Speciale

Fra Hades of Doom

Bjarke Mønsted

Pretentious
quote.

- Famous Person, Born-Died

Fra Hades of Doom

Author	My Name
Advisor	His Name
Co-Advisor	Her Name

Ting, som KU siger der skal stå her

CMOL

Center for Models of Life

Submitted to the University of Great Justice
June 14, 2014

Acknowledgements

Thank you! Thank you all!

Contents

English abstract

WORDS! SOOOOO MANY WORDS!

Dansk sammenfatning

ORD! SAAAAAAAAAA MANGE ORD

Part I

SOCIAL FABRIC PROJECT

Chapter 1

Phone Activity and Quantitative Data Extraction

The main objective of part I of this thesis is to investigate behavioural patterns in the phone activities of the participants in the Social Fabric Project, and to predict various traits of the users based only on their phone logs. Throughout the part, I'll provide brief examples of usage for the software I've written to process the large amount of data available and to apply various prediction schemes to it, while the source code itself is included in the appendix.

My first objective was to investigate how various phone activities correlate with each other temporally, i.e. how a given user's probability for e.g. receiving a call increases or decreases around other activities such as moving around physically. This is the topic of section 1.1.

Next, I set out to replicate some recent research resultsclaiming that people's phone activities predict certain psychological traits. In the most general terms, then, the task consists of predicting a collection of numbers or labels denoted Y based on a set of corresponding data points X. The topic of section 1.2 is the extraction of the many-dimensional data points or *feature vectors* X from the phone logs of the participants, while section 1.3 gives a brief description of the psychological traits Y. Finally, section 1.4 derives an often used linear classification method known as Linear Discriminant Analysis or Fisher's Discriminant and provides a discussion of why it fails for the present dataset, which serves to motivate the more sophisticated prediction schemes introduced in chapter 2.

kilder!!!

1.1 Temporal Correlations in Activity

One category of interesting quantities is the predictability of mobile phone behaviour from recorded behaviour at different times, i.e. the influence of certain events deduced from a user's Bluetooth, GPS or call log data on the tendency of some event to happen in the near past or future. A simple example would be to determine how much placing or receiving a call increases or decreases the probability of a user placing or receiving another call in the period following the first call.

This analysis was performed by comparing an 'activity' signal with a 'background' signal in the following fashion: For each user, the time period around each call is sliced into bins and the each of the remaining calls placed in the bin corresponding to the time between the two calls. Once divided by the total number of calls for the user, this is the activity signal. The background is obtained in a similar fashion but comparing each call in a given user's file with calls in the remaining users' files.

This involves repeatedly binning the time around certain events and then determining in which bin to place other events; a situation in which confusion may arise easily and errors may be hard to identify. To accommodate this, I started out by writing a custom array class designed to greatly simplify the binning procedure. This class called features the following:

- Methods to bin the time around a given event and determine determine which bin a given event falls into. This is useful to implement in the class itself as one then avoids having to continually worry about which bin an event fits into, and as it ensures that bin placement errors can only arise in one small piece of code which can then be tested rigorously.

- Attributes that keep track of the number of events that didn't fit into any bin, and of the current centre of the array, which can then be manipulated to move the array and a method to use this to return a normalized list of the bins.

In short, the binarray can be visualized as a collection of enumerated buckets that can be moved so as to center it on some event and then let other events 'drip' into the buckets. The code for this class is included in A.1.1. In general, objects can be converted to byte streams and stored using Python's pickle module, but as that tends to be both slow and insecure, I generally used json to save my objects. This poses a slight problem as some data types, such as tuples, and custom classes in general are not json

serializable. I got around this by writing some recursive helper methods to help store the relevant information about arbitrary nested combinations of some such objects and to help reconstruct said objects again. These are also included in section A.1.1. As an example of usage, the following code constructs a Binarray, centers it around the present time, and generates a number of timestamps which are then placed in the event. It is then saved to a file using the helper method previously described.

```
from time import time
from random import randint
#Create Binarray with interval +/- one hour and bin size ten minutes.
ba = Binarray(interval = 60*60, bin_size = 10*60)
#Center it on the present
now = int(time())
ba.center = now
#Generate some timestamps around the present
new_times = [now + randint(-60*60, 60*60) for _ in xrange(100)]
for tt in new_times:
    ba.place_event(tt)

#Save it
with open('filename.sig', 'w') as f:
    dump(ba, f)
```

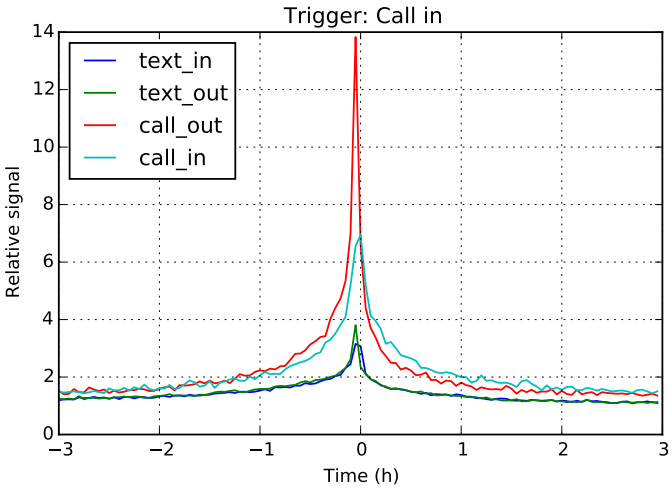
This data will be visualized by plotting the relative signal from the activity of some event, such as in- or outgoing calls or texts, over the background (simply A/B) around another type of event hypothesized to trigger the activity.

1.1.1 Influence of phone calls

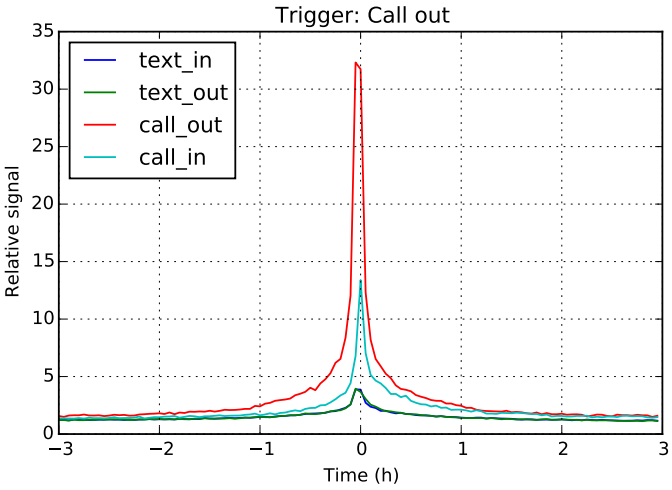
I first investigated the effects of incoming and outgoing calls as triggers for other phone activities. The call logs were stored in a format where each line represents a hashmap with quantities such as call time or call duration mapping to their corresponding value. Below is an example of one such line, were any personal data have been replaced by a random number or hexadecimal string of equal length.

```
{
    "timestamp": 6335212287,
    "number": "c4bdd708b1d7b82e349780ee1e7875caa600c579",
    "user": "ea42a1dbe422f83b0178d158f154f4",
    "duration": 483,
    "type": 2,
    "id": 45687
}
```

the text logs are similar except for the missing duration entry. Computing the relative signal in Binarrays centered on each incoming and outgoing



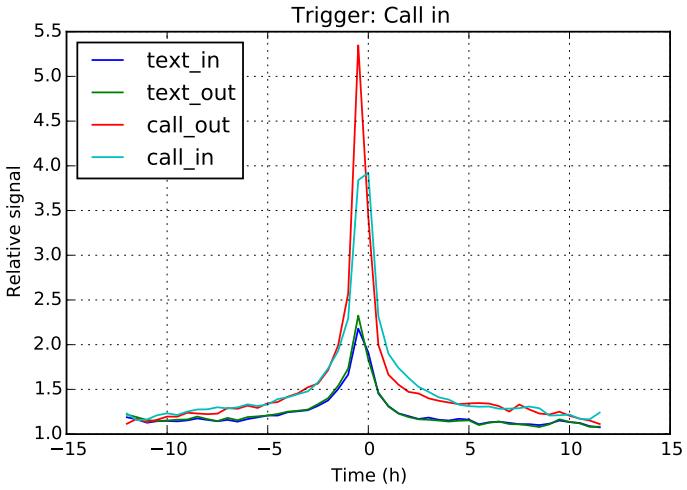
(a) Relative activity of events triggered by incoming calls.



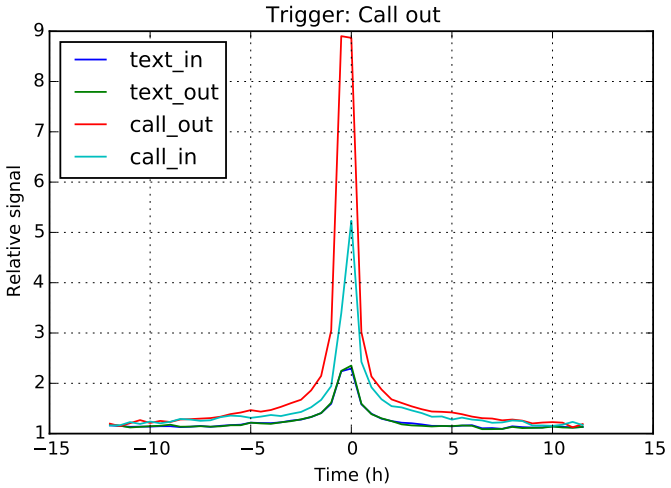
(b) Relative activity of events triggered by outgoing calls.

Figure 1.1: Comparison of the increased activity caused by incoming and outgoing calls over an interval of ± 3 hours around an event with bins of three minutes.

call using bin sizes of three and thirty minutes resulted in the plots shown in figures 1.1 and 1.2, respectively. As the figures clearly show, the all four activities increase significantly for the average user around incoming and outgoing calls.



(a) Relative activity of events triggered by incoming calls.



(b) Relative activity of events triggered by outgoing calls.

Figure 1.2: Comparison of the increased activity caused by incoming and outgoing calls over an interval of ± 12 hours around an event with bins of thirty minutes.

1.1.2 Influence of GPS activity

The raw format of the users' GPS logs looks similar to those of the call and text logs:


```
{
    "timestamp": 8058876274,
    "lon": 6.45051654,
    "user": "0c28e8f4ad9619bca1e5ea4167e10a",
    "provider": "gps",
    "lat": 28.20527041,
    "id": 6429902,
    "accuracy": 39.4
}
```


An analysis similar to that of described in section 1.1.1 was carried out using GPS phone data as triggers. I chose to define a user as being 'active' if they travelled at an average speed of 0.5 $^{m}/_{s}$ between two consecutive GPS log entries, while discarding measurements closely following each other. The reason for this is that the uncertainty on the location measurements could yield false measurements of high average speeds when the measurements are not temporally separated. A lot of the measurements turned out to be grouped somewhat tightly - for instance, approximately 80% of the time intervals were below 100 s. This occurs because the Social Fabric data not only actively records its users' locations with some set interval, but also passively records the location when another app requests it, so when users spend time on apps that need to continually update their position such as Google Maps, a location log entry is written every second. The distribution of intervals between consecutive GPS measurements is shown in figure 1.3. A typical uncertainty on civilian GPS devices is at most 100 m[21], so because I choose to consider a user active if they travel at a mean speed of 0.5 $^{m}/_{s}$, and based on the time spacings shown in figure 1.3, I chose to discard measurements separated by less than 500 s.

An analysis like that of section 1.1.1 reveals that a user's phone activity is significantly increased around times when they are on the move, as shown in figure 1.4. Note the asymmetry of the signal, especially visible in figure 1.4(a). After a measurement of a user being active, the signal dies off about two and a half hours into the future, whereas it persists much longer into the past. Concretely, this means that people's phone activity (tendency to call or text) becomes uncorrelated with their physical activity after roughly two and a half hours, whereas their tendency to move around is increased for much longer time after calling or texting.

The relative signal in figure 1.4(b) appears to be increasing at around ±24 h, which would seem reasonable assuming people have slightly

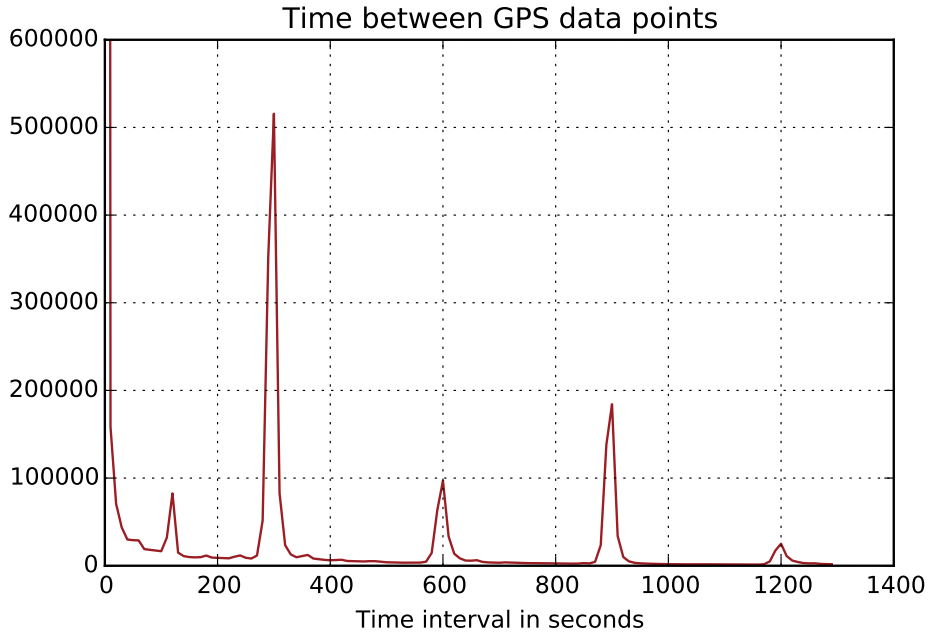
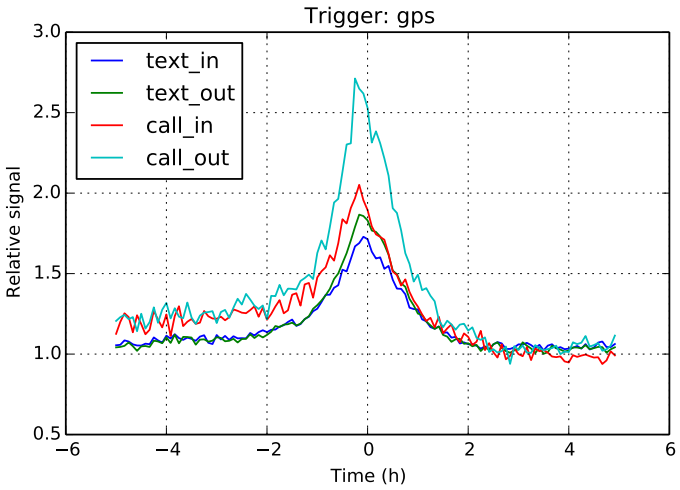
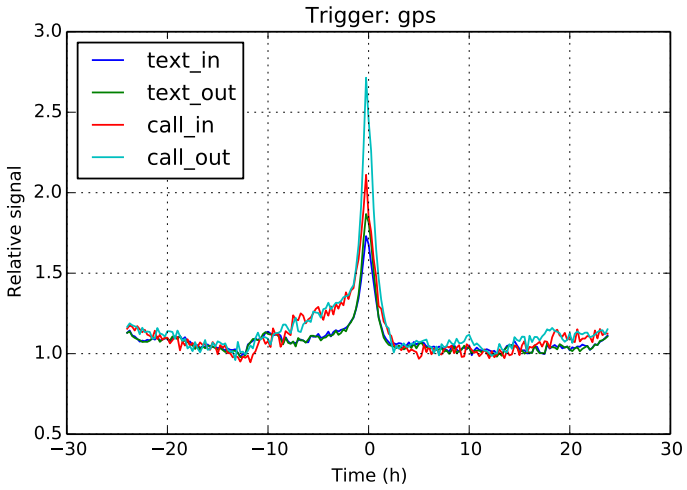


Figure 1.3: Plot of typical temporal spacings between consecutive GPS measurements.

different sleep schedules - if a person is on the move and hence more likely to place a call at time $t = 0$, they're slightly more likely than the general user to be on the move around $t = \pm 24\,\mathrm{h}$. Figure 1.5 shows the same signal extended to $\pm 36\,\mathrm{h}$ where slight bumps are visible 24 hours before and after activity.



