$. Assuming that v is in position q in the path, there are $k_0 = q-1$ nodes before it and $k_1 = k-q$ after. If the path can

be continued in only one direction, the probability of selecting a correct set of neighbors is $\frac{\lceil \alpha\delta(v) \rceil}{\delta(v)}$ which I relax to α to make further computations easier. If the two directions are possible, the probability to pick at least one correct node in one of them is $1-(1-\alpha)^2 = \alpha(2-\alpha)$. Starting from v, we first made two-direction selection until we hit one end of the path and then one-direction selection. So $Pr(X_v = 1 | v \text{ is in position } q) = P_q = (\alpha(2-\alpha))^{min(k_0,k_1)} \alpha^{k-min(k_0,k_1)}$. Finally, because the algorithm can independently find the path by starting from any position, $Pr(X=1) = \sum_{q=1}^{k} P_q = 2\sum_{q=1}^{\lceil k/2 \rceil} (\alpha(2-\alpha))^{q-1} \alpha^{k-q}$ by symmetry. Then

$$Pr(X=1) = 2\alpha^{k-1} \sum_{q=1}^{\lceil k/2 \rceil} (2-\alpha)^{q-1} = 2\alpha^{k-1} \frac{(2-\alpha)^{\lceil k/2 \rceil} - 1}{1-\alpha}$$

Let f be the function $x \to \frac{(2-x)^c - 1}{1-x}$. By setting $x = 1-\epsilon$ and doing a first order Taylor's development, one can find that $\lim_{x \to 1} f(x) = c$. So by continuity, when $\alpha = 1$, $Pr(X=1) = k$, which is rather disturbing for a probability. I think it means that the algorithm will find the path k times and that a normalization constant is missing. Nonetheless, it shows that by increasing α, the probability of false negatives is decreased but at the expense of α^n in the complexity. One can also run the algorithm several times to reduce this probability. If there is more than one path, the same analysis can be done and the probability of finding a path is higher because there is more starting points. Yet, it is more complicated because is some path are crossing, there are more than two direction to pursue the path.

Because the algorithm is randomized, I've ran it 50 times on each input to get a histogram of running time. First, I try to see if the multi processes approach bring some improvements. It is the case when there is only one path (see Figures 1 and 2) and when there is none (see Figures 3 and 4). Then I try to construct a small but difficult instance. In my case, it is a graph that has no solution but a high average degree. Therefore, I make a nine complete graph with two color. The second appears only three time in the graph but four time in the motif. It is indeed difficult because in both case of single and multi process, there is a non negligible part of the runs that take significantly more time than the others (see Figure 5). Finally, I also look at the performance when the number of edges grows in Table 1. But unfortunately, it is hard to find a relation between the performance and the number of edges, nodes or even k. For example, $n = 28, m = 55$ and $k = 9$ is better than $n = 28, m = 55$ and $k = 9$. It also interesting to see that because k/n is never too small, the minimum time is always very small.

instance	edges	average	min	max	correct
16-6-1x6	15	0.159	0.15	0.17	1.0
16-9-1x9	20	5.105	0.15	42.24	0.7
20-6-1x6	25	0.166	0.15	0.20	0.88
24-11-1x11	31	79.832	0.16	828	0.88
28-8-1x8	36	42.327	0.15	1024	.96
28-8-2x4	41	5.028	0.15	130	0.96
32-8-1x8	48	17.031	0.15	520	0.9
28-9-1x9	55	2.924	0.15	24.7	0.9

Table 1: Performance of 50 runs of the algorithm over instances with a growing number of edges and nodes. All the time are in second, correct is the percentage of runs that return the right answer

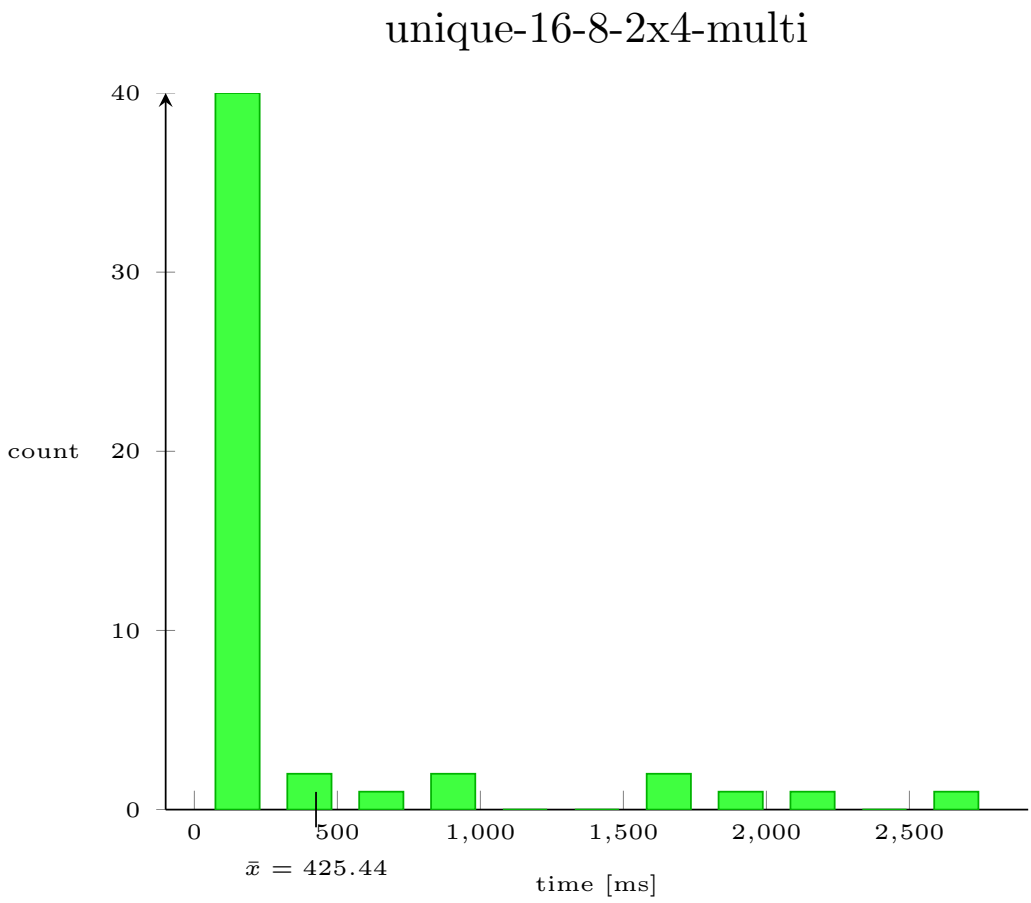


Figure 1: Multi processes with a unique path.

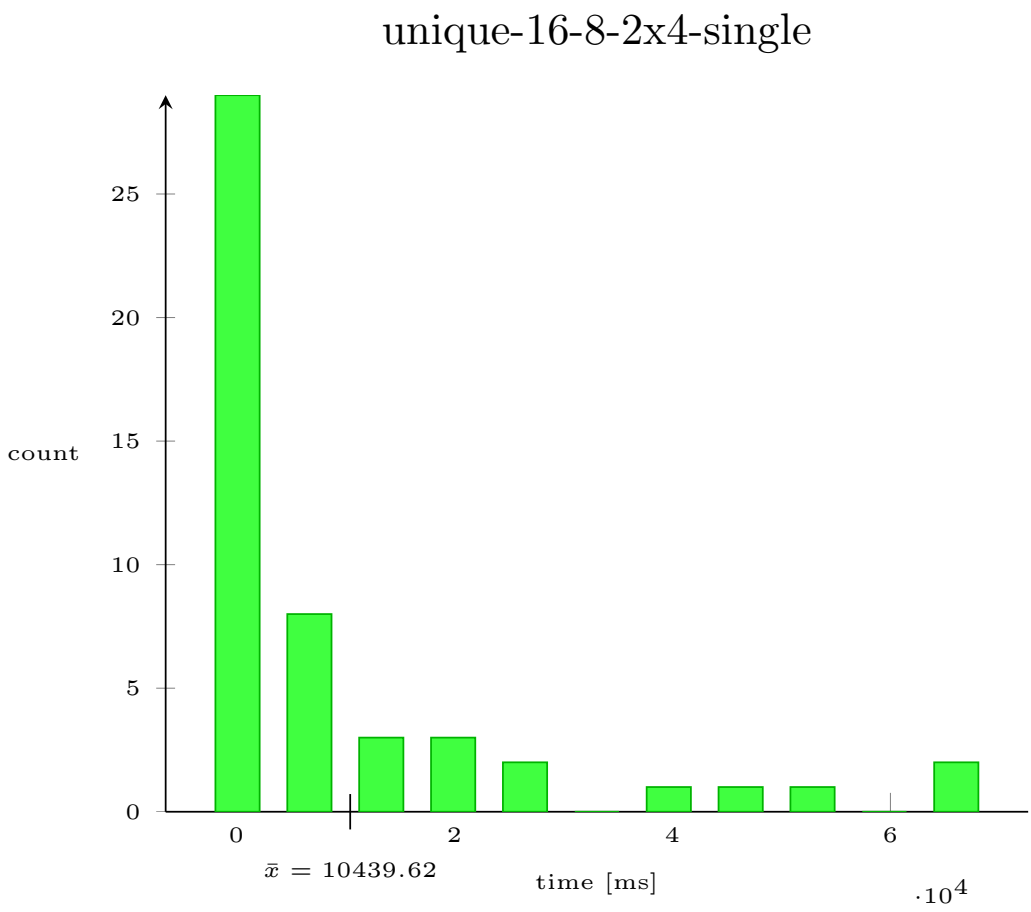


Figure 2: One process with a unique path.

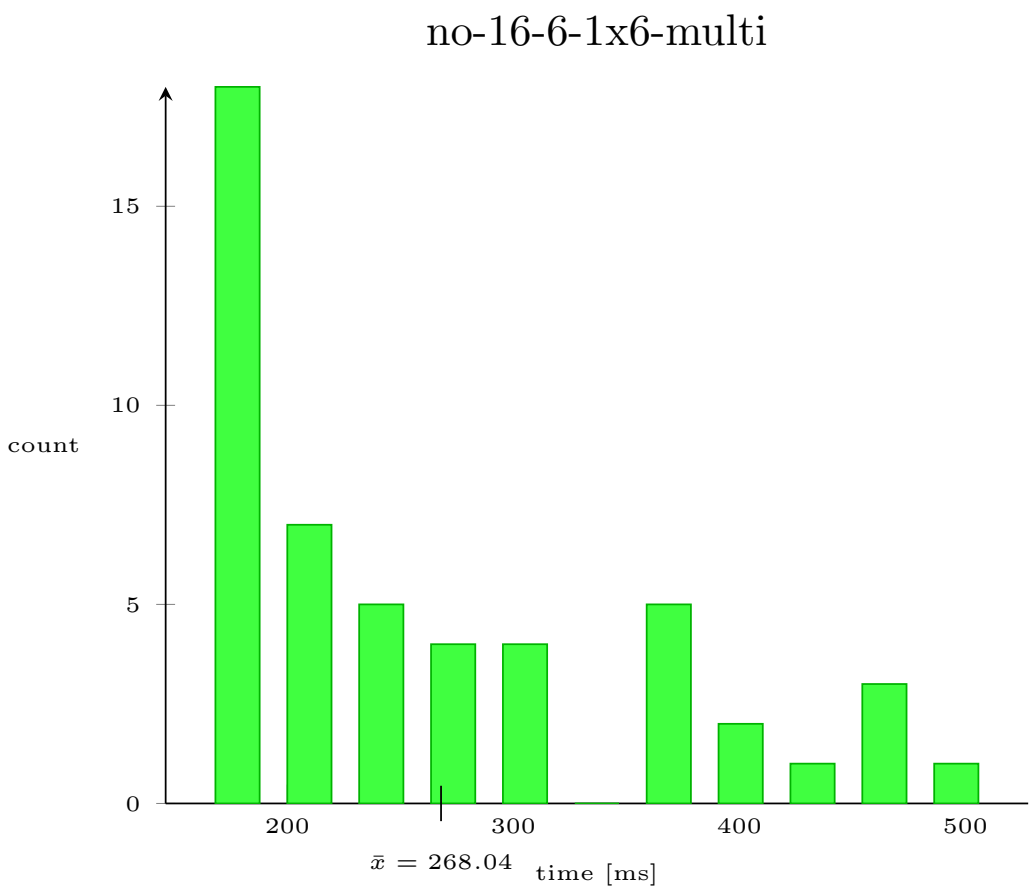


Figure 3: Multi processes on a small instance with no path.