(a) Interval: 5 hours. Bin size: 5 minutes.



(b) Interval: 24 hours. Bin size: 15 minutes.

Figure 1.4: Relative increase of activities triggered by GPS activity.

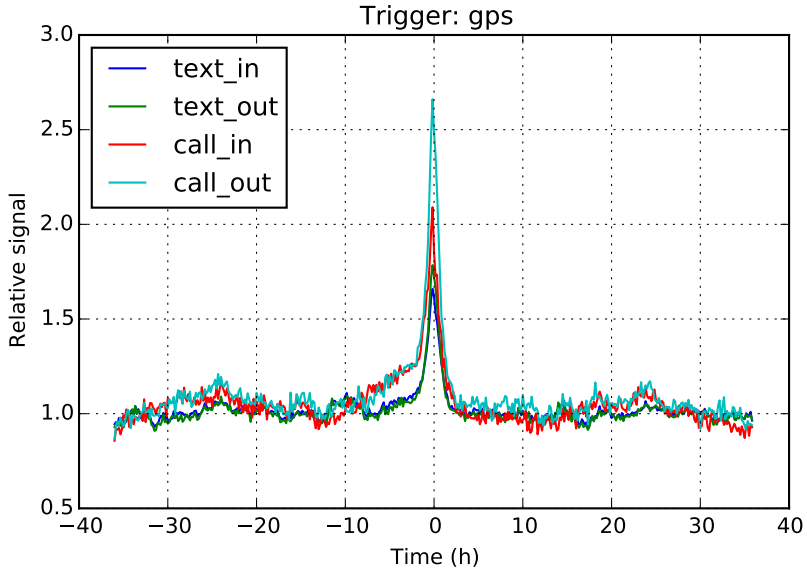


Figure 1.5: GPS-triggered activity increase over an interval of 36 hours using a bin size of 10 minutes.

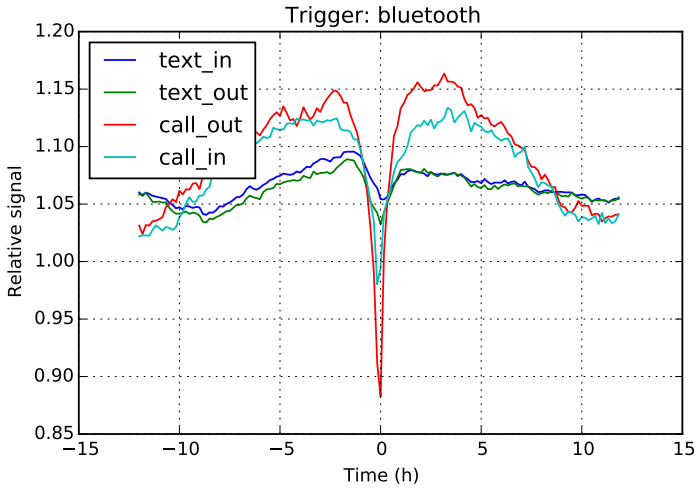
1.1.3 Influence of Bluetooth signal

The following is a randomized entry in a user's bluetooth log.

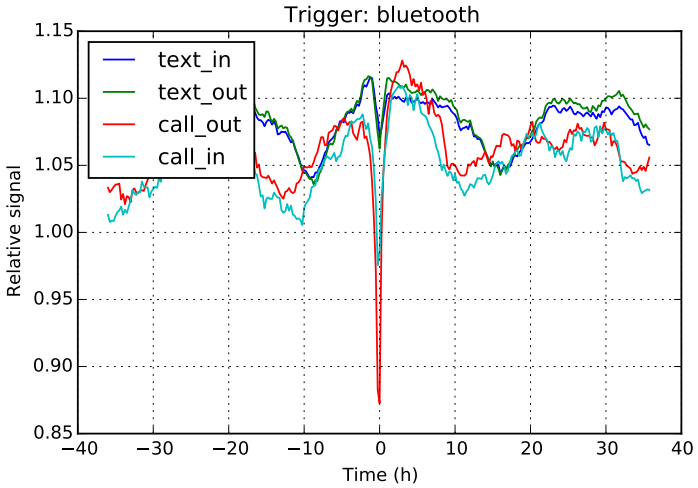

```
{
    "name": "d5306a3672b7a0b8f9696d294ec4b731",
    "timestamp": 6870156680,
    "bt_mac": "1f158ae269d69efa5bb4794ee2a0b2dd68bd3a9badfeaf70f258ad3c74b0c09b",
    "class": 1317046,
    "user": "41cdb7ecaaaec3d33391ed063e7fa2",
    "rssi": −76,
    "id": 4139043
}
```


The 'bt_mac' entry is the MAC-adress of the device which the Bluetooth receiver in the user's phone has registered, so it is reasonable to assume several different MAC addresses occur at several consecutive timestamps. I call the number of repeated MAC adresses needed for a user to be considered social the 'social threshold'. Figures 1.6, 1.7 and 1.8 show the increased activity around times when users were considered social with a threshold of 1, 2 and 4 repeated pings.

Contrary to the previous analyses, phone activities decreased somewhat when users were social. As stated, each of these analyses were fairly similar, I've only explicitly included the code used to extract and save

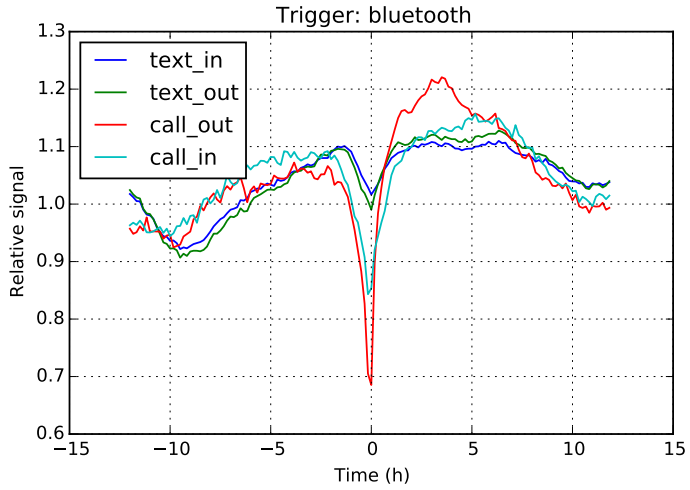


(a) Interval: 12 hours, Bin size: 10 minutes.

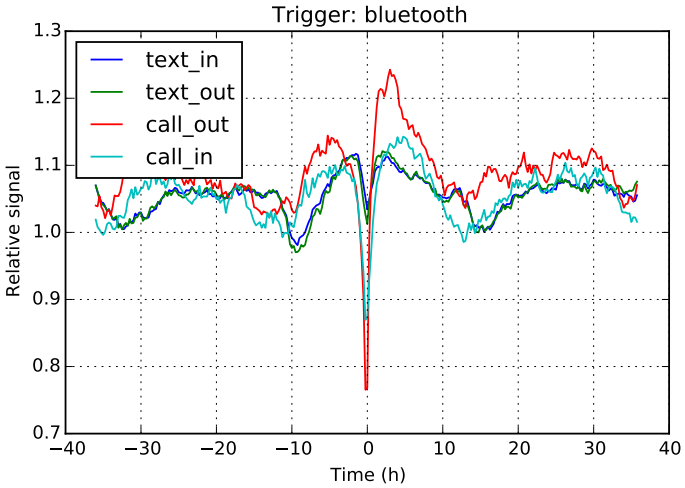


(b) Interval: 36 hours, Bin size: 15 minutes.

Figure 1.6: The effect on phone activity of sociality as measured by the user's Bluetooth signal. The threshold used for being considered social as one repeated signal.

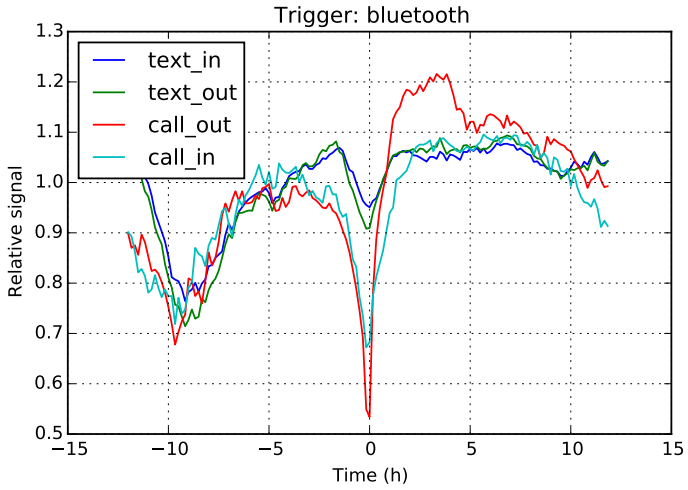


(a) Interval: 12 hours, Bin size: 10 minutes.

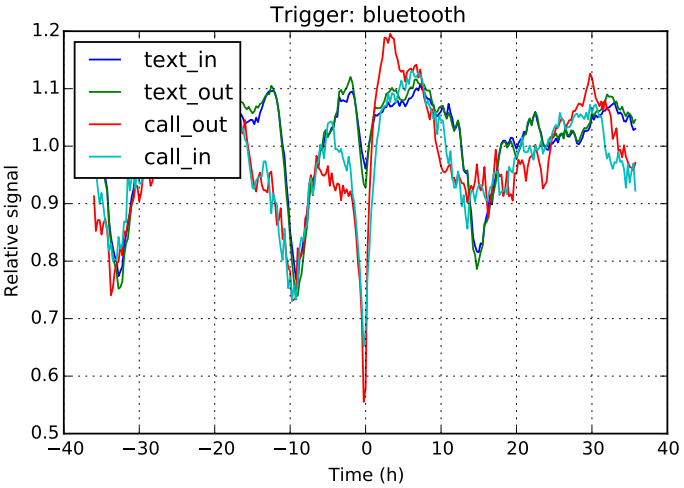


(b) Interval: 36 hours, Bin size: 15 minutes.

Figure 1.7: The effect on phone activity of sociality as measured by the user's Bluetooth signal. The threshold used for being considered social as two repeated signals.



(a) Interval: 12 hours, Bin size: 10 minutes.



(b) Interval: 36 hours, Bin size: 15 minutes.

Figure 1.8: The effect on phone activity of sociality as measured by the user's Bluetooth signal. The threshold used for being considered social as four repeated signals.

Bluetooth data, as well as the code used to load the data and generate figures 1.6 through 1.8. This code is included in section A.1.2.

1.2 Extraction of Input Data

The predictive powers of mobile phone behaviour on the user's psychological profile is currently an area of active research. As part of my thesis work, I have tried to predict the psychological profiles of the SFP participants using various machine learning methods on the available phone logs.

1000 kilder!!!

The software I've written first preprocesses the phone logs to extract various relevant parameters, then collects the parameters and psychological profile scores for each user to serve as input and output, respectively, for the various learning methods. Many of the parameters are chosen following a recent article by de Montjoye et al[6]. The following contains an outline and brief explanation of the extracted parameters.

This section contains a list of the extracted parameters used for psychological profiling along with a brief description of the extraction process when necessary. The preprocessing code is included in section A.1.3.

1.2.1 Simple Call/Text Data

The most straightforward data type is the timestamps from a given user's call/text logs. Six of the parameters used were simply the standard deviation and median of the times between events in the logs for each user's call log, text log, and the combination thereof, excluding time gaps of more than three says on the assumption that it would indicate a user being on vacation or otherwise having a period of telephone inactivity. The entropy S_u of each of the three was also included simply by computing the sum

$$S_u = -\sum_c \frac{n_c}{n_t} \ln_2 \frac{n_c}{n_t}, \tag{1.1}$$

where c denotes a given contact and n_t the total number of interactions, and n_c the number of interactions with the given contact. The number of contacts, i.e. the number of unique phone numbers a given user had contacted by means of calls, texts, and the combination thereof, was also extracted along with the total number of the various kinds of interactions and the contact to interaction ratios. The response rates, defined as the rate of missed calls and incoming texts, respectively, that a given user replied to within an hour, where also determined along with the text

latency defined as the median test response time. Finally the percentage calls and texts that were outgoing was determined as well as the fraction of call interactions that took places during the night, defined as between 22-08.

1.2.2 Location Data

A number of parameters based on the spacial dynamics of the user were also extracted. Among these is the radius of gyration, meaning simply the radius of the smallest enclosing circle enclosing all the registered locations of the user on the given day, and the distance travelled per day. I chose to extract the median and standard deviation of each, filtering out the radii that exceeded 500km so as to keep information about long distance travels in the distance parameter and information about travel within a given region in the radius of gyration parameter.

Cluster Analysis

One parameter which has strong links[6] to psychological traits is the number of locations in which the user typically spends time, and the entropy of their visits to that location. Hence, the task at hand is to identify dense clusters of GPS coordinates for each user. This is a typical example of a task which is very intuitive and quickly approximated by humans, but is extremely computationally expensive to solve exactly. Concretely, the problem of finding the optimal division of n data points into K clusters is formulated as minimizing the 'score' defined as

$$S = \sum_{K} \sum_{x_n \in C_k} |x_n - c_k|^2, \tag{1.2}$$

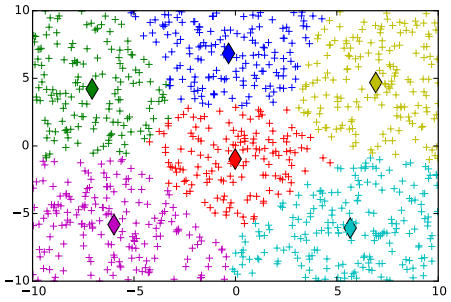
where c_k denotes the centroid of the cluster C_k. Each point x_n is assigned to the cluster corresponding to the nearest centroid. The usual way of approaching this problem is to use Lloyd's algorithm, which consists of initializing the centroids randomly, assigning each point to the cluster corresponding to the centroid which is nearest, then moving each centroid to the center of its points and repeating the last two steps until convergence. As this isn't guaranteed to converge on the global minimum of (1.2), the process can be repeated a number of times, keeping only the result with the lowest value of S. I accomplished this by writing a small Python module to perform various variations of Lloyd's algorithm and to produce plots of the resulting clusters. The code is included in section A.1.5.

This allows one to implement Lloyd's algorithm and visualize its result easily, as the code allows automatic plotting of the result from the

algorithm while automatically selecting different colors for the various clusters. As an example, the following code snippet generates 1000 random points, runs Lloyd's algorithm to determine clusters and saves a plot of the results.

```
points = [[random.uniform(−10,10), random.uniform(−10,10)] for _ in xrange(10**3)]
clusters = lloyds(X = points, K = 6, runs = 1)
draw_clusters(clusters = clusters, filename = 'lloyds_example.pdf')
```

This results in the following visualization:



I chose to modify the algorithm slightly on the following basis: Usually, the algorithm takes as its initial centroids a random sample of the data. I'll call this 'sample' initialization. This leads to a greater number of clusters being initialized in the areas with an increased density of data points, meaning that centroids will be highly cluttered at first, 'fighting' over the dense regions of data points then slowly spreading out. A few such iterations are shown in figure 1.9. However, this method is dangerous: The goal is to identify locations in which a user spends much of their time, i.e. in which more than some threshold of their GPS pings originated, and this initialization is likely to 'cut' the more popular locations into several clusters, neither of which contains more data points than the threshold. One example might be the DTU campus, which is a risk of being divided into several locations with too few data points in each, giving the false impression that user doesn't visit the campus that often. To avoid this effect, I implemented another initialization, 'scatter', in which the clusters start out on points select randomly from the entire range of x, y-values in the user's dataset. This turned out to not only solve the problem described above, but also converge much quicker and reach a slightly lower score as define in (1.2). A few such iterations are shown in figure 1.10. The difference in end results for the two methods is exemplified in figure 1.11. While this works great for users who stay in or around Copenhagen, it will cause problems for people who travel a lot. A user who has visited Australia, for instance, will have their initial clusters spread out across

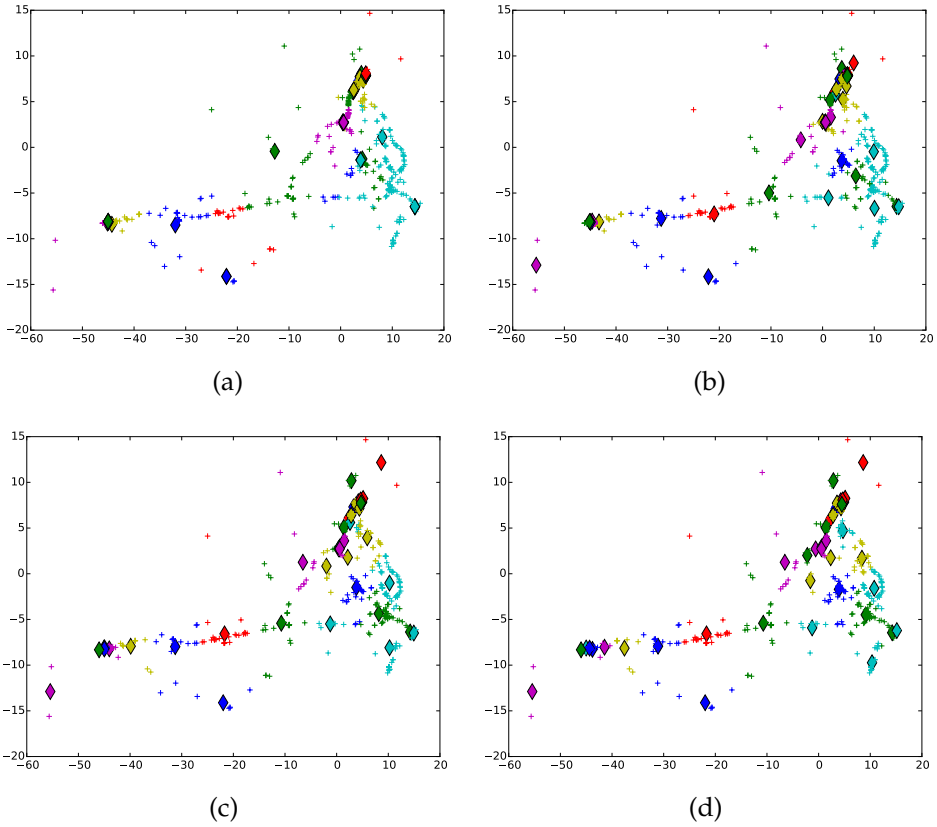


Figure 1.9: A few iterations of Lloyd's algorithm using 'sample' initialization. The axes denote the distance in km to some typical location for the user. Note that clusters are initially cluttered, then slowly creep away from the denser regions.

the globe, and it's highly likely that one them will end up representing all of Denmark. I ended up simply running both versions and keeping the result yielding the highest amount of locations.

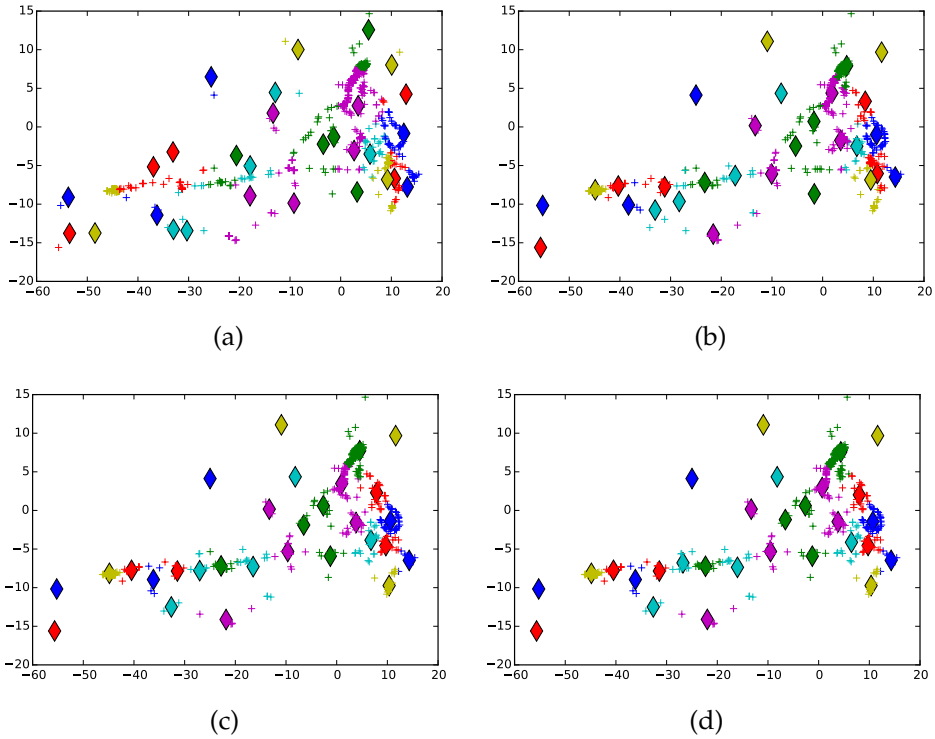
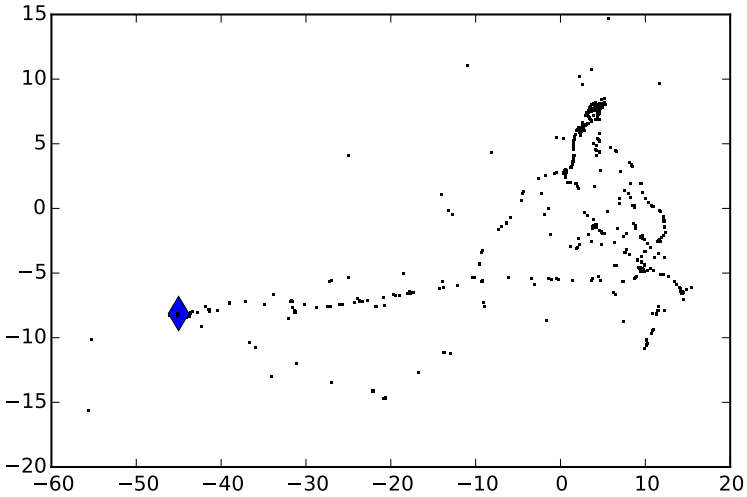
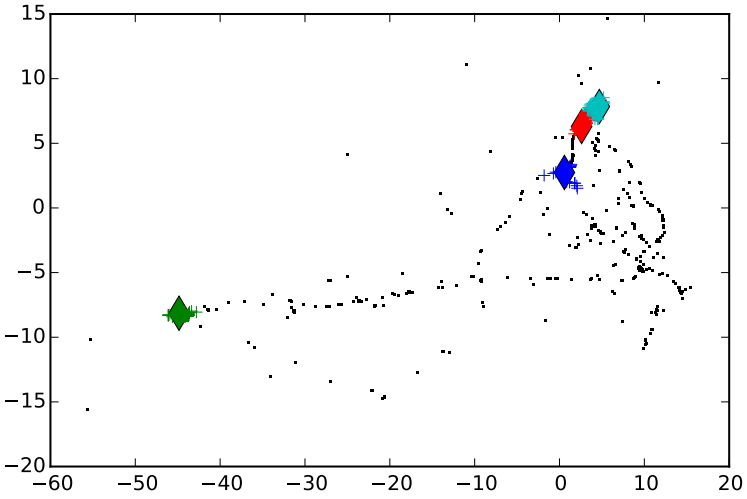


Figure 1.10: A few iterations of Lloyd's algorithm using 'scatter' initialization. The axes denote the distance in km to some typical location for the user. Note that clusters are initially randomly spread across the entire range of x, y-values and converge quickly to a local minimum for (1.2).



(a) Sample initialization.



(b) Scattered initialization.

Figure 1.11: Comparison of the final results of the two initialization methods using 100 initial clusters, a threshold of 5% of the data points before a cluster is considered a popular location and running the algorithm 10 times and keeping the best result. Clusters containing more than 5% of the total amount of data points are in color, whereas the remaining points are black dots.

1.2.3 Time Series Analysis

Another interesting aspect to include is what one somewhat qualitatively might call behavioural regularity - some measure of the degree in which a user's phone activities follow a regular pattern. Quantifying this turns out to take a bit of work. First of all, any user's activity would be expected to closely follow the time of day, so the timestamps of each user's outgoing texts and calls are first converted into 'clock times' meaning simply the time a regular clock in Copenhagen's time zone would display at the given time. This process is fairly painless when using e.g. the UTC time standard, which does not observe daylight saving time (DST), but some subtleties arise in countries that do use DST, as this makes the map from Unix/epoch time to clock time 'almost bijective' - when changing *away* from DST, two consecutive hours of unix time map to the same clock time period (02:00 to 03:00), whereas that same clock period is skipped when changing *to* DST. The most commonly used Python libraries for datetime arithmetic accommodate this by including a dst boolean in their datetime objects when ambiguity might arise, however I simply mapped the timestamps to clock times and ignored the fact twice a year, one time bin will artificially contain contributions from one hour too many or few. One resulting histogram is shown in figure 1.12.

Tilføj lidt om AR-serier når du har bogen!!!

1.2.4 Facebook Data

Unfortunately, the only available Facebook data was a list of each user's friends, so the only contribution of each user's Facebook log was the number of friends the user had.

1.2.5 Bluetooth Data

I extracted a number of different features from each user's Bluetooth log file. First, I set a threshold for when a given user is considered social, as described in section 1.1.3. I chose to use a threshold of two. I then tried to estimate how much time each user spends in the physical company of others in the following way: for each time stamp in the user's Bluetooth log, I checked if the user was social or not and assumed that this status was the same until the following log entry, unless the delay was more than two hours. The rationale behind this is to avoid skewing the measurements if a user turns off their phone for extended periods of time. Otherwise, e.g. studying with a few friends at DTU, turning off your phone and going on vacation for two weeks would give the false impression that the user

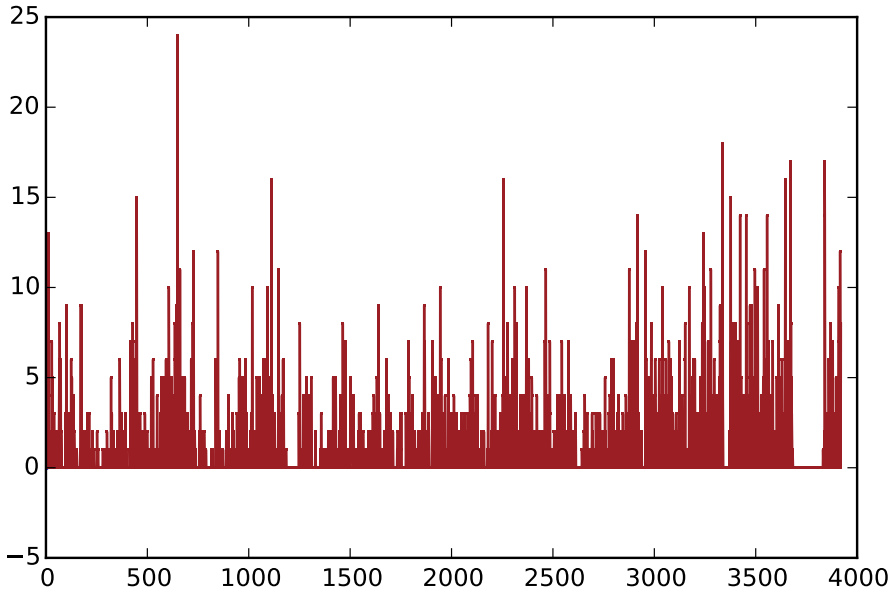


Figure 1.12: Histogram of a user's outgoing calls and texts with a bin size of six hours.

were highly social for a long period of time. I then recorded the fraction of times the user was estimated as being social in this fashion.

Finally, I also wanted some measure of the degrees to which a user's social behaviour follows a pattern. I looked for temporal patterns by fitting AR-series and computing autocorrelation coefficients for each user's social behaviour as described in section 1.2.3. I also chose to compute a 'social entropy' much like (1.1), but weighted by the time the user spends with each acquaintance:

$$E = -\sum_i f_i \ln_2 (f_i), \tag{1.3}$$

$$f_i = \frac{\text{time spent with } i}{\sum_j \text{time spent with } j}. \tag{1.4}$$

Note that the denominator of (1.4) is not equal to the total amount of time spent being social, as the contribution from each log entry is multiplied by the number of people present.

1.3 Output Data

The main emphasis of this part of the thesis is on predicting so-called *Big Five* personality traits. This section contains a brief description of those, following[7]. **Extraversion** signifies how extroverted and sociable a person is. People with high extraversion scores are supposed to be more eager to seek the company of others. **Agreeableness** is supposed to be a measure of how sympathetic or cooperative a person is, whereas **conscientiousness** denotes constraint, self discipline, level of organization etc.. **Neuroticism** signify the tendency to experience mood swings, and is complementary to emotional stability. Finally, **Openness**, also called 'openness to experience', or 'inquiring intellect' in earlier works, signifies thoughtfulness, imagination and so on. These five are collectively referred to as the 'big five' or 'OCEAN' after their initials.

I addition to the above, I also had access to a range self-explanatory traits about the participants such as their gender, whether they smoke etc.

1.4 Linear Discriminant Analysis & Gender Prediction

Linear discriminant analysis is basically a dimensionality reduction technique developed by Fisher in 1936 [8] for separating data points into two or more classes. The general idea is to project a collection of data points in n-dimensional variable space, onto the line or hyperplane which maximizes the separation between classes. Representing data points in n-space by vectors denoted x, the objective is to find a vector ω such that separation between the projected data points on it

$$y = \omega^T x \tag{1.5}$$

is maximized.

To break down the derivation of this method, I will first define a convenient distance measure used to optimize the separation between classes, then solve the resulting optimization problem. For clarity, I'll only describe the case of projection of two classes onto one dimension (i.e. using 'line' rather than 'hyperplane' and so on), although the method generalizes easily.

1.4.1 A measure of separation for projected Gaussians

If the projected data points for two classes a and b follow distributions $\mathcal{N}_a$ and $\mathcal{N}_b$, which are standard Gaussians, $\mathcal{N}_i(x) = \mathcal{N}(x; \mu_i, \sigma_i^2)$, the joint probability distribution for the distance between the projections will be the convolution

$$P(x) = \int_{-\infty}^{\infty} \mathcal{N}_a(y) \cdot \mathcal{N}_b(x - y)\, \mathrm{d}y. \tag{1.6}$$

Computing this for a Gaussian distribution,

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1.7}$$

becomes easier with the convolution theorem, which I'll derive in the following.