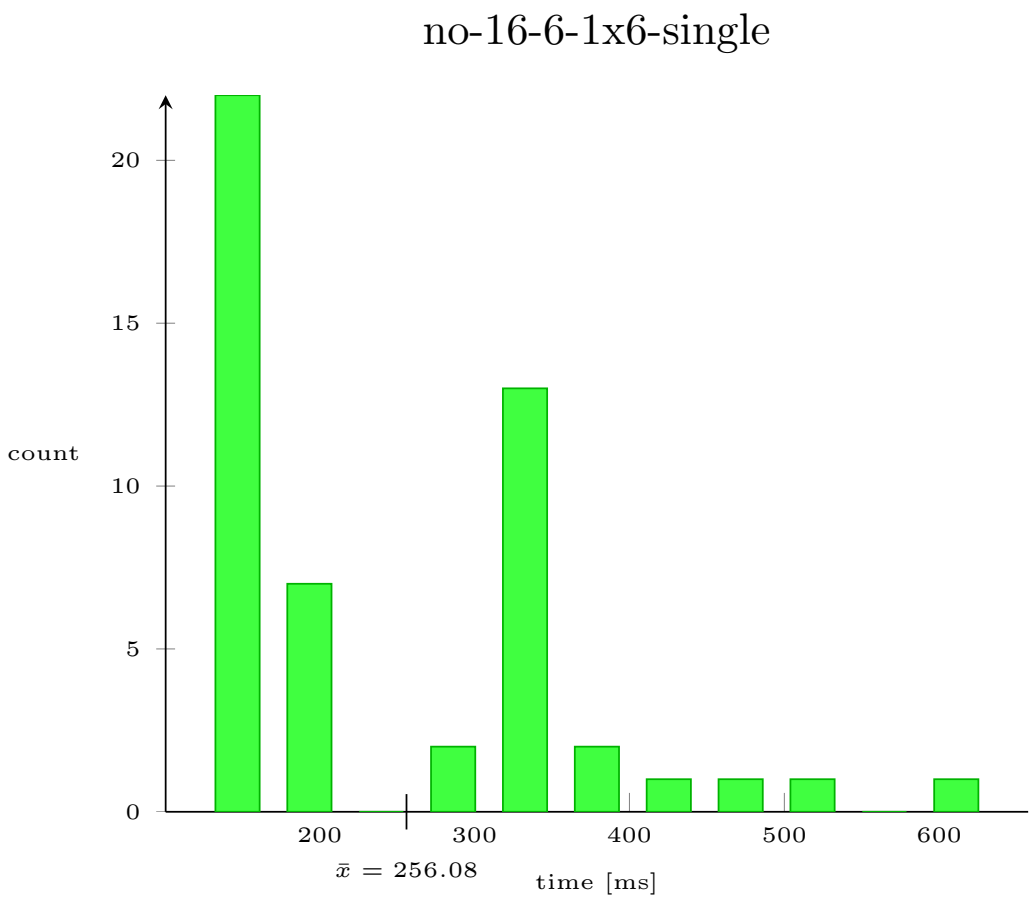


Figure 4: One process on a small instance with no path.

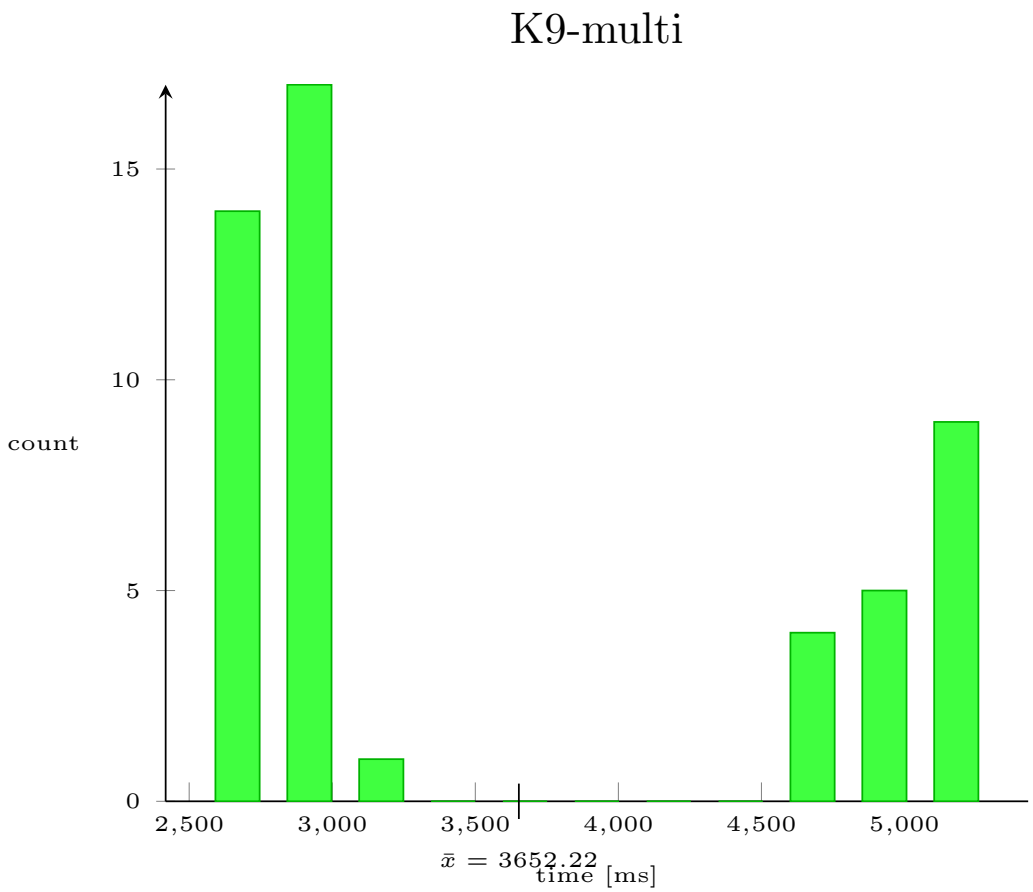


Figure 5: Two cluster of running time on complete graph with no path.

T-61.6050: Project report
Sparse classification using (Deep) Belief Network

Géraud Le Falher GERAUD.LEFALHER@AALTO.FI

Abstract

In this project, I use Deep Belief Network to perform a classification task, namely predict the period of the week at which Flickr photos were taken given their user supplied tags. I find out that by using a single layer, it performs almost as well as other methods but provides a more interpretable model.

1. Introduction

In another course I extracted metadata from photos shared on the Flickr[1] website. Theses photos were all taken in San Francisco since 2008 and contain, among others, the following information: the time and place were the photo was shot and a list of free form tags chosen by the user (that are normalized as lower case words). The goal in that course was to exploit this dataset to describe the geography of the city, either as whole or, given a specific region, find which tags provide a good description of it. A thorough study of this problem can be found in [5].

In this project, I instead wanted to make use of the temporal information by predicting the time at which each photos was taken given its tags. In the general case, this is a regression problem but to make it easier (and because it is not really interesting to know whether it was taken at 8:23 or 8:31), I divided the day into four intervals and the week into two, which result in the eight classes shown in Table 1.

days	0 to 6	6 to 12	12 to 18	18 to 24
Mon—Fri	0	1	2	3
Sat, Sun	4	5	6	7

Table 1: The eight time classes. It reads as follow: photos of class 6 were taken during the week-end between 12am and 6pm.

Another way of reducing the size of the problem was to not consider all tags. Indeed, the dataset contains more that 140,000 unique tags, but some of them are used rarely, only by a few users or over short period of time. Therefore, I kept only those that were used at least 150 times, by at least 25 users and over at least 500 days. Then, for each of the 1,874 tags left, I computed their entropy with respect to their frequency in each class and I removed those above a threshold, as I conjectured they were too generic to be informative (for instance, sanfrancisco, clouds, phone and love were discarded by this process). Finally, I kept only photos that had at least two tags and I put them in a n by d binary matrix where $n = 68859$, $d = 825$ and each row represents a photo, described by a binary vector indicating for each tag whether it appears in the photo or not.

[1] http://www.flickr.com

2. Methods

Because we are left with a standard classification problem, I tried several baseline methods to compare with DBN. Namely:

Naïve Bayes a model that assume all features are independent and come from a binomial distribution. Parameters are simply maximum likelihood estimates and it is often use for bag of words model.

Logistic Regression a discriminative model that estimates the probability to belong to one class and is trained by maximising likelihood.

k-Nearest Neighbors a natural procedure that chooses the class of the majority of the k closest training examples. I used Hamming distance as an approximation of the Euclidean norm because most coordinates are zero.

SVM a linear maximum margin separator. I used LIBSVM implementation of C-SVM [2], which train pairwise classifiers with Gaussian kernel[2].

The DBN is made of several layers of Restricted Boltzmann Machine trained with one step of contrastive divergence and classification was performed by logistic regression on top of it. I used code written by the authors of the Python Theano library [1].

3. Results

3.1. Performance

The performance of the baseline methods are shown in Table 2 on the next page. The main comment is that it seems to be a difficult problem, as the best accuracy, achieved by SVM, is only 69.5%.

However the result of the DBN were no better. We can see on Figure 1 on the following page that with 2 layers, regardless of their size, the error rate is around 35% (except for two anomalous points). I then decided to use only one layer and the error dropped by a few percent although it had a rather erratic behavior. The best result, obtained with 1,175 hidden units and an adapted learning rate, was 32.35%. Yet I used a smaller model (with only 158 hidden units) to interpret this outcome.

[2] $K(x, y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right)$

method	error	time (seconds)	parameters
Naïve Bayes	44.56%	0.9	—
Logistic Regression	42.65%	6.4	$\lambda = 10$
k-NN	39.81%	1440	$k = 5$, Hamming distance
SVM	31.53%	960	$C = 4.02$, $\sigma^2 = 0.97$

Table 2: Error rate and running time of the four baseline methods (along with the choice of parameters with which they were obtained).

3.2. Interpretation

My hope by using a small first layer was that the DBN would be able to perform some kind of clustering over the tags, because many of them share a similar semantic with others. To verify this hypothesis, I selected, for each hidden units, the 10 tags that have the largest weights connection. Relevant examples are shown in Table 3. The unit 6 refers to same sex marriage[3], the 35 to some kind of festivity and the 95 to "Bay to Breakers", a race organized in May in which many participants wear costume.

Since, RBM is a generative model, I also tried to produce some new photos but the tags I got did not let me draw any relevant conclusion.