Denoting convolution by $*$ and Fourier transforms by

$$\mathcal{F}(f) = \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} f(x) \cdot e^{-i\omega x}\, \mathrm{d}x, \tag{1.8}$$

the convolution theorem is derived as follows:

$$\begin{aligned}
\mathcal{F}(f * g) &= \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} f(y) \cdot g(x - y)\, \mathrm{d}y\, e^{-i\omega x}\, \mathrm{d}\omega, && (1.9)\\
&= \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} f(y) \int_{\mathbb{R}^n} g(x - y) e^{-i\omega x}\, \mathrm{d}y\, \mathrm{d}\omega, && (1.10)\\
&= \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} f(y) \int_{\mathbb{R}^n} g(z) e^{-i\omega(z+y)}\, \mathrm{d}z\, \mathrm{d}\omega, && (1.11)\\
&= \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} f(y) e^{-i\omega y} \int_{\mathbb{R}^n} g(z) e^{-i\omega z}\, \mathrm{d}z\, \mathrm{d}\omega, && (1.12)\\
\boxed{\mathcal{F}(f * g) &= (2\pi)^{n/2} \mathcal{F}(f) \cdot \mathcal{F}(g),} && (1.13)
\end{aligned}$$

where the factor in front of the usual form of the theorem $\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$ stems from the convention of using angular frequency in Fourier transforms, as in (1.8), rather than

$$\mathcal{F}(f) = \int_{\mathbb{R}^n} f(x) \cdot e^{-2\pi i v x}\, \mathrm{d}x. \tag{1.14}$$

Using this, the convolution of two Gaussians can be calculated as

$$\mathcal{N}_a * \mathcal{N}_b = (2\pi)^{n/2} \mathcal{F}^{-1} \left(\mathcal{F}(\mathcal{N}_a) \cdot \mathcal{F}(\mathcal{N}_b) \right). \tag{1.15}$$

The required Fourier transform can be massaged into a nicer form by displacing the coordinate system and cancelling out terms with odd parity:

$$\begin{aligned}
\mathcal{F}(\mathcal{N}(x)) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot e^{-i\omega x} \, \mathrm{d}x, \\
&= \frac{1}{2\pi\sigma} \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} e^{-i\omega(x+\mu)} \, \mathrm{d}x, \\
&= \frac{1}{2\pi\sigma} e^{-i\omega\mu} \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} \left(\cos(\omega x) + i \sin(\omega x)\right) \mathrm{d}x, \\
&= \underbrace{\frac{1}{2\pi\sigma} e^{-i\omega\mu}}_{a} \underbrace{\int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} \cos(\omega x) \, \mathrm{d}x}_{I(\omega)}. \qquad (1.16)
\end{aligned}$$

Noting that $I(\omega)$ reduces to an ordinary Gaussian integral at $\omega = 0$ so $I(0) = \sqrt{2\pi}\sigma$, this can be solved with a cute application of Feynman's trick:

$$\begin{aligned}
\frac{\partial I}{\partial \omega} &= -\int_{-\infty}^{\infty} x e^{-\frac{x^2}{2\sigma^2}} \sin(\omega x) \, \mathrm{d}x, \\
&= \int_{-\infty}^{\infty} \sigma^2 \frac{\partial}{\partial x} \left(e^{-\frac{x^2}{2\sigma^2}}\right) \sin(\omega x) \, \mathrm{d}x, \\
&= \sigma^2 e^{-\frac{x^2}{2\sigma^2}} \sin(\omega x)\Big|_{-\infty}^{\infty} - \omega \int_{-\infty}^{\infty} \sigma^2 e^{-\frac{x^2}{2\sigma^2}} \cos(\omega x) \, \mathrm{d}x, \\
&= -\omega\sigma^2 I(\omega) \Leftrightarrow \\
I(\omega) &= C e^{-\sigma^2\omega^2/2}, \\
I(0) = C &= \sqrt{2\pi}\sigma, \\
I(\omega) &= \sqrt{2\pi}\sigma e^{-\sigma^2\omega^2/2}.
\end{aligned}$$

Plugging this into (1.16) gives the result

$$\mathcal{F}(\mathcal{N}) = \frac{1}{\sqrt{2\pi}} e^{-i\omega\mu} e^{-\sigma^2\omega^2/2}. \qquad (1.17)$$

This can be used in conjunction with (1.13) to obtain

$$\begin{aligned}
\mathcal{F}(\mathcal{N}_a * \mathcal{N}_b) &= \sqrt{2\pi} \frac{1}{\sqrt{2\pi}} e^{-i\omega\mu_a} e^{-\sigma_a^2\omega^2/2} \cdot \frac{1}{\sqrt{2\pi}} e^{-i\omega\mu_b} e^{-\sigma_b^2\omega^2/2}, \qquad (1.18) \\
&= \frac{1}{\sqrt{2\pi}} e^{-i\omega(\mu_a-\mu_b)} e^{-(\sigma_a^2+\sigma_b^2)\omega^2/2}, \qquad (1.19)
\end{aligned}$$

which is recognized as the transform of another Gaussian describing the separation with $\mu_s = \mu_a - \mu_b$ and $\sigma_s^2 = \sigma_a^2 + \sigma_b^2$, so taking the inverse Fourier transformation gives the convolution

$$\mathcal{N}_a * \mathcal{N}_b = \frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{(x-\mu_s)^2}{2\sigma_s^2}}. \tag{1.20}$$

Hence, a reasonable measure of the separation of two projected distributions is

$$d = \frac{(\mu_a - \mu_b)^2}{\sigma_a^2 + \sigma_b^2}. \tag{1.21}$$

1.4.2 Optimizing separation

To maximize the separation, the numerator and denominator, respectively, of (1.21) can be rewritten in terms of w in the following way (using $\widetilde{\mu}_i$ to denote projected means) and simplified by introducing scattering matrices:

$$\begin{aligned}
(\widetilde{\mu}_a - \widetilde{\mu}_b)^2 &= \left(w^T(\mu_a - \mu_b)\right)^2, && (1.22)\\
&= w^T(\mu_a - \mu_b)(\mu_a - \mu_b)^T w, && (1.23)\\
&= w^T S_B w, && (1.24)
\end{aligned}$$

and

$$\begin{aligned}
\widetilde{\sigma}_i^2 &= \sum_{y\in i} \frac{1}{N}(y - \widetilde{\mu}_i)^2, && (1.25)\\
&= w^T \sum_{y\in i} (x - \mu_i)(x - \mu_i)^T w, && (1.26)\\
&= w^T S_i w, && (1.27)\\
\widetilde{\sigma}_a^2 + \widetilde{\sigma}_b^2 &= w^T S_W w, && (1.28)
\end{aligned}$$

having introduced the between-class and within-class scatter matrices S_B and S_W by

$$\begin{aligned}
S_B &= (\mu_a - \mu_b)(\mu_a - \mu_b)^T, && (1.29)\\
S_i &= \sum_{y\in i} (x - \mu_i)(x - \mu_i)^T, && (1.30)\\
S_W &= S_a + S_b. && (1.31)
\end{aligned}$$

Hence, the objective is to solve

$$\frac{\mathrm{d}}{\mathrm{d}w}J(w) = \frac{\mathrm{d}}{\mathrm{d}w}\left(\frac{w^T S_B w}{w^T S_W w}\right) = 0, \tag{1.32}$$

$$\frac{\frac{\mathrm{d}\left[w^T S_B w\right]}{\mathrm{d}w} w^T S_W w - w^T S_B w \frac{\mathrm{d}\left[w^T S_W w\right]}{\mathrm{d}w}}{(w^T S_W w)^2} = 0, \tag{1.33}$$

$$2S_B w \cdot w^T S_W w - w^T S_B w \cdot 2S_W w = 0, \tag{1.34}$$

$$S_B w - \frac{w^T S_B w \cdot S_W w}{w^T S_W w} = 0, \tag{1.35}$$

$$S_B w - S_W w J(w) = 0, \tag{1.36}$$

$$S_B w = S_W w J(w), \tag{1.37}$$

$$S_W^{-1} S_B w = J(w) w. \tag{1.38}$$

The optimal projection vector w^* which satisfies this is

$$w^* = S_W^{-1}\left(\mu_a - \mu_b\right). \tag{1.39}$$

Vær lige sikker på at du forstår det her.

Figure 1.13 shows a visualization of this that I generated by drawing (x, y) points from two random distributions to simulate two distinct classes of points. If the distributions are independent and Gaussian, the projections will also form Gaussian distributions, and the probability of a new point belonging to e.g. class a given its coordinates d can be estimated using Bayesian probability

$$P(a|d) = \frac{P(d|a)P(a)}{P(d|a)p(a) + P(d|b)P(b)}, \tag{1.40}$$

where $P(a)$ and $P(b)$ are simply the prior probabilities for encountering the respective classes, and the conditional probabilities, e.g. $P(d|a)$ are simply given by the value of the projected Gaussian $\mathcal{N}(x'; \widetilde{\mu}_a, \widetilde{\sigma}_a)$ at the projected coordinate x'. In practise, even when the points are not independent or Gaussian, so that (1.40) is not a precise estimate of the probability of the point representing a given class, the class with the highest posteriori according to (1.40) still often turns out to be a good guess.

This method accurately predicted the gender of 79.8% of the participants, which is not particularly impressive as 77.3% of participants were male, so a classifier that assumes that every participant is male would have a comparable success rate. An immediate source of concern is the assumption of linearity: It is possible that the data is ordered in such a way that it is possible to separate data points fairly well based on

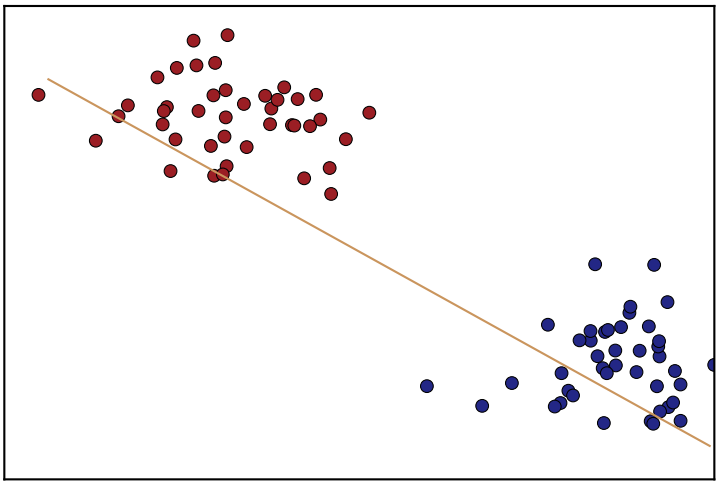


Figure 1.13: Two collections of points drawn from independent Gaussian distributions, representing class a and class b. If the points are projected onto the straight line, which is given by (1.39), the separation between the peaks representing the two classes is maximized.

gender or some psychological trait, just not using a linear classifier. As an extreme example of this, figure 1.14 shows a situation where the points representing one class are grouped together in an 'island' in the middle, isolating them from points representing the remaining class. While it is clear that there's a pattern here, a linear classifier fails to predict classes more precisely than their ratio. Support Vector Machines, or SVMs are another linear classification technique which can be generalized to detect patterns like that in figure 1.14. This is described in section 2.1

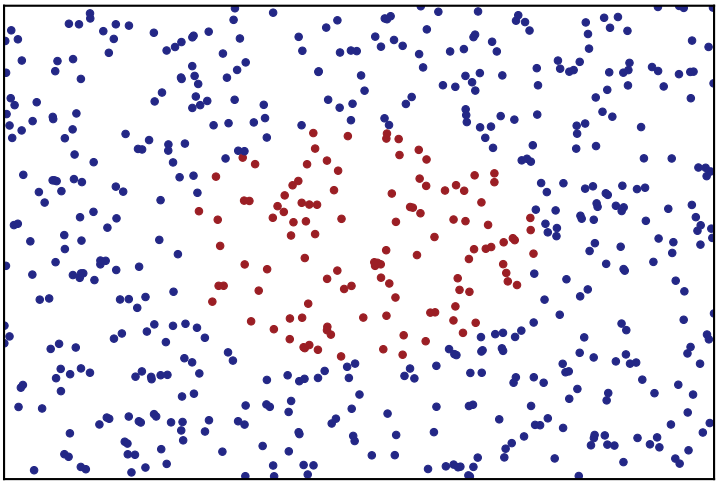


Figure 1.14: An example of data points representing class a are clearly discernible from those of class b, yet a linear Fisher classifier fails to predict the classes more precisely than the ratio of b to a.

CHAPTER 2

Psychological Profiling & Machine Learning

Machine learning is currently a strong candidate for prediction of psychological profiles from phone data. This chapter describes the application of the quantitative data described in setion 1.2 and various machine learning schemes, starting with support vector machines (SVMs).

1000 kilder!!!

Uddyb når der er flere modeller.

2.1 Support Vector Machines

The purpose of this section is to introduce SVMs and attempt to apply them to the data obtained in 1.2. The introduction is mainly based on introductory texts by Marti Hearst [11] and Christopher Burges [5]. SVMs in their simplest form (*simplest* meaning using a linear kernel, which I'll explain shortly) can be thought of as a slight variation on the linear classifier described in section 1.4. However, where LDA finds a line such that the distribution of the points representing various classes projected onto the line is maximized, the aim of SVMs is to establish the hyperplane that represents the best possible slicing of the feature space into regions containing only points corresponding to the different classes. A simple example of this is shown in figure 2.1. Using labels ± 1 to denote classes, the problem may be stated as trying to guess the mapping from an N-dimensional data space to classes $f : \mathbb{R}^N \rightarrow \{\pm 1\}$ based on a set of training data in $\mathbb{R}^N \otimes \{\pm 1\}$. I'll describe separately the properties of

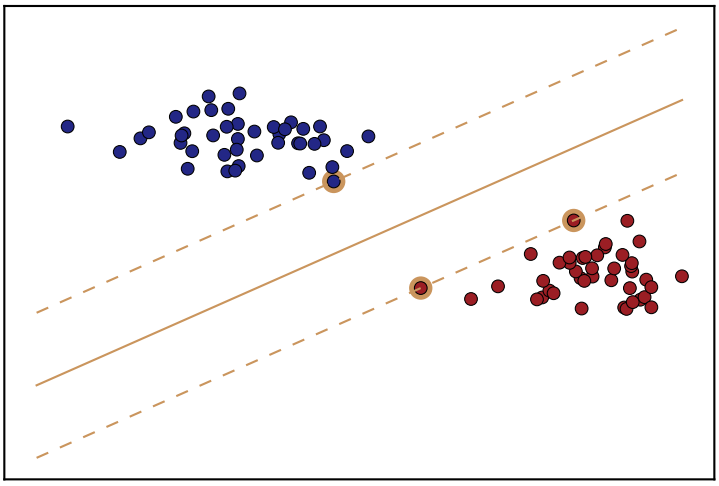


Figure 2.1: The same points as those shown in figure 1.13, except points in class a and class b are now pictured along with their maximally separating hyperplane.

this maximally separating hyperplane, how it is obtained, and how the method is generalized to non-linear classification problems as the 'island' illustrated in figure 1.14.

The well-known equation for a plane is obtained by requiring that its normal vector $\mathbf{w}$ be orthogonal to the vector from some point in the plane $\mathbf{p}$ to any point $\mathbf{x}$ contained in it:

$$\mathbf{w} \cdot (\mathbf{x} - \mathbf{p}) = 0. \tag{2.1}$$

The left hand side of (2.1) gives zero for points in the plane and positive or negative values when the point is displaced in the same or opposite direction as the normal vector, respectively. Hence, sign $(\mathbf{w} \cdot (\mathbf{x} - \mathbf{p}))$ may be taken as the decision function. It is clear from (2.1) that the normal vector may be scaled without changing the actual plane (of course the decision function is inverted if a negative value is chosen), so $\mathbf{w}$ is usually rescaled such that

$$\mathbf{w} \cdot (\mathbf{x} - \mathbf{p}) = \mathbf{w} \cdot \mathbf{x} + b = \pm 1, \tag{2.2}$$

for the points that are closest to the separating plane. Those points located on the margin are encircled in figure 2.1. In general then, the meaning of

the sign and magnitude of

$$\mathbf{w} \cdot \mathbf{x} + b \tag{2.3}$$

will be the predicted class and a measure of prediction confidence, respectively, for new data points. Finally, note that $\mathbf{w}$ can be expanded in terms of the data points that are on the margin in figure 2.1 as

$$\mathbf{w} = \sum_i v_i \mathbf{x}_i, \tag{2.4}$$

these $\mathbf{x}_i$, the position vectors of the margin points in data space, are the 'support vectors' that lend their name to the method.