4. Discussion

Deep Belief Network was the slowest approach (even with one layer it took more than 40 minutes to train and predict) and only the second best performer, a bit behind SVM. At first, I thought that maybe I made an erroneous modification to the code so I tried it with the same parameters and three layers of 500 units over the MNIST dataset, because it has similar dimensions. In that case, the log likelihood of the data was indeed decreasing consistently in each layer and it achieved an accuracy of 1.5%, the expected number in that situation. Therefore I think that the main issue lies in the sparsity of the data, because most photos have only two or three tags. I did not have time to conduct a comprehensive literature review but the papers I found suggest that instead of Gibbs sampling, it may be better to use Metropolis-Hasting [3] or Importance sampling [4], although they are mainly referring to faster running time.

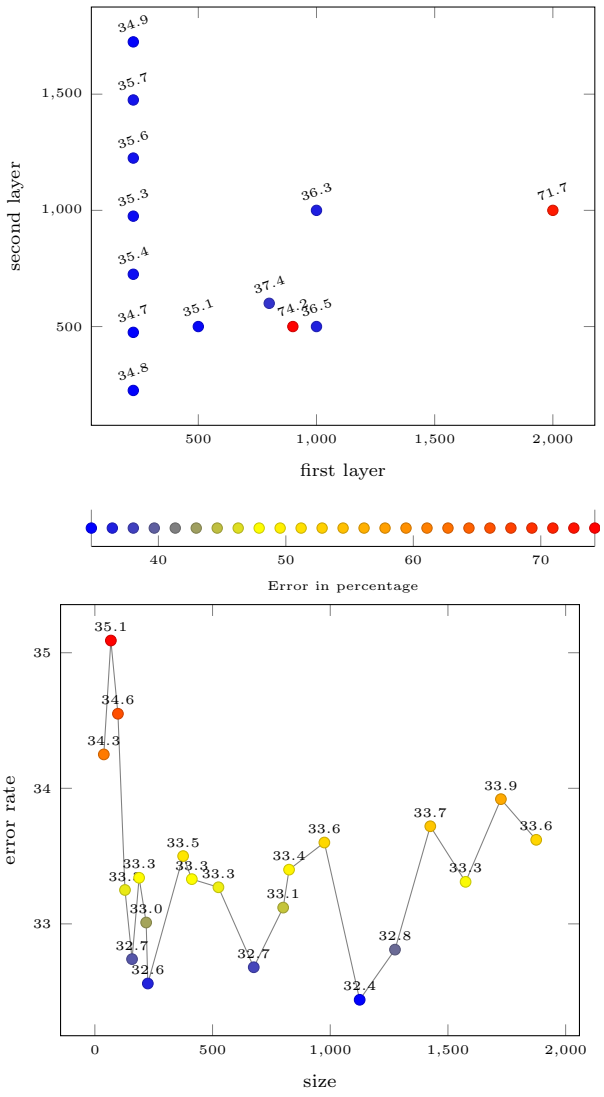


Figure 1: Error rate of DBN as a function of its two (top) or single (bottom) layers size.

[3]Proposition 8 was a law prohibiting such marriage and was later declared unconstitutional.

unit	tags
6	prop8, gaymarriage, equality, marriage, proposition8, court, valentinesday
35	parade, cosplay, pride, sfpride, parks, event, celebration, carnaval
95	baytobreakers, race, parade, costumes, team, racing, santa, college

Table 3: Top-weight tags for some hidden units.

References

[1] Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. URL http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf.

[2] Chang, Chih-Chung and Lin, Chih-Jen. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[3] Dahl, George, Adams, Ryan, and Larochelle, Hugo. Training restricted boltzmann machines on word observations. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 679–686, July 2012. ISBN 978-1-4503-1285-1. URL http://arxiv.org/abs/1202.5695.

[4] Dauphin, Yann and Bengio, Yoshua. Stochastic ratio matching of rbms for sparse high-dimensional inputs. In *Advances in Neural Information Processing Systems 26*, pp. 1340–1348. 2013. URL http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips26/687.pdf.

[5] Rattenbury, Tye and Naaman, Mor. Methods for extracting place semantics from Flickr tags. *ACM Transactions on the Web*, 3(1):1–30, January 2009. ISSN 15591131. URL http://infolab.stanford.edu/~mor/research/RattenburyPlacesSemanticsTweb09.pdf.

Fitting a graph to vector data. T-79.7003 Project paper

Le Falher Géraud Sanja Šćepanović Luiza Sayfullina

December 11, 2013

The purpose of this report is to review the paper "Fitting a graph to vector data". It describes how vector data can effectively be fitted to a graph form in order to solve classification, regression and clustering problems. We'll provide open questions that were left to the reader regarding construction of the graph and make analysis of experimental results.

1 INTRODUCTION

A lot of research has been conducted in using graph representation of a vector data in order to solve various problems. Spectral clustering is one of the examples, where the eigenvectors of the Laplacian matrix are used. Laplacian matrix is built using the Gaussian weights between the vectors. In this paper, a new and more sophisticated method for constructing graph from the set of vectors is presented. The distance metrics used in this approach shows promising results for classification problems, although it has limitations with respect to the sample size and the number of features.

2 CONSTRUCTING THE GRAPH

We are given a set of vectors $x_1, x_2, \ldots, x_n$, where $x_i \in \mathbb{R}^d$. The corresponding graph should have such property that close points should have edges. If the graph is weighted then the way of assigning the weights should be chosen. The common ways of assigning the weights are:

- Fixing a threshold σ and set $w_{i,j} = 1$ if $||x_i - x_j|| \leq \sigma$, 0 otherwise.

- Gaussian distance $w_{i,j} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$
- Euclidean distance $w_{i,j} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||$

In the paper, a more sophisticated approach is presented. For each data point, a vertex is added. Each vertex has weighted degree $d_i = \sum_j w_{i,j}$ and weights between the vertices $w_{i,j}$ are constructed in order to minimize the following function:

$$f(w) = \sum_i ||d_i \boldsymbol{x}_i - \sum_j w_{i,j} \boldsymbol{x}_j||^2 \tag{1}$$

The idea is to minimize the average weighted distance of the point to its neighbors. If all weights are one then we compare the point with its average value of the neighbors. However weights bring flexibility to the neighboring nodes and allow them to differ in different directions from the point. It seems more flexible way of constructing graph.

The intuition behind (1) is the following. Suppose that there is a function $g : \mathbb{R}^d \to \mathbb{R}$ that assigns value to vectors as $y_i = g(\boldsymbol{x}_i)$ and that we know both the weighted graph and the label of all vectors but one, $\boldsymbol{y}_i$. One natural guess is indeed a weighted average of all its neighbours:

$$\hat{\boldsymbol{y}}_i = \frac{\sum_j w_{ij} \boldsymbol{x}_j}{\sum_j w_{ij}} = \frac{1}{d_i} \sum_j w_{ij} g(\boldsymbol{x}_j)$$

and we could measure the quality of this graph by the sum of the square of errors: $\mathcal{E}_g = \sum_i \left(y_i - \frac{1}{d_i} \sum_j w_{ij} g(\boldsymbol{x}_j)\right)^2$

But we do not know the labels[1] so we use our guess instead. We also do not know g, hence we replace it by a family $\mathcal{G}$ of simple functions, coordinate wise identity for instance (that is, $g^{(k)}(\boldsymbol{x}_i) = \boldsymbol{x}_{i,k}$) and we minimize the sum of all errors:

$$\begin{aligned}
\sum_{g \in \mathcal{G}} \mathcal{E}_g &= \sum_{k=1}^{d} \sum_i \left(g^{(k)}(\boldsymbol{x}_i) - \frac{1}{d_i} \sum_j w_{ij} g^{(k)}(\boldsymbol{x}_j) \right)^2 \\
&= \sum_{k=1}^{d} \sum_i \left(\boldsymbol{x}_{i,k} - \frac{1}{d_i} \sum_j w_{ij} \boldsymbol{x}_{j,k} \right)^2 \\
&= \sum_i ||\boldsymbol{x}_i - \frac{1}{d_i} \sum_j w_{ij} \boldsymbol{x}_j||^2
\end{aligned}$$

1 at least not all of them.

Because $\frac{w_{ij}}{d_i}$ leads to non-convex problem, we multiply every term by $d_i^{-\frac{1}{2}}$ to get (1). At this point, we ask ourselves some questions about the family $\mathcal{G}$. Could we weight each of its function according to some prior knowledge about which features are important? Could we use more that d functions, for instance $g^{(k)}(\boldsymbol{x}_i) = \boldsymbol{x}_{i,k}\boldsymbol{x}_{i,k+1}$? And in supervised classification problem, should not one of the function use the label information[2]?

To avoid the zero solution, the authors introduce a constraint on the weighted degrees, namely $d_i > 1$ and call graphs constructed by this way hard graph. It seems to us that hard version of the graph is not the best solution for some datasets. The reason is that the level of connectivity depends on how are the clusters formed. If there are many outliers then some of the nodes are isolated and we should provide the possibility to handle these cases.

α-soft graph is the relaxation of the hard graph, where total sum of the weighted degrees can be less than αn

$$\sum_i \left(\max\left(0, 1 - d_i\right)^2\right) \leq \alpha n \tag{2}$$

The choice of the α can make the algorithm faster, more flexible. We can see from classification results[3] that for 7 out of 9 datasets α-soft version with $\alpha = 0.1$ outperformed hard version in accuracy and time. Possibly simple clustering can give measure of how α should be chosen. If there are many small clusters then smaller α can be chosen.

3 CONCRETE IMPLEMENTATION

3.1 Hard graph

To find the weights of the hard graph, we need to minimize $f(\boldsymbol{w}) = \sum_i ||d_i\boldsymbol{x}_i - \sum_j w_{i,j}\boldsymbol{x}_j||^2$. If we let L be the Laplacian matrix of the graph and X the n by d matrix representing our set of vector, we can rewrite f as $f(\boldsymbol{w}) = ||LX||_F^2$ where this Frobenius norm is defined as $||M||_F^2 = \sum_{i,j} M_{i,j}^2$. L is linear in the weight and the problem is therefore quadratic. But for the purpose of numerical optimization, we would like to formulate it in term of Euclidean norm.

To do this, let introduce some notations. Over n vertices, there are $m = \binom{n}{2}$ possible edges. Let U be the signed n by m incidence matrix such that if edge $e = (i, j)$ exists, $U_{i,e} = 1$

2 It is straightforward because it must also returns a value for unknown instances.

3 Showed in Table 2 of the paper

and $U_{j,e} = -1$. If we define W to be the diagonal matrix of weights, we can write $L = UWU^T$. Furthermore, letting $\boldsymbol{x}^k$ be the k^{th} column of X, we define the auxiliary quantities $\boldsymbol{y}^{(k)} = U^T\boldsymbol{x}^k$ and $Y^{(k)}$, the matrix whose diagonal is $\boldsymbol{y}^{(k)}$ and zero elsewhere. Armed with this, we can rewrite the minimization problem as:

$$\begin{aligned} f(\boldsymbol{w}) &= \sum_i ||d_i\boldsymbol{x}_i - \sum_j w_{i,j}\boldsymbol{x}_j||^2 = ||LX||_F^2 \\ &= \sum_{k=1}^{d} ||L\boldsymbol{x}^k||^2 = \sum_{k=1}^{d} ||UWU^T\boldsymbol{x}_k||^2 \\ &= \sum_{k=1}^{d} ||UW\boldsymbol{y}^{(k)}||^2 = \sum_{k=1}^{d} ||UY^{(k)}w||^2 = ||Mw||^2 \end{aligned}$$

where M is defined by block as

$$M = \begin{pmatrix} UY^{(1)} \\ \vdots \\ UY^{(d)} \end{pmatrix}$$

M is a dn by $\binom{n}{2}$ matrix, meaning it has $\Theta(dn^3)$ elements. Therefore, solving this problem is potentially computationally heavy. Yet we hope that our vectors lie on some low dimensional manifold in a way that they do not have too many close neighbours. In fact, it is proved in Theorem 3.1 of the paper[4] that graphs found by this process have at most $d(n+1)$ edges (or equivalently average degree at most $2(d+1)$. We exploit the fact that solution are sparse build it incrementally.

Indeed, naively, we would construct U_K—the incidence matrix of the complete graph—, the corresponding M_K and then solve $\min_w f(\boldsymbol{w}) = ||M_K\boldsymbol{w}||^2$, subject to $d_i \geq 1$. Calling the Lagrange multipliers $\boldsymbol{z}$, the associated Lagrange function is:

$$\Lambda_K(\boldsymbol{w}, \boldsymbol{z}) = ||M_K\boldsymbol{w}||^2 - \boldsymbol{z}^T(A_K\boldsymbol{w} - 1)$$

where $A_K = |U_K|$ and $d = A_K w$.

Instead, we start by choosing a random subset of edge, build the corresponding U and M, and solve $\min_w ||M\boldsymbol{w}||^2$ using the `quadprog` function of MATLAB. If some edges weights become zero, we remove them. We then need a way to know whether we have found a feasible solution of the complete problem. Such solutions must satisfy the Karush-Khun-Tucker conditions. Especially, $\forall i,j\ \frac{\partial \Lambda_K}{\boldsymbol{w}_{i,j}} \geq 0$. Wherever it is not the case, we update U and M with these missing edges and solve again until we cannot add edges anymore.

4 the proof is presented in section 4 on page 6.

To gain some confidence in our code, we generate a small artificial dataset ($n = 22$ and $d = 3$) and solve the minimization problem either directly or with this iterative procedure. It produced weights vector $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ that were consistent in the sense that $\frac{||\boldsymbol{w}_2 - \boldsymbol{w}_1||}{||\boldsymbol{w}_1||}$ was less than 0.1%. In Figure 1 on page 9 is another illustration in dimension 2. Let say that the x axis is the age of individuals, y axis their income and that the color indicates their class: whether they will default on a loan[5]. The width of the edges represents their weights and we notice that, whereas there are two clusters, there are edges between them. Furthermore, one can prove that in dimension 2, the graph obtained is always planar.

3.2 α-soft graph

In the case of a soft graph, the constraint $A\boldsymbol{w} \geq 1$ is dropped in favor of $\eta(\boldsymbol{w}) = ||\max(\boldsymbol{0}, \boldsymbol{1} - A\boldsymbol{w})||^2 \leq \alpha n$ as introduced in (2). $\eta(\boldsymbol{w})$ indicates how much weighted degrees are smaller than 1 and because $f(\boldsymbol{w})$ is always improved by reducing all weights uniformly, a α-soft weights vector satisfy (2) with equality, that is $\eta(\boldsymbol{w}) = \alpha n$.

Let the optimization problem be

$$\min_{\boldsymbol{w}} \{f(\boldsymbol{w}) + \mu \cdot \eta(\boldsymbol{w}) \ : \ \boldsymbol{w} \geq \boldsymbol{0}\} \tag{3}$$

where $\mu \in \mathbb{R}^+$ controls how much emphasis we put on the degree constraint. When $\mu = 0$, we only optimize $f(\boldsymbol{w})$ and therefore set $\boldsymbol{w} = \boldsymbol{0}$. As μ goes to infinity, we are increasingly concerned by making sure that $A\boldsymbol{w} = \boldsymbol{1}$ (or in other words, $\eta(\boldsymbol{w})$ goes to zero). Furthermore, for any μ, if $\boldsymbol{w}$ is a solution to (3), it is a solution of an $\alpha' = \eta(\boldsymbol{w})/n$-soft graph. The strategy is then to fix an initial value of μ, solve (3) and adjuste μ to bring α' arbitrarily close to the α we want.

The paper then claimed that (3) is equivalent to minimizing $\mathcal{L}$ defined below and that is a non negative least square problem. But we only manage to write as another quadratic program:

$$\begin{aligned}
\mathcal{L} &= \min_{\boldsymbol{w},\boldsymbol{s}} ||M\boldsymbol{w}||^2 + \mu||\boldsymbol{1} - A\boldsymbol{w} + \boldsymbol{s}||^2 \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T(M^TM + \mu A^TA)\boldsymbol{w} + \mu \boldsymbol{s}^T I^T I \boldsymbol{s} + \mu \left(1 - 2\boldsymbol{1}^T A\boldsymbol{w} + 2\boldsymbol{1}^T\boldsymbol{s} - 2\boldsymbol{w}^T A^T \boldsymbol{s}\right) \\
&= \min_{\boldsymbol{y}} \boldsymbol{y}^T \begin{pmatrix} M^TM + \mu A^TA & -\mu A^T \\ -\mu A & \mu I \end{pmatrix} \boldsymbol{y} - 2 \begin{pmatrix} \mu \boldsymbol{1}^T A \\ -\mu \boldsymbol{1}^T \end{pmatrix} \boldsymbol{y} + \mu \boldsymbol{1}^T \boldsymbol{1} \\
&= \min_{\boldsymbol{y}} \boldsymbol{y}^T H \boldsymbol{y} + \boldsymbol{f}\boldsymbol{y} = \min_{\boldsymbol{y}} \Lambda(\boldsymbol{y})
\end{aligned}$$

5 As no legend is provided, the reader can choose the colors interpretation that best suits its own age.

where $\boldsymbol{y}$ is the $m+n$ vector $\begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{s} \end{pmatrix} \geq 0$.

As there are no constraint and thus no Lagrange multipliers:

$$\frac{d\Lambda}{d\boldsymbol{w}} = 2(M^T M + \mu A^T A)\boldsymbol{w} - 2\mu A^T(\boldsymbol{s} + \mathbf{1})$$

From this point, we are back in the previous case with only minor modification to the quadratic program and its derivative. We were thus expecting that our existing code would perform as well. Yet it was definitely not the case. The issue was that although many edges made the derivative negative, none of them were close enough to zero to be removed. But we cannot simply add them because of the sparsity constraint. At the end, we renounced to compute α graph, even if the paper suggests that running would have been greatly improved.

4 SPARSITY OF THE FITTED GRAPH

The authors find examples of sets of vectors for which the fitted graph solution is not unique (this happens when there exists certain symmetry among the vectors). However, no matter on the uniqueness, they prove that all of the graph solutions as defined will be sparse. More precisely, the theorem states that for any set of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ there exists a solution to both of the hard and soft constraints that has no more than $n(d+1)$ edges, which implies sparsity whenever d is small enough compared with n, as it is the case in many machine learning applications[6].

The proof is by contradiction. Let us recall that we are minimizing a quadratic program of the form $||Mw||^2$, and let us assume that we found a solution $\boldsymbol{w}$. If we assume that the number of edges in a hard or α-soft graph fitted to the given vectors is greater than $n(d+1)$, say q, then there exists a vector of weights $\boldsymbol{w}^*$ such that $\begin{bmatrix} M \\ A \end{bmatrix} \boldsymbol{w}^* = 0$ and $\boldsymbol{w}^*$ has non-zero values restricted to these q edges.

This, however, creates the contradiction, since using such a vector $\boldsymbol{w}^*$, we can now create a weights vector, that is also solution to the graph problem, but will have less edges. Namely, we can take $\boldsymbol{w}' > 0$ by finding suitable $r \in \mathbb{R}$ such that $\boldsymbol{w}' = (\boldsymbol{w} + r\boldsymbol{w}^*)$ and at least one of the edges from $\boldsymbol{w}$ is zeroed in $\boldsymbol{w}'$. Such $\boldsymbol{w}'$ satisfies our hard quadratic problem, since $Mw' = M(\boldsymbol{w} + r\boldsymbol{w}^*) = Mw + rMw^* = Mw$. Moreover, since $Aw' = A(\boldsymbol{w} + r\boldsymbol{w}^*) = Aw + rAw^* = Aw$, it also satisfies any degree constraints.

6 But not always. For instance, genomic dataset often have few samples with thousand or even million of features.

Thus, the solution as defined can have at most $n(d+1)$.

5 EXPERIMENTAL RESULTS

Constructed graph can be used in solving classification, regression and clustering problems. After calculating the Laplacian of the constructed graph, we can solve classification problem of the following form:

$$x^T L x = \sum_{i,j \in E} w_{i,j}(x_i - x_j)^2 \tag{4}$$

$(x_i)_{i \in \{1,\dots,n\}}$ denotes classes of instances, some of them which are known. The idea is to set the same labels to the close points, by solving another quadratic problem and hence minimizing the energy dissipated by the graph seen as an electrical network. This is the formula for binary classification task. It seems that there should be some limit on the number of training and test samples. Some of the clustering points can have small proportion of labels that can lead to low accuracy. Hard clustering can potentially alleviate this problem, because it can make bigger clusters where unlabeled points will have more labeled neighbours.

Classification experiments were held for benchmark datasets with small amount of dimensions and sample size. Results were compared with SVM, Full Bayes Classifier, Hill Climbing Learning Algorithm, KNN, Averaged One Dependence Estimator. For many of the datasets the performance was on the best in case of accuracy. For two datasets Ionosphere and Sonar the accuracy was the highest. Actually the dimensions of these two datasets were the highest. In such cases, simple Euclidean distance suffers from the curse of dimensionality whereas this type of distance, which takes into account multiple points, has advantages. Results on our own four datasets are shown in Table 1.

Dataset	n	d	SVM	Graph	Time (seconds)
Liver disorders	172	8	60.58 %	**68.00** %	276
Diabetes	300	10	62.25 %	**72.33** %	3423
Breast cancer	300	12	55.58 %	**96.60** %	15499
Twitter	300	13	**82.50** %	72.66 %	1855

Table 1: Accuracy and training time of this method on four dataset compared with SVM

As we can see the accuracy for 3 out of 4 datasets was better than SVM. Breast cancer dataset showed big improvement in the accuracy. There is a huge gap between the values

in one feature and the others. At the same time twitter dataset is nearly binary and the performance of SVM is better. We can't compare SVM and Hard-graph method in sense of the computational time. Moreover the biggest deficiency of the method is in the limitations on the sample and feature size. As we saw previously, the amount of potential edges grows as $O(n^2)$, then the number of variables increases fast. Dimensions on the data is restricted as well.

6 CONCLUSION

Our project deals with the paper on fitting a graph to a set of vectors. The purpose of such an idea is both theoretical and practical. From theoretical side, it is of interest to find a graph that exhibits nice combinatorial properties, like sparsity or planarity in two dimensions. Being able to assign such a graph to any set of vectors provides insights about the vector-set. From practical side, working with the fitted graph can help solving machine learning problems on the given vector set. In particular, we have experimented with classification. The results are promising, having better accuracy compared to other methods in many cases. This approach has the further advantage of not requiring any parameter tuning.

However, described approach of fitting graph to vectors is not ideal by any means. The authors leave many questions open with regards to the properties of such graph. For example, uniqueness is not guaranteed, although the authors conjecture that it holds with high probability up to a small perturbation to break potential symmetry. More importantly, labels are not incorporated into building of this kind of graphs. One issue inherent to the approach is that adding a single node requires recomputing the whole graph, a disadvantage compared to some of the existing approaches. We also discussed the disadvantages in regard to the sample size and the number of features.

Nevertheless, as we also tested in part, this approach can be practical when solving certain types of classification, regression and clustering problems.