2.1.1 Obtaining the Maximally Separating Hyperplane

Assuming first that it is possible to slice the data space into two regions that contain only points corresponding to one class each, and that the plane's normal vector has already been rescaled according to (2.2), the following inequalities hold:

$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b \geq 1, y_i = +1, \\ \mathbf{x}_i \cdot \mathbf{w} + b \leq -1, y_i = -1. \end{aligned} \tag{2.5}$$

Multiplying by y_i, both simply become

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \geq 0. \tag{2.6}$$

The distance between the separating plane and each of the margins in figure 2.1 is $1/|\mathbf{w}|$, so in order to maximize the separation, $|\mathbf{w}|$ must be minimized. For mathematical convenience, $\frac{1}{2}|\mathbf{w}|^2$, rather than $|\mathbf{w}|$ is included in the Lagrangian, which then becomes

$$L = \frac{1}{2}|\mathbf{w}|^2 - \sum_i \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b - 1 \right), \tag{2.7}$$

must be minimized with the constraints

$$\alpha_i \geq 0, \tag{2.8}$$

$$\frac{\partial L}{\partial \alpha_i} = 0. \tag{2.9}$$

A result from convex optimization theory known as Wolfe Duality[20] states that one may instead maximize the above Lagrangian subject to

$$\nabla_w L = \frac{\partial L}{\partial b} = 0, \tag{2.10}$$

which gives conditions

$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j, \tag{2.11}$$

$$\sum_j \alpha_j y_j = 0. \tag{2.12}$$

These can be plugged back into (2.7) to obtain

$$L_D = \frac{1}{2}\sum_i \sum_j \alpha_i y_i \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_i \alpha_i y_i \Big(\mathbf{x}_i \cdot \sum_j \alpha_j y_j \mathbf{x}_j + b\Big) + \sum_i \alpha_i, \tag{2.13}$$

$$L_D = -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j + \sum_i \alpha_i. \tag{2.14}$$

A problem with this is that eqs 2.5 can only be satisfied in the completely separable case, although it is easy to imagine an example in which a classifier performs well but not flawlessly on the training set. For instance, if two points, one from each class, in figure 2.1 were permuted, the classifier shown in the plot would still do a very good job, but eqs. 2.5 would not be satisfiable, causing the method to fail. This is remedied by introducing slack variables[3]

$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b &\geq 1 - \xi_i, \quad y_i = +1, \\ \mathbf{x}_i \cdot \mathbf{w} + b &\leq -(1 - \xi_i), \quad y_i = -1, \\ \xi_i &\geq 0, \end{aligned} \tag{2.15}$$

which allows the algorithm to misclassify. This should come without a cost to the overall Lagrangian, or one would just end up classifying randomly, so a 'cost term', $C \cdot \sum_i \xi_i$ is added as well. The value of C (as well as a few similar parameters which have yet to be introduced) is usually determined experimentally by simply varying it across some space of possible values and choosing the value resulting in the best performance - see for instance figure 2.3. In the case of the misclassification cost parameter C, low values will result in low performance, whereas too large values will result in overfitting. The above can be rewritten exactly as previously, except another set of non-negative Lagrange multipliers μ_i are added to (2.7) to ensure positivity of the ξ_i, resulting in

$$L = \frac{1}{2}|\mathbf{w}|^2 + C \cdot \sum_i \xi_i - \sum_i \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b - 1 + \xi_i\right) - \sum_i \mu_i \xi_i. \tag{2.16}$$

This results in the same dual Lagrangian L_D as before, but with an upper bound on the α_i:

$$0 \leq \alpha_i \leq C. \tag{2.17}$$

The methods outlined above can also be used to solve regression, rather than classification, problems[19]. The training data will then be in $\mathbb{R}^{N+1}$ rather than in $\mathbb{R}^N \otimes \{\pm 1\}$, and the value of the decision function in (2.3) is predicted instead of using only its sign. The criterion of correct classification from (2.6) is replaced by the demand that predictions be within some tolerated margin ϵ of the true value of the training point y_i, so (2.5) becomes

$$-\epsilon \leq \mathbf{x}_i \cdot \mathbf{w} + b - y_i \leq \epsilon \tag{2.18}$$

so when slack variables ξ_i and ξ_i^* (for the lower and upper bound, respectively) like in (2.15) are introduced, the Lagrangian from (2.16) becomes

$$L = \frac{1}{2}|\mathbf{w}|^2 + C\sum_i \left(\xi_i + \xi_i^*\right), \tag{2.19}$$

with constraints

$$\mathbf{x}_i \cdot \mathbf{w} + b - y_i \geq -\left(\epsilon + \xi_i\right), \tag{2.20}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b - y_i \leq \epsilon + \xi_i^*, \tag{2.21}$$

$$\xi_i, \xi^* \geq 0. \tag{2.22}$$

The main point to be emphasized here is that the training data $\mathbf{x}_i$ only enter into the dual Lagrangian of (2.14) as inner products. This is essential when extending the SVM model to nonlinear cases, which is the subject of the following section.

2.1.2 Generalizing to the non-linear case

The fact that the data $\mathbf{x}_i$ only occur as inner products in (2.14) makes one way of generalizing to non-linearly separable datasets straightforward: Referring back to figure 1.14, one might imagine bending the plane containing the data points by curling the edges outwards in a third dimension after which a two-dimensional plane could separate the points very well. In general, this means applying some mapping

$$\Phi : \mathbb{R}^l \to \mathbb{R}^h, \quad h > l, \tag{2.23}$$

to the $\mathbf{x}_i$ (l and h are for low and high, respectively). For example, one could look for a mapping such that the new inner product becomes

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = \left(\mathbf{x}_i \cdot \mathbf{x}_j\right)^2. \tag{2.24}$$

I'll describe the components of each vector separately, so I'm going to change notation to let the subscripts denote coordinates and using $\mathbf{x}$ and $\mathbf{y}$ as two arbitrary feature vectors, where the latter shouldn't be confused with the class labels used earlier. As an example, in two dimensions the above becomes

$$(\mathbf{x}\cdot\mathbf{y})^2 = \left(\sum_{i=1}^{2} x_i y_i\right)^2 = x_1^2 y_1^2 + 2x_1 y_1 x_2 y_2 + x_2^2 y_2^2, \tag{2.25}$$

meaning that one possibility for Φ is

$$\Phi : \mathbf{x} \mapsto \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix} \tag{2.26}$$

This can be generalized to d-dimensional feature vectors and to taking the n'th power rather than the square using the multinomial theorem:

$$\left(\sum_{i=1}^{d} x_i\right)^n = \sum_{\sum_{i=1}^{d} k_i = n} \frac{n!}{\prod_{l=1}^{d} k_l!} \prod_{j=1}^{d} x_j^{k_j}, \tag{2.27}$$

where the subscript $\sum_{i=1}^{d} k_i = n$ simply means that the sum goes over any combination of d non-negative integers k_i that sum to n. I wish to rewrite this slightly for two reasons: to simplify the notation in order to make a later proof more manageable, and to help quantify how quickly the number of dimensions in the output space grows to motivate a trick to avoid these explicit mappings.

As stated, the sum on the RHS of (2.27) runs over all combinations of d integers which sum to n. This can be simplified by introducing a function K, which simply maps

$$K : n, d \mapsto \left\{ \{k\} \in \mathbb{N}^d \,\middle|\, \sum_{i=1}^{d} k_i = n \right\}, \tag{2.28}$$

and denoting each of those collections $\{k\}_i$ so each of the coefficients in (2.27) can be written

$$\frac{n!}{\prod_{i=1}^{d} k_i!} = C_{\{k\}}. \tag{2.29}$$

Then, (2.27) becomes

$$\left(\sum_{i=1}^{d} x_i\right)^n = \sum_{K(n,d)} C_{\{k\}} \prod_{j=1}^{d} x_j^{k_j} \tag{2.30}$$

To show how quickly the dimensions of the required embedding space grows, note that the dimension is equal to the number of terms in the sum above, i.e.

$$\dim(\mathbb{R}^h) = |K(n,d)| = \left| \left\{ \{k\} \in \mathbb{N}^d \left| \sum_{i=1}^{d} k_i = n \right. \right\} \right|, \tag{2.31}$$

which can be computed using a nice trick known from enumerative combinatorics.

Consider the case where $n = 5$ and $d = 3$. $K(5,3)$ then contains all sets of 3 integers summing to 5, such as $1,3,1$ or $0,1,4$. Each of these can be uniquely visualized as 5 unit values distributed into 3 partitions in the following fashion:

$$\circ \mid \circ \circ \circ \mid \circ,$$
$$\mid \circ \mid \circ \circ \circ \circ,$$

and so on. It should be clear that you need n $\circ$-symbols and $d-1$ $\mid$ separators. The number of possible such combinations, and hence the dimensionality of the embedding space, is then

$$\binom{n+d-1}{n} = \frac{(n+d-1)!}{n!(d-1)!}. \tag{2.32}$$

This number quickly grows to be computationally infeasible, which motivates one to look for a way to compute the inner product in the embedded space without performing the explicit mapping itself. This is the point of the so-called 'kernel trick', which I'll introduce in the following.

The idea of the kernel trick is that since only the inner products between feature vectors in the embedded space are required, one might as well look for some function K of the original feature vectors which gives the same scalar as the inner product in the embedded space, i.e.

$$K(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}). \tag{2.33}$$

In the polynomial case treated above, the correspondence between the kernel function $K(\mathbf{x},\mathbf{y})$ and the explicit mapping Φ is straightforward:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^n, \tag{2.34}$$

$$\Phi(\mathbf{x}) = \sum_{K(n,d)} \sqrt{C_{\{k\}}} \prod_{j=1}^{d} x_j^{k_j}, \tag{2.35}$$

so that (2.33) is true by the multinomial theorem and the above considerations. However, situations arise in which the explicit mapping Φ isn't directly obtainable, and the correspondence of the kernel function to inner products in higher dimensional spaces is harder to demonstrate. This is the subject of the following section.

Radial Basis Functions

One commonly used kernel function is the RBF, or radial basis function, kernel:

$$K(\mathbf{x}, \mathbf{y}) = e^{|\mathbf{x}-\mathbf{y}|^2/2\sigma}. \tag{2.36}$$

Burges [5] shows that the polynomial kernel is valid, so I'll show how the argument extends to the RBF kernel in the following.

Mercer's condition[18] states that for a kernel function $K(\mathbf{x}, \mathbf{y})$, there exists a corresponding Hilbert space $\mathcal{H}$ and a mapping Φ as specified earlier, iff any L^2-normalizable function $g(\mathbf{x})$ satisfies

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) \, d\mathbf{x} \, d\mathbf{y} \geq 0. \tag{2.37}$$

This can be shown be rewriting (2.36) as

$$K(\mathbf{x}, \mathbf{y}) = e^{(\mathbf{x}-\mathbf{y})\cdot(\mathbf{x}-\mathbf{y})/2\sigma} = e^{|\mathbf{x}|^2/2\sigma} e^{|\mathbf{y}|^2/2\sigma} e^{-\mathbf{x}\cdot\mathbf{y}/\sigma}, \tag{2.38}$$

and expanding the last term in $(\mathbf{x} \cdot \mathbf{y})$ as

$$e^{-\mathbf{x}\cdot\mathbf{y}/\sigma} = \sum_{i=0}^{\infty} \frac{(-1)^i}{i!\sigma^i} (\mathbf{x} \cdot \mathbf{y})^i, \tag{2.39}$$

but using (2.30) on the dot product gives

$$(\mathbf{x} \cdot \mathbf{y})^i = \left(\sum_{j=1}^{d} x_j y_j \right)^i = \sum_{K(i,d)} C_{\{k\}} \prod_{j=1}^{d} x_j^{k_j} y_j^{k_j} \tag{2.40}$$

so the Taylor expansion becomes

$$e^{-\mathbf{x}\cdot\mathbf{y}/\sigma} = \sum_{i=0}^{\infty} \sum_{K(i,d)} \frac{(-1)^i}{i!\sigma^i} C_{\{k\}} \prod_{j=1}^{d} x_j^{k_j} y_j^{k_j}, \tag{2.41}$$

which can be plugged back into (2.38) to yield

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{\infty} \sum_{K(i,d)} \frac{(-1)^i}{i!\sigma^i} C_{\{k\}} e^{|\mathbf{x}|^2/2\sigma} e^{|\mathbf{y}|^2/2\sigma} \prod_{j=1}^{d} x_j^{k_j} y_j^{k_j}. \tag{2.42}$$

The underlying reason for these algebraic shenanigans is that (2.42) is clearly separable so that the integral in (2.37) from Mercer's condition becomes

$$\int K(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \tag{2.43}$$

$$=\sum_{i=0}^{\infty}\sum_{K(i,d)}\frac{(-1)^i}{i!\sigma^i}C_{\{k\}}\int_{\mathbb{R}^{2d}}\mathrm{e}^{|\mathbf{x}|^2/2\sigma}\mathrm{e}^{|\mathbf{y}|^2/2\sigma}\prod_{j=1}^{d}x_j^{k_j}y_j^{k_j}g(\mathbf{x})g(\mathbf{y})\,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \tag{2.44}$$

$$=\sum_{i=0}^{\infty}\sum_{K(i,d)}\frac{(-1)^i}{i!\sigma^i}C_{\{k\}}\left(\int_{\mathbb{R}^d}\mathrm{e}^{|\mathbf{x}|^2/2\sigma}\prod_{j=1}^{d}x_j^{k_j}g(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)\cdot\left(\int_{\mathbb{R}^d}\mathrm{e}^{|\mathbf{y}|^2/2\sigma}\prod_{j=1}^{d}y_j^{k_j}g(\mathbf{y})\,\mathrm{d}\mathbf{y}\right) \tag{2.45}$$

$$=\sum_{i=0}^{\infty}\sum_{K(i,d)}\frac{(-1)^i}{i!\sigma^i}C_{\{k\}}\left(\int_{\mathbb{R}^d}\mathrm{e}^{|\mathbf{x}|^2/2\sigma}\prod_{j=1}^{d}x_j^{k_j}g(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)^2 \tag{2.46}$$

$$\geq 0. \tag{2.47}$$

Hence, radial basis functions satisfy Mercer's condition and the kernel described above can be plugged into the dual Lagrangian from (2.14) to obtain

$$L_D=-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathrm{e}^{|\mathbf{x}_i-\mathbf{x}_j|^2/2\sigma}+\sum_i\alpha_i, \tag{2.48}$$

which must be maximized subject to the same constraints as earlier. The concrete optimization procedure is complicated and already implemented in most machine learning libraries, so I choose not to go into details with that, but instead to demonstrate the effectiveness of the RBF kernel approach on the non-linear-separable points that were generated earlier. figure 2.2 shows the points again, along with the *decision frontier* i.e. the curve which separates regions in which points are classified into separate classes. The danger of overfitting should be clear from figure 2.2. If the cost of misclassification C and the sharpness of the RBFs, usually denoted by $\gamma = 2/\sigma$ are set sufficiently high, the algorithm will simply end up with a tiny decision boundary around every training point of class a, resulting in flawless classification on the training set, but utter failure on new data. The typical way of evaluating this is to perform k-fold validation, meaning that the available data is k equal parts and the SVM is consecutively trained on $k-1$ parts and tested on the remaining. A variant of this, which my code uses, is stratified k-fold validation, which only differs in that the data is partitioned so as to keep the ratio between the different classes in each parts as close to equal as possible.