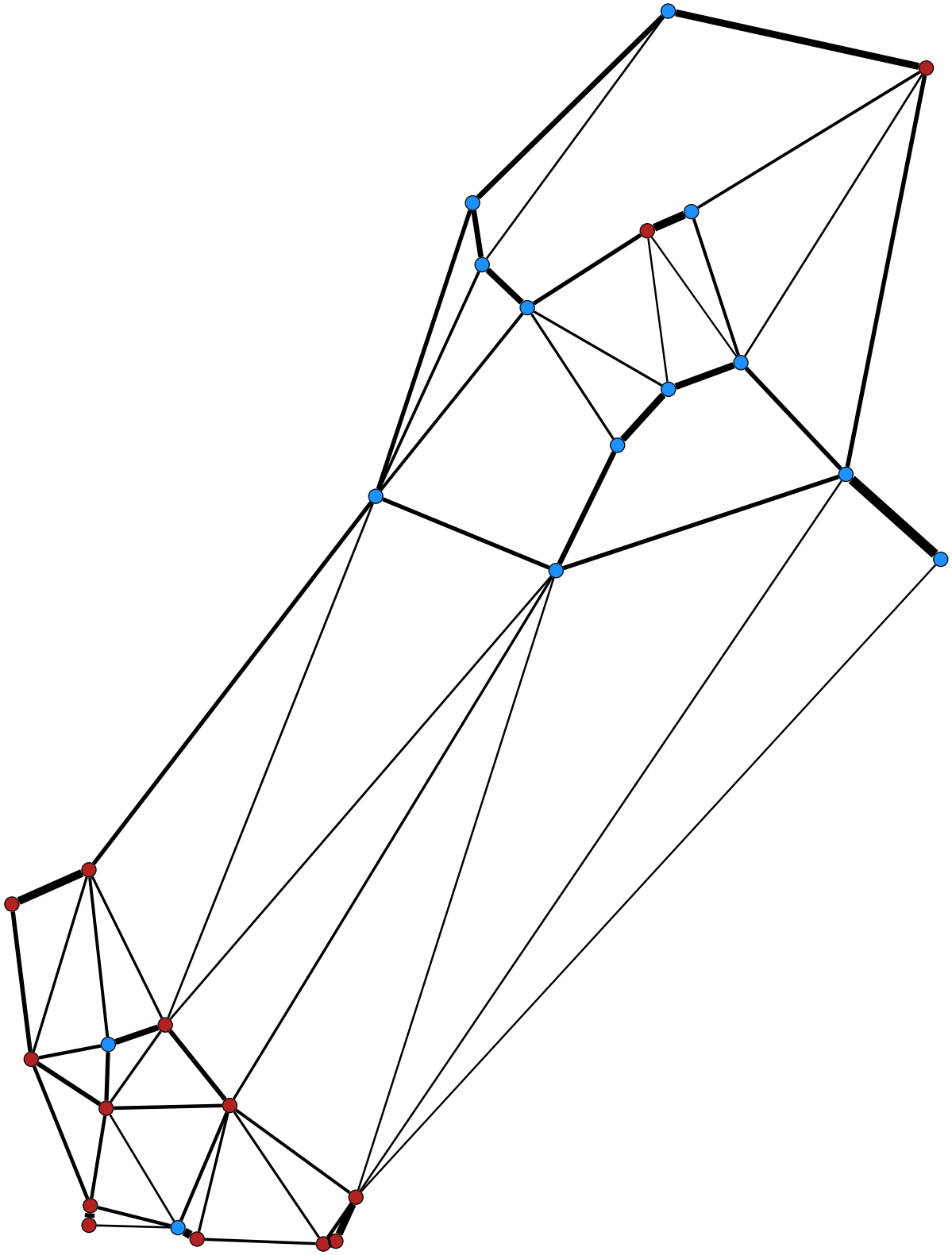


Figure 1: Small example of hard graph construction.

AALTO UNIVERSITY

T-61.5910 RESEARCH PROJECT

FINAL REPORT

Social Map Annotation

Author:
Géraud Le Falher
geraud.lefalher@aalto.fi

Supervisors:
Aristides Gionis
aristides.gionis@aalto.fi

Michael Mathioudakis
michael.mathioudakis@hiit.fi

January 15, 2014

Aalto University
School of Science

The rise of location aware devices means that an increasing portion of online user generated content carries geographical information. We describe how these spatial information can be combined with the traditional tagging system to discover meaningful relations. Extracting metadata of localized photos from Flickr website, we indeed try to uncover association rules between locations and tags. This allows us to tackle three questions: find where a given tag appears significantly, select tags that best characterize a given location and retrieve which locations generate the more interest. We provide answers for the city of San Francisco by filtering tags and computing spatial statistics. Yet much remains to be done: validate these results, be less prescriptive but more data driven and extend to different scales and others dimensions (time, users).

1 INTRODUCTION

A common way of organizing information on websites it to rely on tags provided by users. Although these words are chosen freely, we do not expect them to be arbitrary and thus hope to extract limited semantics from them. In this work, we focus on tags associated with geolocated photos extracted from Flickr[1]. Using this crowdsourced set of (tags, location) pairs, we investigate three problems:

- Given a tag, find the places in a city where it is significantly concentrated.
- Conversely, given a location, find the tags that best describe it compared with other locations of similar scale.
- Finally, find those places that generate the most interest.

The first answer could be applied in a touristic context. When arriving in a new city, a person interested in baseball or streetart could find relevant places according to the experience of inhabitants and previous visitors. The second answer could be used by Flickr to suggest relevant tags when users upload photos. Consequently, more sensibly tagged photos allow more relevant search results and improve the website user experience. The last answer could for instance be the first step of an automatic travel guide that select points of interest in a city or a region.

Other applications may include inferring missing information (like location or time of the day) from the tags or create tailored visualisation.

2 SETTING

2.1 Description of the Dataset

Using the Flickr API, we downloaded metadata from every photo satisfying a set of criteria: they contained at least one tag, they were located, they have been uploaded after January 1st, 2008 and they belonged to a predefined rectangular region. Most of the work was done on a set of around 780 000 photos in the city of San Francisco, but we also got data from a part of California[2] and over the whole United States.

More precisely, in addition to tags and location, we know when each photos was taken and uploaded, by which user and what title was given to it (the title was not used except when it contained hashtags, which were converted to tags). Thus a typical data point looks like this:

1 http://www.flickr.com/
2 from San José to Reno

```
loc     : [-122.392501, 37.77515],
taken   : "2008-03-24 14:55:40",
user_id : "37417902@N00",
tags    : ["sanfrancisco", "california", "bridge", "chinabasin"]
title   : "sf 4th st bridge 8"
```

2.2 Uncertainty of the data

This retrieval process was naturally not perfect. In addition to some API calls returning strange results, the casual nature of the data explains their inherent noise.

- User id are subject to caution since nothing prevents people to upload photos on behalf of others. It would require serious effort to detect it but one may expect it is rather uncommon. Moreover, the mere fact that the upload take place still denotes a relation between the user and the photos.
- While timestamp issued by mobile phones are likely to be correct, as their internal clock is synchronized by internet, this may not always be the case for dedicated cameras. More concerning than usual drift of low quality clock is the situation of tourists coming from different timezone. Yet as I could not think of any simple solution to that problem, I just ignored it and carried on.
- To ensure the quality of the localization, I restricted myself to photos whose precision is deemed "street level" by Flickr. The potential problem is that it would cost an extra request to know whether this location was given by GPS (in which case the camera position is accurate) or by the user at upload time. In the latter case, in addition to the general imprecision of the method, it is ambiguous whether this location refer the place where the photo was shot or the position of the photo's subject[3].
- Finally, without additional request, the tags obtained are those normalized by Flickr. This normalization is not bijective but it is assumed that two tags with the same normalized form were close in the first place.

Overall, these restrictions are not really problematic. Yet there is another one that is not specific to a given field. Users have the possibility to upload photos by batch and assign them common location and tags. In some case, this could skew the corresponding distributions. Take the tag 14thstreet as an example. One user have uploaded more than 3 500 photos at one corner street during a marathon whereas only a handful of others users have employed this tag, which is therefore not as popular as the raw number would suggest. To alleviate this situation, we performed the following preprocessing step.

First, we duplicated the "tags" field of every photos. Then, for each user u and each tag t, we computed the distribution of photos tagged t by u and removed the tags from the photos that

3 Think of a bridge taken from a nearby hill.

appeared more than $T = 120$ times in the same place (the same cell of the 200 discrete grid) in the same time (2 weeks interval). With such threshold, it removed around 20% of all tags and it somewhat modified the list of top tags but with no clear pattern. Yet tags with very low entropy like 14thstreet did not appear anymore because they lost most of their support. A better way to deal with that issue could be to weight tags by the number of their users but it would computationally more expensive.

2.3 Statistical exploration

The first thing was to look at the tags to get a sense of them. In San Francisco after the preprocessing, there was a total of 4 977 625 occurrences of the 145 242 unique tags. But their distribution varies widely, between the most popular one, sanfrancisco, used 373 427 times and the 101 361 one that are used less than 5 times. Some of them are shown in Table 1 but a more synthetic visualization is presented in Figure 1a on page 7, where we can see that like words in a written documents, tags follows a power law.

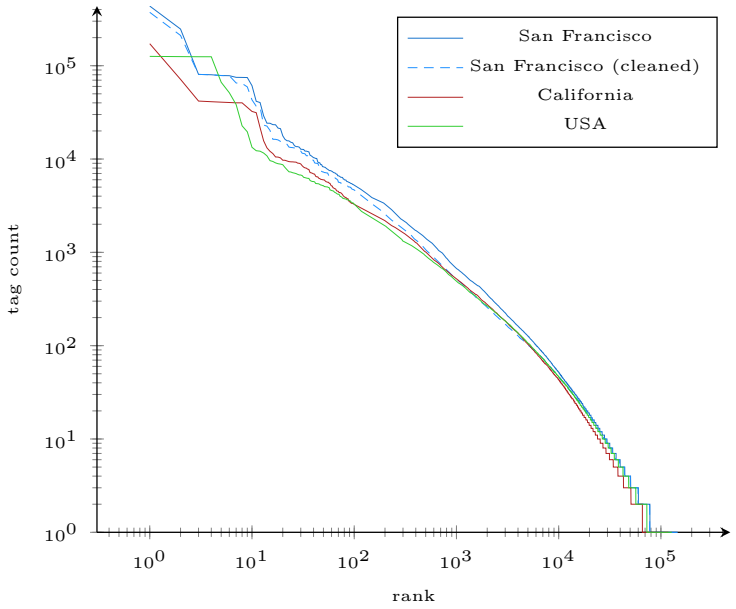
First 15 tags	between 100 and 1000	after 90 000
sanfrancisco	2013	sfgiantsfan
california	pacific	rolexbigboatseries
iphoneography	february	proshowgold
square	foundinsf	neutraface
squareformat	dolorespark	natur
instagramapp	japaneseteagarden	lusty
unitedstates	boat	lightousetender
sf	5k	jennyholzer
usa	national	img0562jpg
ca		djguyruben
san	cruise	cutebaby
francisco	above	cardamine
goldengatepark	july2009	californiaproduce
2010	effortlesslyuploadedbymyeyeficard	aroundwithb1
iphone	dayofdecision	aquateenhungerforcemooninite

Table 1: A sample of San Francisco tags, depending of their rank.

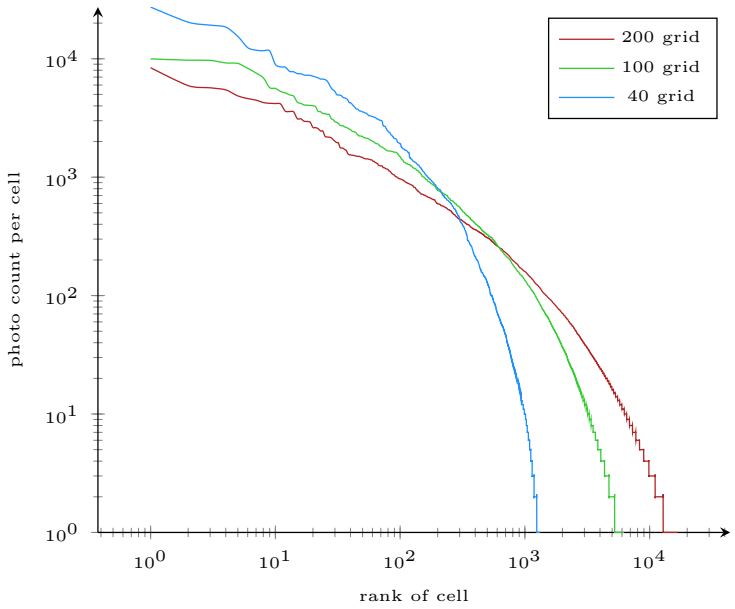
After making these observations, we decided to consider only tags with enough support, both to ease the computational effort and to avoid outliers. For each tags, we computed three simple metrics: total count, distinct users count and time span. Using the three thresholds (150 photos, 25 users, 500 days), we kept only 1 874 tags. It may seem quite restrictive but they still cover 68.6% of all occurrences and these thresholds can be changed later[4].

4 For instance, with $(20, 2, 0)$, we get 12 959 tags and 84.4% coverage.

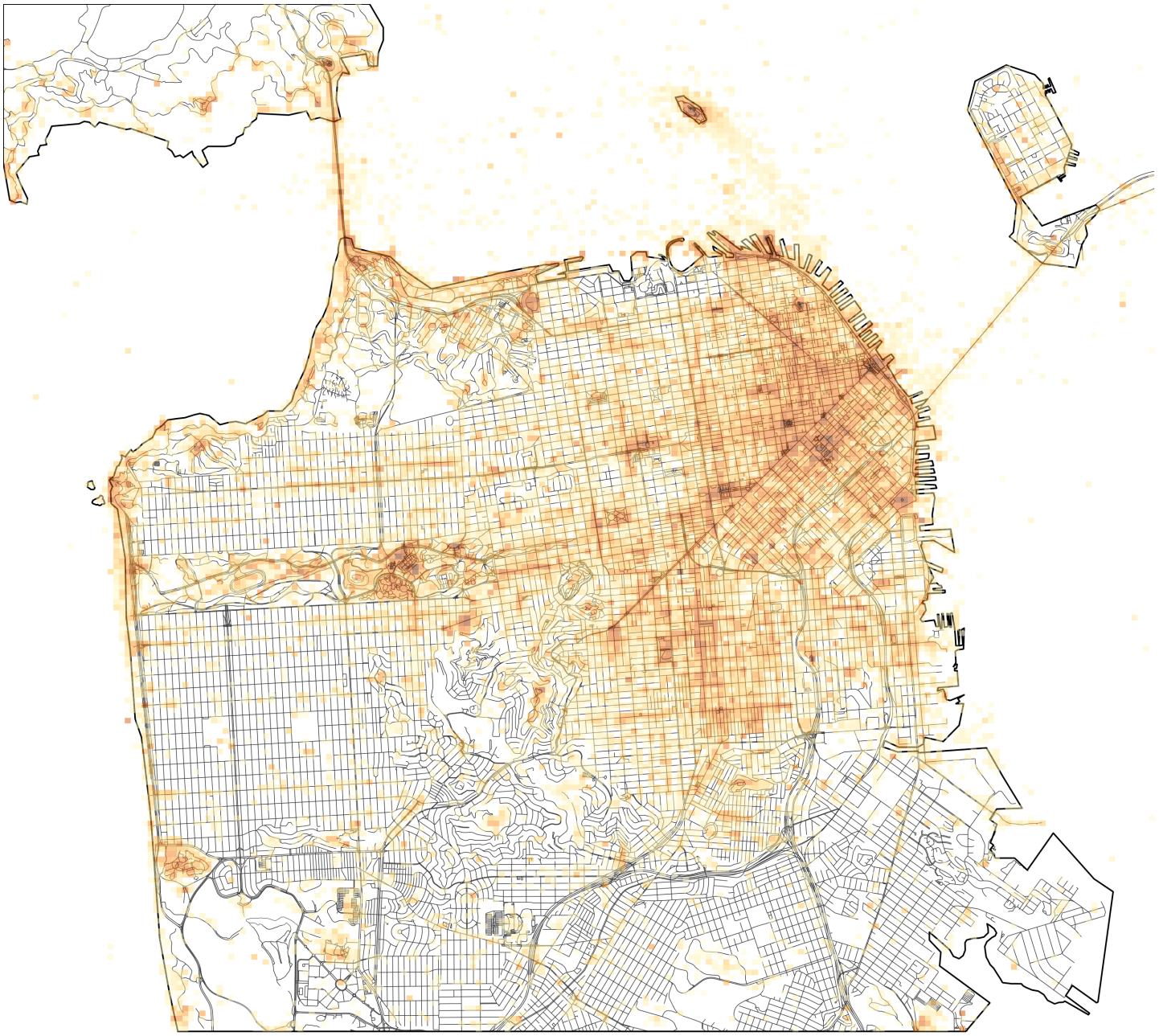
We can then conduct a similar analysis over the locations in which photos appear. Because of their large number, it was not convenient nor readable to display them individually. Therefore, we discretized space as a regular grid of size 200 by 200. A natural way of visualizing them is to draw a heatmap (Figure 1c on the following page). We notice again that they are far from being uniformly distributed and that some neighborhoods are more popular than others. More quantitatively, plotting number of photos of each location as a function of their rank (Figure 1b on the next page), we notice that it first follows a power law and after some point, a more abrupt one. Moreover, the same phenomena occurs for other grid size, albeit with different α coefficients. Despite this similar behavior, it was more tricky to explicitly exclude parts of the city.



(a) Tag distribution log-log scale over three regions of different scale.



(b) Spatial distribution of photos over three grids with different granularity.



(c) Photos count in logarithmic scale (the darker, the more photos).

Figure 1: Various ways of visualizing content distribution.

Let us return to one of our original question, find which tags describe a given location. The first approach would to filter this 1 874 tags to keep only those that are enough concentrated at one position and reject those that too uniformly distributed. After that, it would simply be a matter of returning those that appear in the place of interest. An example of this two kind of tags are museum and street. As shown in Figure 2, museum photos are mostly located around five or six points whereas street is more diffuse. But instead of looking at a map, we want a numerical statistic that allow us to distinguish between this two cases.

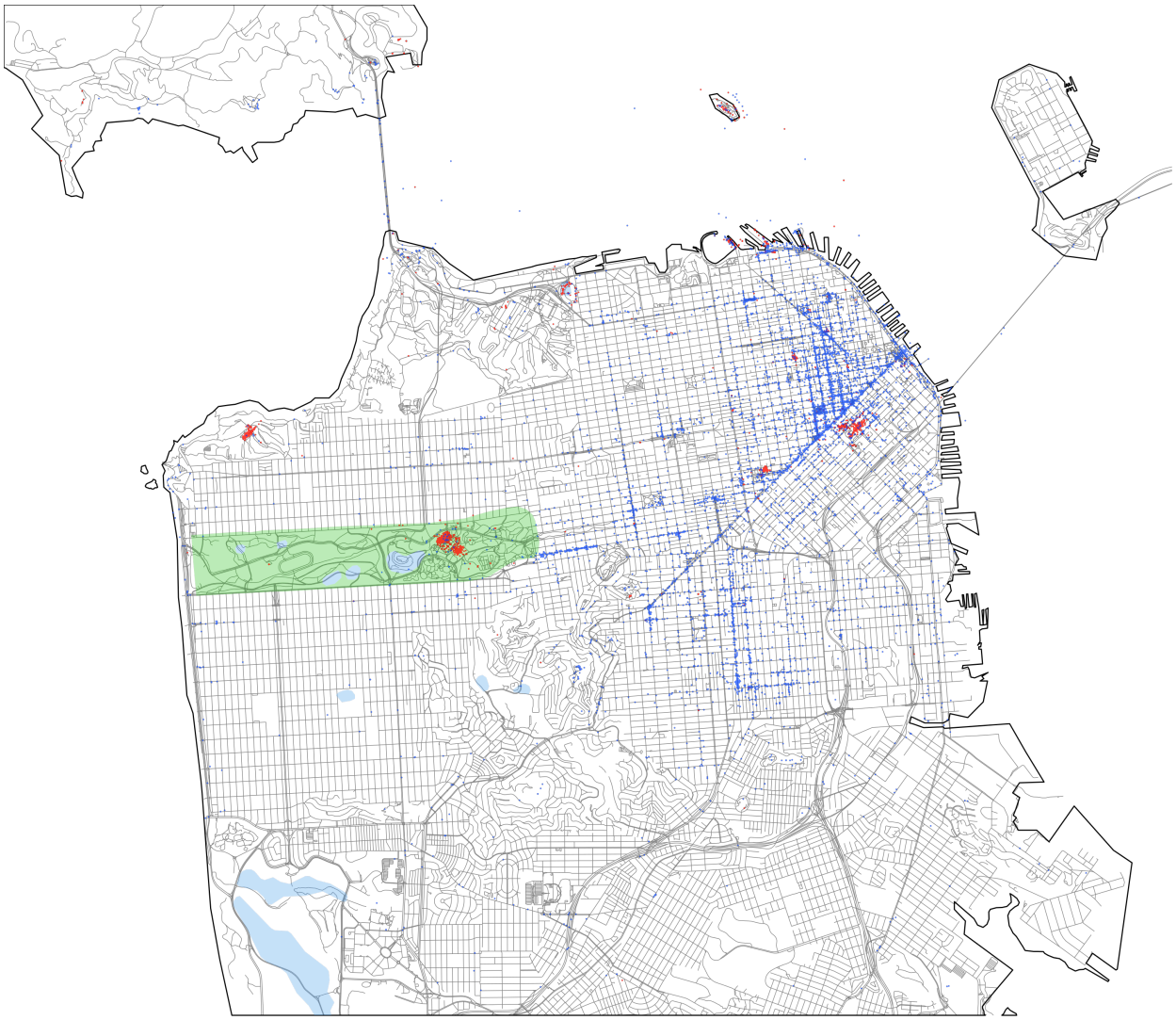
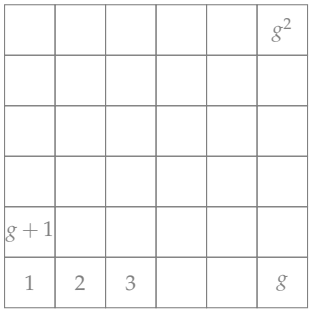


Figure 2: Red dots denote photos tagged museum while blue ones are street.

3 METHODS

Let first define some notation. As mentioned before, the city is divided in a $g \times g$ grid made of rectangular cells 1 through g^2 (like pictured on the right). For a cell i, let $b(i)$ be the total number of photos in i^{th} cell [5], $B = \sum_i b(i)$ the total number of photos and $b_f(i) = \frac{b(i)}{B}$ the frequency of each cell. For a tag t, we define in the same manner $t(i)$, T and $t_f(i)$, this time considering only photos which have tags t.