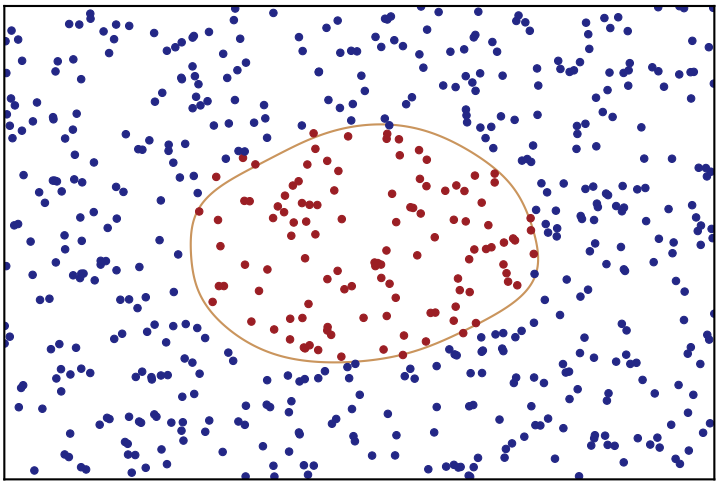


Figure 2.2: The 'island' scenario of figure 1.14 revisited. The points representing class a and class b have been mapped to a higher-dimensional space in which it is possible to construct a separating hyperplane whose decision frontier is also shown.

The γ parameter is often fixed by performing a grid search similar to that discussed earlier. Figure 2.3 shows the resulting heat map from a grid search.

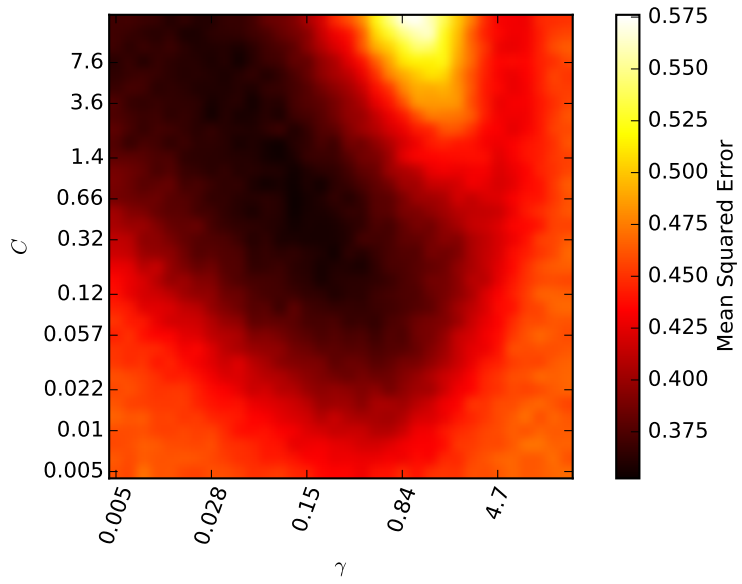


Figure 2.3: Result of a grid search for the optimal combination of values for the cost parameter C and the sharpness γ of the Gaussian kernel function giving optimal values of $C = 0.52$ and $\gamma = 0.12$.

2.1.3 Implementing a custom weighted radial basis function kernel

In general, the features extracted as described in section 1.2 are too numerous for efficient implementations of most machine learning schemes. [6] work around this by determining the linear correlation between each feature and the target values, and then discarding features whose correlation is below some threshold and letting the rest contribute equally in the SVM. They never explicate this threshold but state that they include features 'significantly related' to the target values, which convention would suggest means a threshold of 0.05. [15] have demonstrated significant improvements in SVM performance by assigning variable importances to each feature, so I implemented a modified kernel function where I don't just discard features with linear correlations below some threshold but also assign to each remaining feature a normalized weight given by its correlation with the output variable.

I do this by defining a diagonal matrix $\hat{M}$ where the i, ith element is the linear correlation coefficient between feature i and the target variable, and using the matrix as a metric in the exponent of the usual radial basis

functions so the kernel $K(\mathbf{x}, \mathbf{y})$ becomes

$$e^{\gamma|\mathbf{x}-\mathbf{y}|^2} \rightarrow e^{\gamma(\mathbf{x}-\mathbf{y})^T \hat{M}(\mathbf{x}-\mathbf{y})} \tag{2.49}$$

Note that this does not change the validity of the proof I gave in section 2.1.2, so this weighted RBF kernel also satisfies Mercer's condition and may hence be used as a kernel function. It turned out to be nontrivial to write an implementation of this which had a syntax consistent with that of the default methods available in the library I used, and which provided a similarly simple way to grid search over its parameters (C and γ as in the usual RBF kernel SVM along with a threshold parameter). I ended up solving this by writing a metamethod to provide a kernel matrix based on an input list of variable importances as well as a default sharpness parameter denoted γ.

```
def make_kernel(importances, gamma = 1.0):
    '''Returns a weighted radial basis function (WRBF) kernel which can be
     passed to an SVM or SVR from the sklearn module.

     Parameters:
     -----------------------
     importances : list
       The importance of each input feature. The value of element i can mean
       e.g. the linear correlation between feature i and target variable y.
       None means feature will be weighted equally.

     gamma : float
       The usual gamma parameter denoting inverse width of the gaussian used.
     '''
    def kernel(x,y, *args, **kwargs):
        d = len(importances) #number of features
        impsum = sum([imp**2 for imp in importances])
        if not impsum == 0:
            normfactor = 1.0/np.sqrt(impsum)
        else:
            normfactor = 0.0
        #Metric to compute distance between points
        metric = dok_matrix((d,d), dtype = np.float64)
        for i in xrange(d):
            metric[i,i] = importances[i]*normfactor
        #
        result = np.zeros(shape = (len(x), len(y)))
        for i in xrange(len(x)):
            for j in xrange(len(y)):
                diff = x[i] - y[j]
                dist = diff.T.dot(metric*diff)
                result[i,j] = np.exp(-gamma*dist)
        return result
    return kernel
```

With that in place, it was a simple matter to implement classifiers and regressors by inheriting from the default classes and overriding the constructor methods like this:

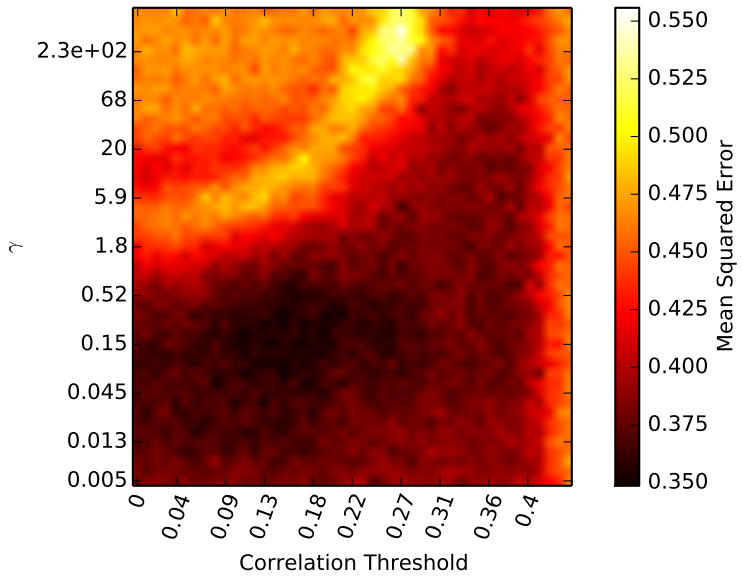


Figure 2.4: Heat map resulting from a grid search over the parameter space of the linear correlation threshold and the default sharpness γ of the radial basis functions showing optimal values of 0.25 and 0.15 for γ and the threshold, respectively.

```
class WRBFR(svm.SVR):
    '''Weighted radial basis function support vector regressor.'''
    def __init__(self, importances, C = 1.0, epsilon = 0.1,
                 gamma = 0.0):
    kernel = make_kernel(importances = importances, gamma = gamma)
    super(WRBFR, self).__init__(C = C, epsilon = epsilon, kernel = kernel)
```

Figures 2.4 and 2.5 show heat maps resulting from grid searches over the threshold parameter versus γ and C, respectively. Interestingly, the ideal correlation threshold seems to be well above the value used in [6].

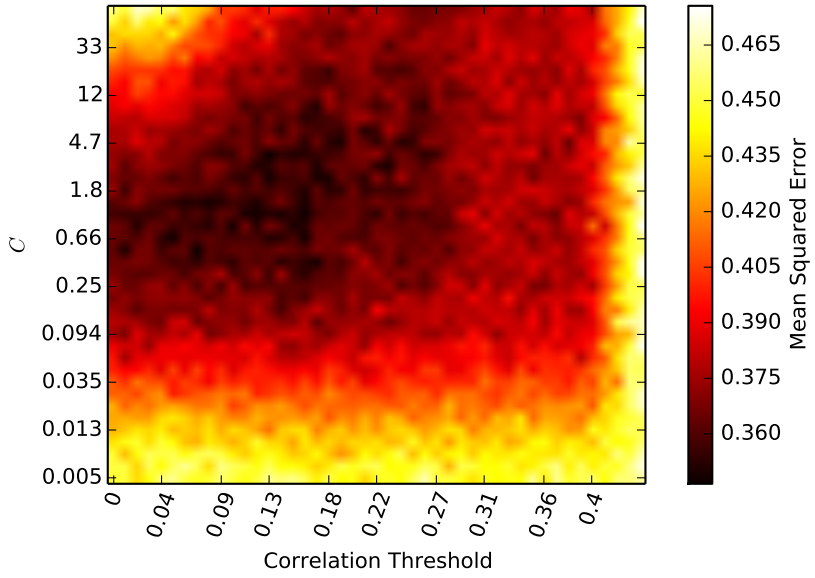


Figure 2.5: Heat map resulting from a grid search over the parameter space of the linear correlation threshold and the cost parameter C of the radial basis functions showing optimal values of 3.6 and 0.13 for γ and the threshold, respectively.

2.1.4 Statistical subtleties

An important note should be made here about some often neglected subtleties relating to uncertainties. Physicists often deal with measurements that can assumed to be independently drawn from a normal distribution $\mathcal{N}(x_i; \mu, \sigma^2)$ due to the central limit theorem. With a large number of measurements n, the standard deviation of a sample

$$\sigma^2 = \frac{1}{N} \sum_i^N (x_i - \mu)^2 \,, \tag{2.50}$$

converges as $N \to \infty$ to the maximum likelihood, minimum variance unbiased estimator for the true variance of the underlying distribution with unknown mean

$$\hat{\sigma}^2 = \frac{1}{N-1} \sum_i^N (x_i - \mu)^2 \,. \tag{2.51}$$

The standard deviation σ and the width of the underlying gaussian $\hat{\sigma}^2$ can then often be used interchangeably. This tempts some people into the

questionable habit of always assuming that the sample standard deviance can be used as the 68% confidence interval of their results.

When using a K-fold validation scheme, the performance scores for the various folds cannot be assumed to be independently drawn from an underlying distribution, as the test set of one fold is used in the training sets of the remaining folds. In fact, it has been shown [1] that there is no unbiased estimator for the variance of the performance estimated using K-fold validation. However, as K-fold validation is more effective than keeping the test, and training data separate, which can be shown using Jensen's inequality along with some basic properties of expectation values [2], I'll mostly use K-fold regardless. As the standard deviation still provides a qualitative measure of the consistency of the model's performance, I'll still use the sample STD in a usual fashion, such as error bars, unless otherwise is specified, but the reader should keep in mind that these do not indicate precise uncertainties whenever K-fold validation has been involved.

2.2 Decision Trees & Random Forests

Another popular machine learning scheme is that of random forests, which consist of an ensemble of decision trees. A decision tree is a very intuitive method for classification problems which can be visualized as a kind of flow chart in the following fashion. As usual, the problem consists of a set of feature vectors $\mathbf{x}_i$ and a set of corresponding class labels y_i. A decision tree then resembles a flowchart starting at the root of the tree, at each node splitting into branches and finally branching into leaves at which all class labels should be identical. At each node, part of the feature vector is used to split the dataset into parts. This resembles the 'twenty questions' game, in which one participant thinks of a famous person and another attempts to guess who it is by asking a series of yes/no-questions, each one splitting the set of candidates in two parts. In this riddle game and in decision tree learning, there are good and bad questions (asking whether the person was born on March 14th, 1879 is a very bad first question, for instance). There are several ways of quantifying how 'good' a yes/no-question, corresponding to a partitioning of the dataset, is.

On metric for this is the Gini Impurity Index I_G, which is computed by summing over each class label:

$$I_G = \sum_i f_i (1 - f_i) = 1 - \sum_i f_i^2, \tag{2.52}$$

where f_i denotes the fraction of the set the consists of class y_i. Using this as a metric, the best partitioning is the one which results in the largest drop in total Gini impurity following a branching. Another metric is the information gain measured by comparing the entropy before a split with a weighted average of the entropy in the groups resulting from the split. Denoting the fractions of the various classes in the parent group, i.e. before splitting, by f_i and the two child groups by a_i and b_i, the information gain is

$$I_E = -\sum_i f_i \log_2 f_i + \frac{n_a}{N} \sum_i a_i \log_2 a_i + \frac{n_b}{N} \sum_i b_i \log_2 b_i. \tag{2.53}$$

However, if too many such nodes are added to a decision tree, overfitting, i.e. extreme accuracies on training data but poor performance on new data, becomes a problem. This can be remedied by instead predicting with a majority vote, or averaging in the case of regression problems, from an ensemble of randomized decision trees called a random forest. The main merits of random forests are their accuracy and ease of use, and their applications as auxiliary methods in other machine learning schemes, which I'll elaborate on shortly.

The individual trees in a random forest are grown using a randomly selected subset of the training data for each tree. The data used to construct a given tree is referred to as 'in bag', whereas the remaining training data is referred to as 'out of bag' (OOB) for the given tree. At each node, a set number of features is randomly selected and the best possible branching, cf. the above considerations, is determined. The only parameters that must be tweaked manually are the number of trees in the forest, number of features to include in each branching, and the maximum tree depth. While other variables such as the metric for determining branching quality as described above, may be customized, those aren't essential to achieve a decent predictor, which is robust in regard to both overfitting and irrelevant parameters.[4]

There doesn't seem to be a single universally accepted way of adjusting these parameters, so I chose a somewhat pragmatic approach of simply checking how well various choices for each parameter performed on a randomly selected trait. For instance, figure 2.6 shows how well a random forest predicted the tertiles of participants' extroversion as a function of the fraction of available features each tree was allowed to include in each branching. This was done using a large number of trees ($n = 1000$) and using each of the two metrics described earlier. The number of features used pr split doesn't seem to have any significant effect on performance, and as the entropy metric seems to perform as well or slightly better than Gini impurity, I decided to stick to that. A similar plot of the performance

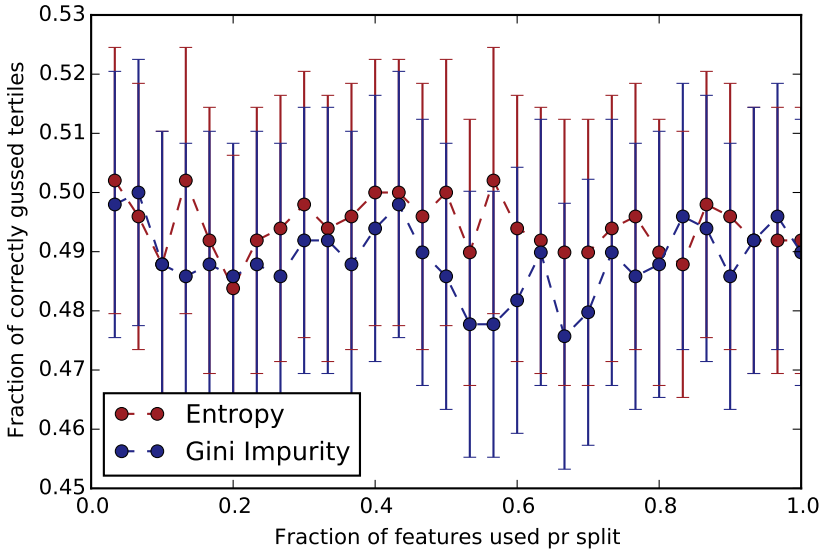


Figure 2.6: Performance of a random forest with 1000 decision trees using various fractions of the available features in each branching using both the entropy and the Gini impurity metric to determine the optimal branching. The number of features seems not to play a major role, and the entropy metric seems to perform slightly better in general.

of various numbers of decision trees in the forest is shown in figure 2.7. The performance seems to stagnate around 100 trees, and remain constant after that, so I usually used at least 500 trees to make sure to get the optimal performance, as runtime wasn't an issue.