With these frequencies we can compute entropy. Let

$$H(t,g) = -\frac{1}{2\log g}\sum_{i=1}^{g^2} t_f(i)\log t_f(i)$$

be the entropy of tag t on a grid of size g. The normalization factor ensures that regardless of g, the values will range from 0 (all photos in the same cell) to 1 (uniform distribution). Tags with extremal values are presented in Table 2 on page 11 for three grids: 200×200 (each cell is 80 by 70 meters long, less than a block), 80×80 (200×180 meters, slightly more that a block) and 20×20 (800×700 meters, maybe a district). We notice that the lowest values relate to specific location like museum while highest one are generic. Whereas $g = 200$ and $g = 80$ yield comparable result, it is no more the case for $g = 20$. In particular high valued tags become even more generic (sky, dog, ...).

To further differentiate tags that are highly concentrated, we can compute the Kullback Leibler divergence of their distribution with the one of all the photos. For that, we define

$$D_g(t||b) = \frac{-1}{\log \min_{b_f(i)>0} b_f(i)} \sum_{i=1, b_f(i)>0}^{g^2} t_f(i)\log\frac{t_f(i)}{b_f(i)}$$

This time, values range from 1 (when all the t photos are in the cell where there are the fewest total photos and thus are maximally distinct) to 0 (for $t = b$, which is not possible). Again extremal values are shown in Table 3 on page 12. It also differentiates between the two kinds of tag but compared with entropy, it seems more robust to change in the grid dimensions as the three rows shows roughly the same set of tags. Weirdly enough, there is also a semantic shift for position specific tags. Whereas those picked by entropy were mostly buildings, here they relate more to recreational areas like parks.

This two statistics are interesting but they have one major drawback, they give a single value for each tag. While this is convenient for ranking them, it loses all the information about their position. That why we finally use the Kulldorf spatial scan statistic[4]. Let $R_{i,w,h}$ be a rectangular region defined by its bottom left cell i, its width w and its height h as the following set of cells:

$$R_{i,w,h} = \{i + k + lg,\ k \in [\![0,\dots,w-1]\!],\ l \in [\![0,\dots,h-1]\!]\}$$

5 The so called background, hence the notation.

For a given R, what we will now call its discrepancy with respect to tag t is

$$d(t,R) = \begin{cases} t_f(R)\log\dfrac{t_f(R)}{b_f(R)} + (1-t_f(R))\log\dfrac{1-t_f(R)}{1-b_f(R)} & \text{if } \dfrac{t_f}{b_f} \geq \dfrac{1-t_f}{1-b_f} \text{ and } t_f \geq T \\ 0 & \text{otherwise} \end{cases}$$

and we recognize the Kullback Leibler divergence restricted to R and its complementary. T is a threshold ensuring that we consider only region with enough support to be significant.

To compute it, we implemented the exhaustive method described in [1, Algorithm 3] with two modifications. First, letting w and h go from 1 to g, there are $g^2 \cdot g \cdot g$ possible regions. But to speed up computations and because we did not want to get regions covering a large portion of the city, we restricted their maximum size. Then, instead of returning the most discrepant region, we maintained a list of the top K ones. Like before, it is informative to look at Table 4 on page 13 to see which tags get high and low values (over all their regions). This time, the values are not normalized between different grid size hence only the relative order is meaningfull. Again tag with high discrepancy are related with locations such as parks. On the other side, tags with low discrepancy are more difficult to interpret because they mostly do not refer to geographic features.

$g = 200$		$g = 80$		$g = 20$	
Lowest entropy					
111minna	.046	theindependent	.008	museumofmodernart	.002
billgraham[]	.052	dnalounge	.012	yerbabuena[]	.003
rodin	.053	franklloydwright	.022	californiapalace[]	.003
teagarden	.058	greatamerican[]	.022	museemecanique	.006
dnalounge	.063	californiapalace[]	.025	cupidsspan	.007
bottomofthehill	.064	bottomofthehill	.025	pier45	.007
cafedunord	.067	saintspeter[]	.026	missiondolorespark	.008
theindependent	.069	rodin	.034	theindependent	.010
franklloydwright	.070	honor	.035	clarionalley	.011
warfield	.074	billgraham[]	.035	asianartmuseum	.012
greatamerican[]	.075	asianartmuseum	.038	fairmont	.012
Highest entropy					
usa	.688	color	.728	sunset	.751
sf	.696	northerncalifornia	.729	dog	.753
instagramapp	.700	square	.735	purple	.755
square	.700	instagramapp	.736	nikon	.756
squareformat	.700	squareformat	.736	sky	.757
iphoneography	.703	iphoneography	.737	blue	.766
unitedstates	.703	iphone	.737	d200	.767
iphone	.720	california	.738	color	.769
foundinsf	.724	sanfrancisco	.744	northerncalifornia	.773
california	.726	gwsf	.766	gwsf	.812
sanfrancisco	.738	foundinsf	.803	foundinsf	.825

Table 2: The tags with lowest and highest entropy for three different grid size (the abbreviated ones are billgrahamcivicauditorium, californiapalaceofthelegionofhonor, yerbabuenacenterforthearts, greatamericanmusichall and saintspeterandpaulchurch).

$g = 200$		$g = 80$		$g = 20$	
Highest divergence					
lakemerced	.587	lakemerced	.568	grandviewpark	.506
grandviewpark	.563	grandviewpark	.554	lakemerced	.485
hunterspoint	.533	hunterspoint	.520	sterngrove	.484
sterngrove	.532	westportal	.514	hunterspoint	.477
westportal	.528	sterngrove	.512	fortfunston	.459
catamaran	.524	chinabeach	.502	chinabeach	.450
dubocepark	.519	dubocepark	.484	westportal	.446
buenavistapark	.518	glenpark	.482	nfl	.441
chinabeach	.518	buenavistapark	.477	candlestickpark	.440
ccsf	.517	candlestickpark	.477	glenpark	.432
glenpark	.508	fortfunston	.471	sfsu	.430
Lowest divergence					
san	.058	san	.032	squareformat	.011
ca	.058	ca	.030	2011	.011
instagramapp	.055	instagramapp	.030	square	.011
squareformat	.055	squareformat	.030	iphoneography	.011
square	.054	unitedstates	.030	francisco	.010
iphoneography	.053	square	.030	san	.010
unitedstates	.053	iphoneography	.029	unitedstates	.009
usa	.046	usa	.026	ca	.008
sf	.042	sf	.021	sf	.006
california	.021	california	.011	california	.004
sanfrancisco	.011	sanfrancisco	.006	sanfrancisco	.002

Table 3: The tags with lowest and highest Kullback Leibler divergence for three different grid size.

$g = 200$		$g = 80$		$g = 20$	
Highest discrepancy					
grandviewpark	7.493	grandviewpark	7.113	grandviewpark	6.521
sterngrove	7.208	sterngrove	6.338	sterngrove	5.998
local	6.881	candlestickpark	6.318	fortfunston	5.924
alcohol	6.666	dubocepark	6.152	candlestickpark	5.638
dubocepark	6.467	pier7	6.072	bayareadisco[]	5.484
yoda	6.467	nfl	6.052	nfl	5.411
coronaheights	6.374	fortfunston	5.948	chinabeach	5.392
performer	6.374	bottomofthehill	5.917	glenpark	5.338
circus	6.371	slims	5.804	sfsu	5.326
candlestickpark	6.285	westportal	5.752	westportal	5.258
bernalheightspark	6.158	bayareadisco[]	5.732	californiapalace[]	5.225
Lowest discrepancy					
instagramapp	0.013	street	0.062	logo	0.054
squareformat	0.013	sign	0.062	friends	0.053
square	0.012	lofi	0.061	sign	0.053
iphoneography	0.012	hair	0.059	yellow	0.052
sf	0.009	hipstamatic	0.059	bayarea	0.051
xproii	0.008	food	0.059	shozu	0.051
sierra	0.007	motionblur	0.057	party	0.050
california	0.004	large	0.056	mirror	0.050
rise	0.003	2009	0.056	hipstamatic	0.050
amaro	0.002	nashville	0.056	purple	0.049
sanfrancisco	0.001	bayarea	0.055	truck	0.048

Table 4: Tags with lowest and highest discrepancy for three different grid size (the abbreviated ones are bayareadiscorymuseum and californiapalaceofthelegionofhonor).

4 EXPERIMENTAL EVALUATION

With this last statistic, we will now see how we can answer the three initial questions, albeit not optimally.

4.1 Tag positioning

To find where a tag appears, we simply compute its discrepancy over all regions that satisfy some size constraints and we keep track of the K ones with the highest discrepancy. When this is done, we merge some of the overlapping regions and discard the others. Namely, we rank region by discrepancy, start from the top one, merge it with its two highest neighbors and remove the others ones. Then we move to next cluster until we have visited all regions once.

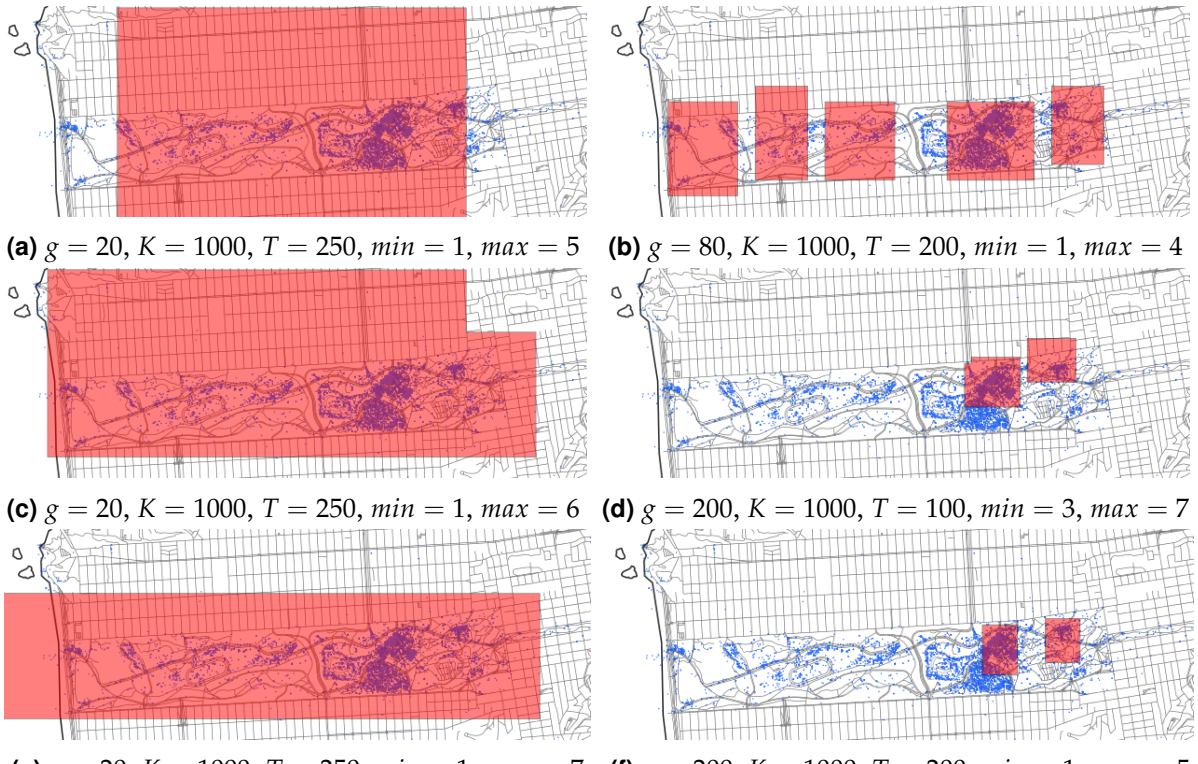
While this is conceptually easy, the main drawback is the sensitivity to the scale of the grid and even for the same grid, the choice of parameters. Figure 3 on the following page show result for museum and Figure 4 on the next page for goldengatepark. Because they cover area of small and larger size, we see that the appropriated grid is different. The problem is that we need to find the correct one before starting the computations.

(a) $g = 200$, $K = 1000$, $T = 130$, $min = 2$, $max = 5$ **(b)** $g = 80$, $K = 1000$, $T = 250$, $min = 1$, $max = 4$

(c) $g = 200$, $K = 1000$, $T = 250$, $min = 2$, $max = 5$ **(d)** $g = 20$, $K = 1000$, $T = 350$, $min = 1$, $max = 1$

(e) $g = 200$, $K = 1000$, $T = 300$, $min = 2$, $max = 2$ **(f)** $g = 20$, $K = 1000$, $T = 350$, $min = 1$, $max = 4$

Figure 3: Photos with tags museum are blue dots and red rectangles represent high discrepancy regions. g is the subdivision of the grid, K the maximum number of regions before merging, T the minimum number of photos in a region to be considered, and min and max are the size constraints of the regions in number of cells.



(a) $g = 20$, $K = 1000$, $T = 250$, $min = 1$, $max = 5$ **(b)** $g = 80$, $K = 1000$, $T = 200$, $min = 1$, $max = 4$

(c) $g = 20$, $K = 1000$, $T = 250$, $min = 1$, $max = 6$ **(d)** $g = 200$, $K = 1000$, $T = 100$, $min = 3$, $max = 7$

(e) $g = 20$, $K = 1000$, $T = 250$, $min = 1$, $max = 7$ **(f)** $g = 200$, $K = 1000$, $T = 200$, $min = 1$, $max = 5$

Figure 4: Same as Figure 3 but for goldengatepark.

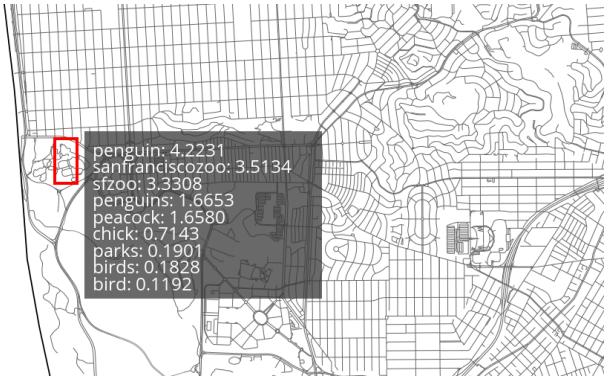
4.2 Location describing

We can also use discrepancy to describe a location as follows: we perform the merging process just presented for all supported tags. For each of them, we will thus obtain a few regions with an associated scalar value. Given a query region, we go through this list and return matching tags, sorted by discrepancy. Screenshots of a working demonstration are displayed in Figure 6 on the following page. One issue they do not show is that the list of results is sometimes very sensitive to the rectangle selected. But they exhibit the another one; this list can be very long and it would be desirable to limit it (for instance in Alcatraz (6c), return only tags with discrepancy above 3). The matching process can also be improved, maybe by scaling the discrepancy by the overlapping area between the query and the tag regions.

4.3 Map covering

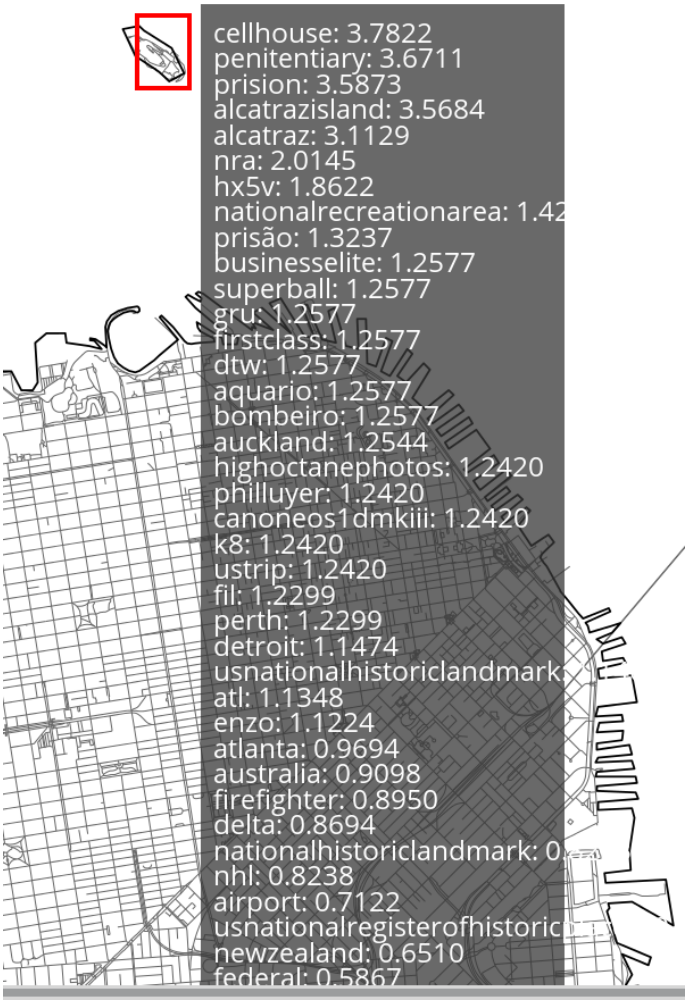
Finally, as a proxy for finding interesting locations, we tried to pave the map with suitable tags. More precisely, we selected the highest discrepancy region of each tag, ranked them and return the top ones. Result is shown in Figure 5. In its current form, the visualization quickly become unreadable if we allow overlapping tags. But there are more fundamental issues. Namely, because it was computed with the values from the 200×200 grid and regions smaller than 4 cells at most, it misses larger areas like the Golden Gate Park. Moreover, discrepancy is an arguable criterion of interest. Even if at first look, it seems to return mostly relevant locations, it lacks flexibility and will always yield the same result, regardless of the user typology and its preferences.

Figure 5: The 20 more discrepant tags are their characteristics location.



(a) San Francisco Zoo, where Google Map indeed reports the existence of a "Penguin Island".

(b) California Palace of the Legion of Honor, a museum with many Auguste Rodin's sculptures.



(c) Alcatraz Island, aka "The Rock", home of an abandoned prison.

Figure 6: Description of three points of interest in San Francisco.

5 CONCLUSION & FUTURE WORK

By **summarizing** what we have done so far, we will be able to see which **limitations** can be addressed in further work. After extracting photos metadata from Flickr and exploring their location and their tags, we computed some statistics about them and notably the discrepancy of the most supported tags. We then use this information to highlight various relationships between tags and locations. But how can these results can be improved?

The first point to look at is **the data**. As mentioned on page 4, they come with some noise and we expect the results to be more relevant if we manage to **clean them**. Yet it remains to be seen if it is worth the extra effort. A simpler direction would be to **increase their amount**. This can be done by crawling photos from others cities, to see if the same methods produce the same results or if working only with San Francisco have introduced some bias. Another approach would be to use more diverse sources like instagram, foursquare or any other social networks that allows users to share localized and tagged contents.

The second limitation that arise from focusing on a single area is that we were not confronted enough with one fundamental issue of spatial analysis, the **modifiable areal unit problem**[3], which states that statistical measurements can be profoundly affected by the choice of the spatial partition and its characteristics size (in our case, the grid and its cells dimension). It is an open question whether the current approach can simply be tuned to tackle different area like states, countries or even the whole world, or if it needs radical change and scale invariant statistics.

A related issue is that the presented approach comes with **many parameters and thresholds**. That would be fine if they encode some prior knowledge but the truth is, they were chosen empirically. Indeed, the problem is mostly unsupervised. For instance, except for specific tag like the name of buildings, most of them can be found in several locations. Likewise, places can be described by many tags, maybe of various relevance but with no clear cut. Finally, finding interesting locations is rather subjective and, depending of the chosen criterion, can yield diverse results. The rather rigid grid division is therefore quite limited, not least because not all locations are rectangular! It may thus be more appropriate to try some **unsupervised clustering methods**, for instance based on the Euclidean norm in the case of the photos locations or drawn from natural language **topic modeling** approaches in the case of tags[2].

Even with this lack of definitive results, it would be helpful to have a way to **evaluate them**. One way could be to generate **artificial data** from which we know precisely what should be found inside, but it would again require some assumptions of our part. We can also create **hand picked ground truth** by manually annotating tags and locations, as in [6]. To cope with ambiguity, it would be even better to aggregate evaluations from several individuals[6]. An easier alternative would be to manually extract famous landmarks from existing tourist guides and assess precision and recall of the method.

6 But also more time consuming, although this may be presented as a game, like GeoGuessr: http://geoguessr.com/.

Finally, whereas photos come with **four kinds of information**tags, location, user and timewe only work with the first two and it would undoubtedly be more interesting to **use all of them**. Let us try to list systematically what we can do of it.

First, by focusing on a single one and building a profile using the rest, we may devise a **similarity measure and use it to perform clustering**. In the case of users, it may divide between wealthy and not, young and old, male or female [5], tourist or inhabitant or richer category like family with kids and traveler of a package tour. As said above, tags can be grouped into common topic[7]. Lastly, although times and locations are subset of $\mathbb{R}$ and $\mathbb{R}^2$ and as such already have a natural metric, they carry additional semantic that we would like to exploit. For instance, we assume that by periodicity, Sundays in 2008 and 2013 share common patterns although they are far apart in time. Likewise, maybe locations in front of the water or that consist of park have similar characteristics.

Then we should consider **pair of concepts**. The present work deals with tags $\leftrightarrow$ locations in both directions, thus there are $\binom{4}{2} - 1$ others to look for: tags $\leftrightarrow$ time, tags $\leftrightarrow$ users, locations $\leftrightarrow$ times, locations $\leftrightarrow$ users and times $\leftrightarrow$ users. Yet all of them may not be interesting and we need to hierarchize our priorities.

REFERENCES

[1] D. Agarwal, A. McGregor, J. M. Phillips, S. Venkatasubramanian, and Z. Zhu. Spatial scan statistics: approximations and performance study. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 24–33. ACM, 2006. URL https://people.cs.umass.edu/~mcgregor/papers/06-kdd.pdf.

[2] D. M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, Apr. 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133826. URL http://www.cs.princeton.edu/~blei/papers/Blei2011.pdf.

[3] C. Gehlke and K. Biehl. Certain effects of grouping upon the size of the correlation coefficient in census tract material. *Journal of the American Statistical Association*, 29(185A): 169–170, Mar. 1934. doi: 10.2307/2277827. URL http://www.jstor.org/stable/2277827.

[4] M. Kulldorff. A spatial scan statistic. *Communications in Statistics-Theory and methods*, 26(6):1481–1496, 1997. URL http://www.satscan.org/papers/k-cstm1997.pdf.

[5] A. Popescu, G. Grefenstette, et al. Mining user home location and gender from flickr tags. In *ICWSM*, 2010.

[6] T. Rattenbury and M. Naaman. Methods for extracting place semantics from Flickr tags. *ACM Transactions on the Web*, 3(1):1–30, Jan. 2009. ISSN 15591131. doi: 10.1145/1462148.1462149. URL http://portal.acm.org/citation.cfm?doid=1462148.1462149.

7 Or at first, we can look at co-occurrences.