The robustness to irrelevant features and overfitting described earlier also plays a role in the application of random forests in conjunction with other schemes. SVMs as described in section 2.1 can be sensitive to irrelevant data[15]. There exist off-the-shelf methods, such as recursive feature elimination (RFE)[10], for use with linear SVMs, but to my knowledge, there is no 'standard' way to eliminate irrelevant features when using a non-linear kernel. However, it is possible to use a random forest approach to obtain the relative importance of the various features and then use only the most important ones in another machine learning scheme which is less tolerant to the inclusion of irrelevant data. The relative importance of feature j can be estimated by first constructing a random forest and evaluating its performance s, then randomly permuting the values of feature j across the training sample and measure the damage it does to

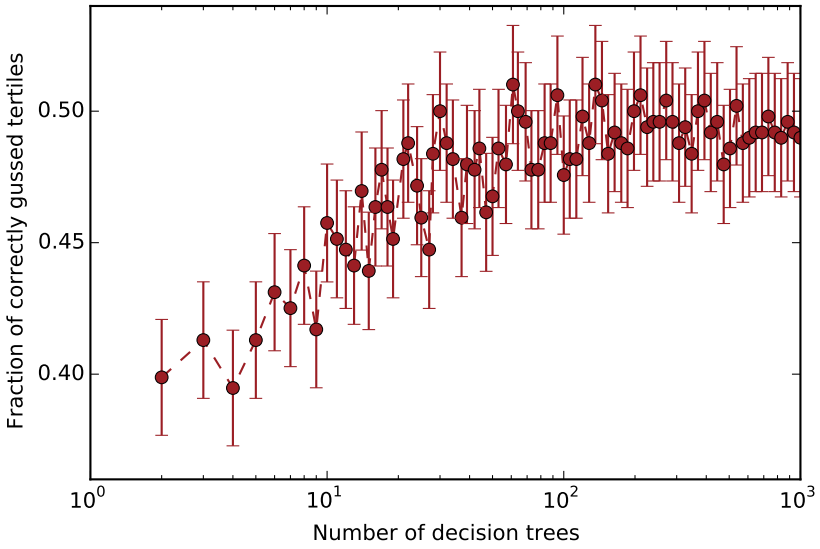


Figure 2.7: Example of a random forest performance versus number of decision trees. Performance seems to increase steadily until about 100 trees, then stagnate.

the performance of the forest by comparing with the permuted score s_p. The ratio of the mean to standard deviation of those differences:

$$w_j = \frac{\langle s - s_p \rangle}{\text{std}(s - s_p)} \tag{2.54}$$

Random forests also provide a natural measure of similarity between data points. Given data points i and j these can be plugged into all their OOB decision trees, or a random subset thereof, and the fraction of the attempts in which both end up at the same leaf can be taken as a measure of similarity. This can be used to generate a proximity matrix for the data points, and it can be used as a metric for determining the nearest neighbours of a new point in conjunction with a simple nearest neighbour classifier.

2.3 Nearest Neighbour-classifiers

Do iiit!

Skriv en masse om den smart random forest NN-model.

2.4 Results

2.4.1 Big Five Personality Traits

HARJ

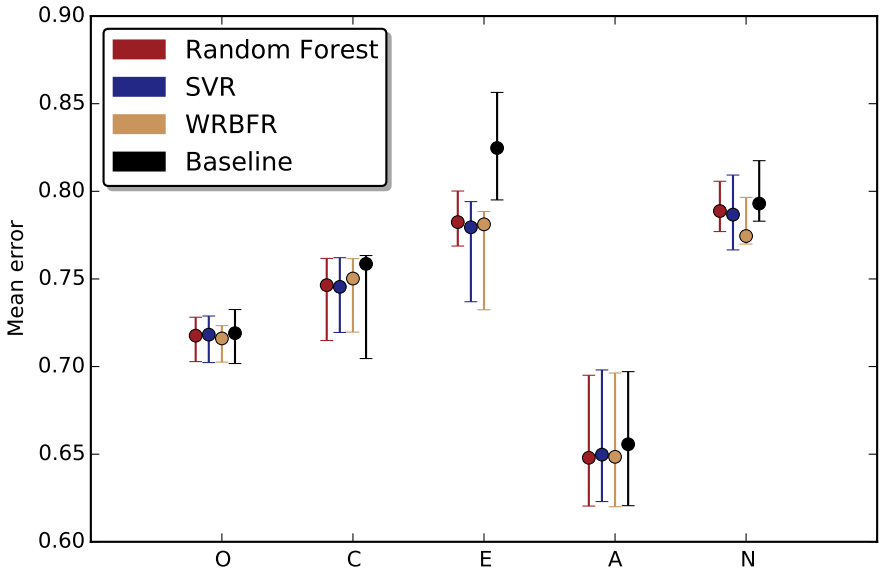


Figure 2.8: Comparison of performance of models using random forest, support vector regression and weighted radial basis function regressor with a **baseline model** which always predicts the mean of the training sample. The y axis shows the mean error of each model and the error bars show the 95-percentile around the median scores obtained by running on 1000 bootstrap samples.

2.4.2 Miscellaneous Traits

Opdater med ML

Her står der ting om figur 2.9.

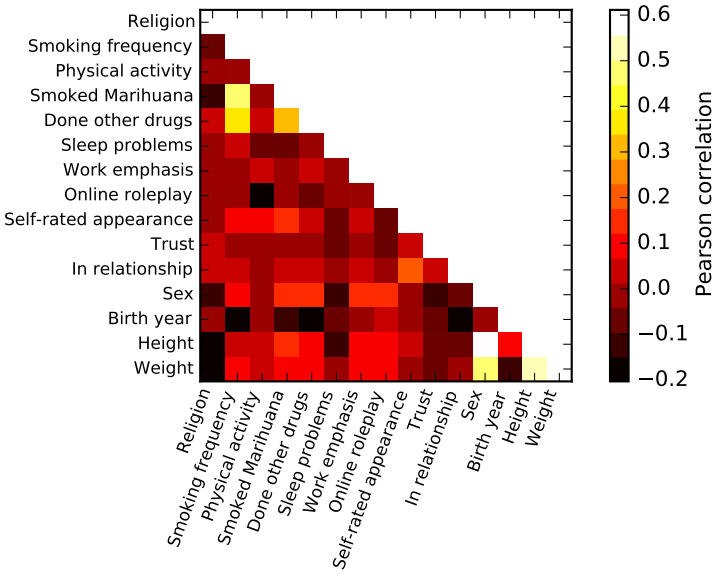


Figure 2.9: Triangle of Pearson correlation coefficients for various miscellaneous traits.

Part II

WIKIPEDIA-BASED EXPLICIT SEMANTIC ANALYSIS

Chapter 3

Wikipedia-based Explicit Semantic Analysis

Natural language processing has long been both a subject of interest and a source of great challenges in the field of artificial intelligence. The difficulty varies greatly depending with the different language processing tasks; certain problems, such as text categorization, are relatively straightforward to convert to a purely mathematical problem, which in turn can be solved by a computer, whereas other problems, such as computing semantic relatedness, necessitates a deeper understanding of a given text, and thus poses a greater problem. This sections aims firstly to give a brief introduction to some of the most prominent techniques used in language processing in order to explain my chosen method of explicit semantic analysis (ESA), and secondly to explain in detail my practical implementation of an ESA-based text interpretation scheme.

ref

3.1 Methods

This section outlines a few methods used in natural language processing, going into some detail on ESA while touching briefly upon related techniques.

3.1.1 Bag-of-Words

ref

An example of a categorization problem is the 'bag of words' approach, which has seen use in spam filters.Here, text fragments are treated as unordered collections of words drawn from various bags, which in the case of spam filters would be undesired mails (spam) and desired mails (ham). By analysing large amounts of regular mail and spam, the probability of drawing each of the words constituting a given text from each bag can be computed, and the probability of the given text fragment representing draws from each bag can be computed using Bayesian statistics.

More formally, the text T is represented as a collection of words $T = \{w_1, w_2, \ldots, w_n\}$, and the probability of T actually representing draws from bag j is hence

$$\begin{aligned} P(B_j|T) &= \frac{P(T|B_j)P(B_j)}{P(T)}, \\ &= \frac{\prod_i P(w_i|B_j)P(B_j)}{\sum_j \prod_i P(w_i|B_j)P(B_j)}, \end{aligned} \tag{3.1}$$

for an arbitrary number of bags labelled by j. This method is simple and powerful whenever a text is expected to fall in one of several discrete categories (such as spam filters or language detection). However, for more complex tasks it proves lucrative to attempt instead to assign some kind of meaning to text fragments rather than to consider them analogous to lottery numbers or marbles. As an intuitive example of the insufficiency of the bag-of-words approach in more complex tasks, consider the problem of determining how related one text fragment is to another, which I'll treat in detail in section 3.3.2 in order to determine the degree in which a media campaign has impacted a series of texts mined from social media. A simplified example of a media campaign by Google for their augmented reality product 'Google Glass' could be the text fragment 'Google unveils Google Glass'. If, in reaction to this, an excited user writes 'Larry Page launches intelligent eyewear', a naive bag-of-words like (3.1) would determine that the words in the user's exclamation could not have been drawn from the 'bag' represented by Google's campaign and so would wrongfully conclude that the two are unrelated. A more reasonable approach would assign some meaning to the words in which 'Larry Page' (Google's CEO) and 'Google', and 'Google Glass' and 'intelligent eyewear' are highly related, just as 'unveil' and 'launch' should be ascribed similar meanings.

This notion of meaning will be elaborated on shortly, as it varies depending on the method of choice, but the overall idea is to ascribe to words a meaning which depends not only on the word itself, but

also on the connection between the word and and existing repository of knowledge. The reader may think of this as mimicking the reading comprehension of humans. In itself, the word 'dog' for instance, contains a mere 24 bits of information if stored with a standard encoding, yet a human reader immediately associates a rich amount of existing knowledge to the word, such as dogs being mammals, related to wolves, being a common household pet, etc. The objective of both explicit and latent semantic analysis is to establish a high-dimensional 'concept space' in which words and text fragments are represented as vectors. The difference between explicit and latent semantic analysis is the method used to obtain said concepts, a explained in the following sections.

3.1.2 Semantic Analysis

Salton et al proposed in their 1975 paper *A Vector Space Model for Automatic Indexing*[16] an approach where words and text fragments are mapped with a linear transformation to vectors in a high-dimensional concept space,

$$T \to |V\rangle = \sum_i v_i |i\rangle, v_i \in \mathbb{R}, \tag{3.2}$$

where a similarity measure of two texts can be defined as the inner product of two normalized such vectors,

$$S(V,W) = \langle \hat{V} | \hat{W} \rangle = \frac{\sum_i v_i w_i}{\left(\sum_i v_i^2\right)\left(\sum_i w_i^2\right)}, \tag{3.3}$$

and the cosine quasi-distance can be considered as a measure of semantic distance between texts:

$$D(V,W) = 1 - S(V,W). \tag{3.4}$$

This approach has later seen use in the methods of Latent Semantic Analysis (LSA) and Explicit Semantic Analysis (ESA). Both methods can be said to mimic human cognition in the sense that the transformation from (3.2) is viewed as a mapping of a text fragment to a predefined *concept space* and thus, processing of texts relies heavily on external repositories of knowledge.

The difference between LSA and ESA is how the concept space is established. Although I have used solely ESA for this project, I will give an extremely brief overview of LSA for completeness following (Landauer 1998 [12]). LSA constructs its concept space by first extracting

every unique word encountered in a large collection of text corpora and essentially uses the leading eigenvectors (i.e. corresponding to the largest eigenvalues) of the word-word covariance matrix as the basis vectors of its conceptual space. This is the sense in which the concept are latent - rather than interpret text in terms of explicit concepts, such as 'healthcare', LSA would discover correlations between words such as 'doctor', 'surgery' etc. and consider that a latent concept. Owing to the tradeoff between performance and computational complexity, only about 400 such vectors are kept[14]. In psychology, LSA has been proposed as a possible model of fundamental human language acquisition as it provides computers a way of estimating e.g. word-word relatedness (a task which LSA does decently) using nothing but patterns discovered in the language it encounters[12].

In contrast, the concepts in ESA correspond directly to certain parts of the external text corpora one has employed to construct a semantic analyser. Concretely, the matrix playing the role of the reduced covariance matrix in LSA has columns corresponding to each text corpus used and rows corresponding to individual terms or words, with the value of each matrix element denoting some measure of relatedness between the designated word and concept. I have used the English Wikipedia, so naturally each concept consists of an article, although the process could easily be tuned to be more or less fine-grained and associate instead each concept with e.g. a subsection or a category, respectively. Of course, a wholly different collection of texts could also be used - for instance a version of ESA more suited to compare the style or period of literary works could be constructed using a large collection of literature such as the Gutenberg Project. However, with no prior knowledge of the subject matter of the text to be analysed, Wikipedia seems like a good all-round solution considering its versatility and the massive numbers of volunteers constantly keeping it up to date.

I wish to point out two advantages of ESA over LSA. First, it is more successful, at least using the currently available text corpora and implementation techniques. A standard way of evaluating the performance of a natural language processing program is to measure the correlation between relatedness scores assigned to pairs of words or text fragments by the computer and by human judges. In both disciplines, ESA has outperformed LSA since it was first implemented ([9], p 457).

Second, the concepts employed in ESA are directly understandable by a human reader, whereas the concepts in LSA correspond to the leading moments of the covariance matrix. For example, to test whether the first semantic analyser built by my program behaved reasonably, I fed it a snippet of a news article on CNN with the headline "*In Jerusalem,*

the 'auto intifada' is far from an uprising". This returned an ordered list of top scoring concepts as follows: "Hamas, Second Intifada, Palestinian National Authority, Shuafat, Gaza War (2008-09), Jerusalem, Gaza Strip, Arab Peace Initiative, Yasser Arafat, Israel, West Bank, Temple Mount, Western Wall, Mahmoud Abbas", which seems a very reasonable output.

3.2 Constructing a Semantic Analyser

The process of applying ESA to a certain problem may be considered as the two separate subtasks of first a very computationally intensive construction af the machinery required to perform ESA, followed by the application of said machinery to some collection of texts. For clarity, I'll limit the present section to the details of the former subtask while description its application and results in section 3.3.

The construction itself is divided into three steps which are run in succession to create the desired machinery. The following is a very brief overview of these steps, each of which is elaborated upon in the following subsections.

1. First, a full Wikipedia XML-dump[1] is parsed to a collection of files each of which contains the relevant information on a number of articles. This includes the article contents in plaintext along with some metadata such as designated categories, inter-article link data etc.

2. Then, the information on each concept (article) is evaluated according to some predefined criteria, and concepts thus deemed inferior are purged from the files. Furthermore, two list are generated and saved, which map unique concepts and words, respectively, to an integer, the combination of which is to designate the relevant row and column in the final matrix. For example, the concept 'Horse' corresponded to column 699221 in my matrix, while the word 'horse' corresponded to row 11533476.

3. Finally, a large sparse matrix containing relevance scores for each word-concept pair is built and, optionally, pruned (a process used to remove 'background noise' from common words as explained in section 3.2.3)

These steps are elaborated upon in the following.

[1] These are periodically released at http://dumps.wikimedia.org/enwiki/

3.2.1 XML Parsing

The Wikipedia dump comes in a rather large (~50GB unpacked for the version I used) XML file which must be parsed to extract each article's contents and relevant information. This file is essentially a very long list of nested fields where the data type in each field is denoted by an XML tag, such as `<text> blabla </text>`. A very simplified example of a field for one Wikipedia article is shown in 3.1 The content of each field has

```
<page>
  <title>Horse</title>
  <ns>0</ns>
  <id>12</id>
  <revision>
    <id>619093743</id>
    <parentid>618899706</parentid>
    <timestamp>2014-07-30T07:26:05Z</timestamp>
    <contributor>
      <username>Eduen</username>
      <id>7527773</id>
    </contributor>
    <text xml:space="preserve">
===Section title===
[[Image:name_of_image_file] Image caption]]]
Lots of educational text containing, among other things links to [[other article|text to display]].
</text>
    <sha1>n57mnhttuhxxpq1nanak3zhmmmcl622</sha1>
    <model>wikitext</model>
    <format>text/x-wiki</format>
  </revision>
</page>
```

Snippet 3.1: A simplified snippet, of a Wikipedia XML dump.

already been sanitised by Wikipedia so that if for instance the symbol '<' is entered into an article, it is instead represented as '&lt' in the XML file. To this end, I wrote a SAX parser, which processes the dump sequentially to accommodate its large size. When running, the parser walks through the file and sends the various elements it encounters to a suitable processing function depending on the currently open XML tag. For example, when a 'title' tag is encountered, a callback method is triggered which assigns a column number to the article title and adds it to the list of processed articles. For the callback method for processing the 'text' fields, I used a bit of code from the pre-existing Wikiextractor project to remove Wiki markup language (such as links to other articles being displayed with square brackets and the like) from the content. This code is included in section A.2.5. The remainder of the parser is my work, and is included in

section A.2.1. Throughout this process, the parser keeps a list of unique words encountered as well as outgoing link information for each article. These lists, along with the article contents, are saved to files each time a set number of articles have been processed. The link information is also kept in a hashmap with target articles as keys and a set articles linking to the target as values. The point of this is to reduce the computational complexity of the link processing as detailed in the following section.

3.2.2 Index Generation

The next step reads in the link information previously saved and adds it as ingoing link information in the respective article content files. The point of this approach is that link information is initially saved as a hashmap so the link going to a given article can be found quickly, rather that having to search for outgoing links in every other article to determine the ingoing links to each article, which would be of $O(n^2)$ complexity.

Following that, articles with sufficiently few words and/or ingoing/-outgoing links are discarded an index lists for the remaining articles and words are generated to associate a unique row/column number with each word/concept pair. The code performing the step described here is included in section A.2.2.

3.2.3 Matrix Construction

This final step converts the information compiled in the previous steps into a very large sparse matrix. The program allows for this to be done in 'chunks' in order to avoid insane RAM usage. Similarly, the matrix is stored in segments with each file containing a set number of rows in order to avoid loading the entire matrix to interpret a short text.

The full matrix is initially constructed using a DOK (dictionary of keys) sparse format in which the i, jth element simply counts the number of occurrences of word i in the article corresponding to concept j. This is denoted count(w_i, c_j). The DOK format as a hashmap using tuples (i, j) as keys and the corresponding matrix elements as values and is the fastest format available for element-wise construction. The matrix is subsequently converted to CSR (compressed sparse row) format, which allows faster operations on rows which performs much quicker when computing TF-IDF (term frequency - inverse document frequency) scores and extracting concept vectors from words, i.e. when accessing separate rows corresponding to certain words.

Each non-zero entry is then converted to a TF-IDF score according to

$$T_{ij} = \Big(1 + \ln\Big(\text{count}(w_i, c_j)\Big)\Big) \ln\left(\frac{n_c}{df_i}\right), \tag{3.5}$$

where n_c is the total number of concepts and

$$df_i = |\{c_k, w_i \in c_k\}| \tag{3.6}$$

is the number of concepts whose corresponding article contains the ith word. Thus, the first part of (3.5), $1 + \ln\Big(\text{count}(w_i, c_j)\Big)$ is the *text frequency* term, as it increases with the frequency of word i in document j. Similarly, $\ln\left(\frac{n_c}{df_i}\right)$ in (3.5) is the *inverse document frequency* term as it decreases with the frequency of documents containing word i. Thus, the TF-IDF score as somewhat complement to entropy in that it goes to zero as the fraction of documents containing word i goes to 1, and takes its highest values if word i occurs with high frequency in only a few documents[13]. While (3.5) is not the only expression to have those properties, empirically it tends to achieve superior results in information retrieval[17].

Giver det mening?

Each row is then L^2 normalized (divided by their Euclidean norm):

$$T_{ij} \to \frac{T_{ij}}{\sqrt{\sum_i T_{ij}^2}}. \tag{3.7}$$

Finally, each row is pruned to reduce spurious associations between concepts and articles with a somewhat uniform occurrence rate. This was done in practice by following the pragmatic approach of Gabrilovich [9] of sorting the entries of each row, move a sliding window across the entries, truncating when the falloff drops below a set threshold and finally reversing the sorting. The result of this step is the matrix which computes the interpretation vectors as described in 3.1.2. The code is included in section A.2.3.

Hearst nævner noget offentligt tilgængeligt Reuters-data som folk øver tekstklassifikation på. Det kunne være ret sjovt. Det kunne være sjovt at lave 'semantisk nearest neighbor'

3.3 Applications & Results

Having constructed a necessary machinery, I wrote a small Python module to provide an easy-to-use interface with the output from the computations described earlier. The code for this is included in section A.2.4. The module consists mainly of a SemanticAnalyser class, which loads in the previously mentioned index lists and provides methods for various

computations such as estimating the most relevant concepts for a text, determining semantic distance etc. For example, the following code will create a semantic analyser instance and use it to guess the topic of the input string:

```
sa = SemanticAnalyser()
sa.interpret_text("Physicist from Austria known for the theory of relativity")
```

This returns a sorted list of the basis concepts best matching the input string, where the first element is of course 'Albert Einstein'. The SemanticAnalyser class contains equally simple methods to interpret a target text file or keyboard input, to calculate the semantic similarity or cosine distance between texts, and to compute interpretations vectors from a text.

The same module contains a TweetHarvester class which I wrote in order to obtain a large number of tweets to test the semantic analyser on, as tweets are both numerous and timestamped, which allows investigations of the temporal evolution of tweets matching a given search term. The TweetHarvester class provides an equally simply interface - for instances, the 100 most recent tweets regarding a list of companies can be mined and printed by typing

```
terms = ['google', 'carlsberg', 'starbucks']
th = TweetHarvester()
th.mine(terms, 100)
print th.harvested_tweets
```

in addition to actively 'mining' for tweets matching a given query, the class can also passively 'listen' for tweets while automatically saving its held tweets to a generated date-specific filename once a set limit of held tweets is exceeded:

```
th = TweetHarvester(tweets_pr_file = 100)
th.listen(terms)
```

The downloaded tweets are stored as tweet objects which contain a built in method to convert to a JSON-serializable hashmap, an example of which is provided in 3.2. As can be seen in the example, the tweet object contains not only the tweets textual content but also a wide range of metadata such as the hashtags contained in the tweet, users mentioned, time of creation, language etc. Using this, I wrote a script that harvested tweets mentioning some selected brand names for about 5 months which resulted in a massive dataset of 370 million tweets. This section demonstrates some applications of the methods outlined earlier to this dataset.

```
{u'contributors': None,
 u'coordinates': None,
 u'created_at': u'Wed Jun 03 12:16:23 +0000 2015',
 u'entities': {u'hashtags': [],
               u'symbols': [],
               u'urls': [],
               u'user_mentions': [{u'id': 630908729,
                                   u'id_str': u'630908729',
                                   u'indices': [0, 12],
                                   u'name': u'Alexandra White ',
                                   u'screen_name': u'lexywhite86'}]},
 u'favorite_count': 0,
 u'favorited': False,
 u'geo': None,
 u'id': 606071930282745857L,
 u'id_str': u'606071930282745857',
 u'in_reply_to_screen_name': u'lexywhite86',
 u'in_reply_to_status_id': 605991263129714688L,
 u'in_reply_to_status_id_str': u'605991263129714688',
 u'in_reply_to_user_id': 630908729,
 u'in_reply_to_user_id_str': u'630908729',
 u'is_quote_status': False,
 u'lang': u'en',
 u'metadata': {u'iso_language_code': u'en', u'result_type': u'recent'},
 u'place': None,
 u'retweet_count': 0,
 u'retweeted': False,
 u'source': u'<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone
     </a>',
 u'text': u'@lexywhite86 if carlsberg did mornings \U0001f602',
 u'truncated': False,
 u'user': {u'contributors_enabled': False,
           u'created_at': u'Thu Apr 05 13:48:00 +0000 2012',
           u'description': u'',
           u'favourites_count': 169,
           u'follow_request_sent': False,
           u'followers_count': 110,
           u'friends_count': 268,
           u'geo_enabled': False,
           u'id': 545986280,
           u'id_str': u'545986280',
           u'lang': u'en',
           u'listed_count': 0,
           u'location': u'',
           u'name': u'Robert Murphy',
           u'notifications': False,
           u'protected': False,
           u'screen_name': u'7Robmurphy',
           u'statuses_count': 1650,
           u'time_zone': None,
           u'url': None,
           u'utc_offset': None,
           u'verified': False}}
```

Snippet 3.2: Example of a downloaded tweet.

3.3.1 Trend Discovery and Monitoring

A simple application of the methods outlined above is a purely exploratory analysis. As an example, using code included in section A.2.6, I extracted the 10 concepts most closely related to every tweet mentioning Carlsberg for each week over a period of a few months. This gave a bar chart like figure 3.1 for each of those weeks. Figure 3.1 implies some significant relation between Carlsberg and Raidió Teilifís Éireann (RTE), the Irish national radio and television service. I then discovered that RTE usually does a quarterly report on Carlsberg's earnings, and that some web sites encouraged their followers to drink Carlsberg and listen to RTE's coverage of the Eurovision Song Contest taking place in week 18 of 2014.

Another application is to manually select a few concepts of interest and then monitor how the number of tweets strongly related to them develops over time. For example I did an exploratory analysis as described above and occasionally saw the concept 'Kim Little' surface. This turned out to be a young football player signed to a team sponsored by Carlsberg, and when I produced a series of bar charts like that shown in figure 3.2 and combined them into an animation, the bar representing Little tended to peak around dates on which news stories featuring her or her team could be found. Of course these method are only effective insofar the concepts of interest actually have an associated Wikipedia article. However, these concepts are merely the basis vectors of the semantic space in which an arbitrary text can be represented, so this procedure can be extended fairly elegantly to estimate the impact of e.g. a press release on social media. This is the subject of section 3.3.2.

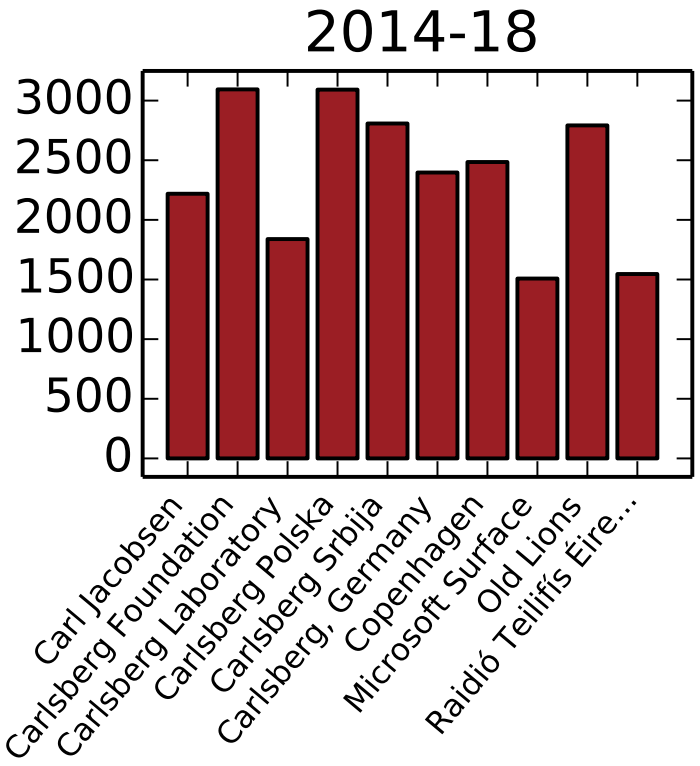


Figure 3.1: Bar chart representing the number of times various concepts appeared in the top 10 most semantically related concepts for each given tweet taken from week 18 of 2014.

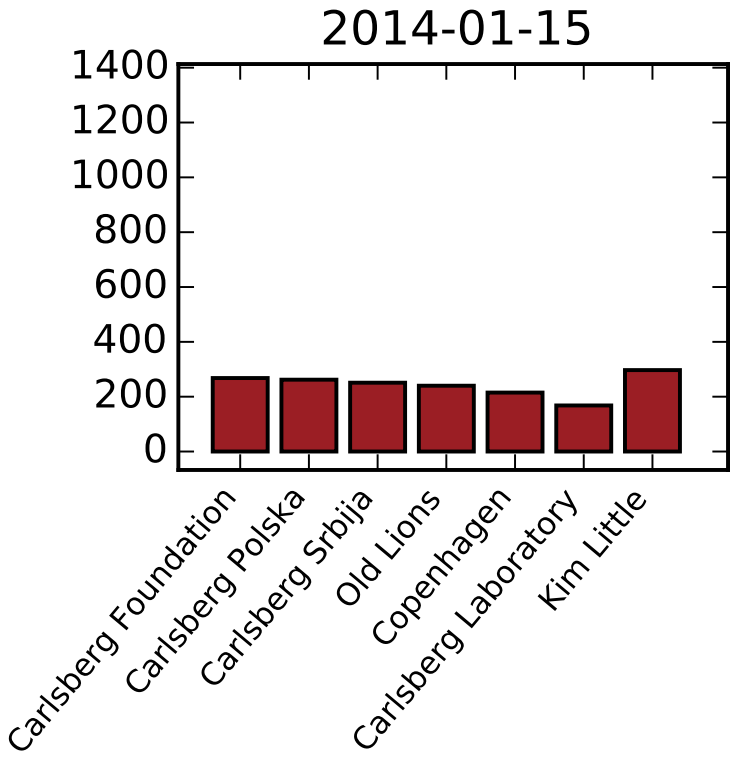


Figure 3.2: Bar chart showing the concepts most related to tweets about Carlsberg for January 15th 2014.

3.3.2 Measuring Social Media Impact

One possible application of the software I wrote to perform ESA is to provide a quantitative measure of the impact of some event on social media. For instance, some corporation or organization might be interested in learning precisely effectively a campaign or press release has reached the general public as it is expressed by social media. While my tweet harvesting script was running, Google posted a blog entry[2] on their experiments with producing psychedelic images with deep neural network called *Deep Dreams* which received widespread attention on social media.

Using the text from the Deep Dreams blog post as a reference text, I converted each English tweet about Google from a period around the blog post to a semantic vector using (3.2) and computed their semantic cosine similarity with the reference text as described by (3.3). Figure 3.3 shows this behaviour in the time around the release of the blog post. A measure of impact should of course take into account the rate at which new tweets occur. Just as one would expect, not only semantic relatedness, but also tweet frequency increase drastically around an interesting event. To make the signal from figure 3.3 independent of the normal tweet frequency, yet sensitive to changes in it, I first find the difference in semantic relatedness between the signal and a reference signal obtained long before the event to be investigated, then modulate the signal by multiplying each bin with the activity A given by the ratio between the current and baseline tweet frequency. As normal Twitter activity tends to vary greatly over a 24 hour period, I obtained a baseline tweet frequency by computing the average tweet rate, weighted by the activity, for each discreet time interval of the day, and then repeating that signal into the period after the event of interest. The baseline along with the observed tweet rate is shown in figure 3.4(a). Modulating in this fashion and computing the difference between the observed signal and the baseline gives an expression of the 'impact rate' of a given time bin, which can be integrated to obtain a measure of the total impact which is independent of the average tweet rate before publication, which is practical if one wishes to compare e.g. the success of media campaigns for companies or organizations of varying sizes. This is shown in figure 3.5. If one wishes to include the number of tweets posted in the impact measure, the result can be computed and visualized nicely by again obtaining a baseline cosine similarity from a period prior to publication and then considering e.g. the cumulative deviation from that following the release, as shown in figure 3.6.

[2]The original blog post is available at http://googleresearch.blogspot.dk/2015/06/inceptionism-going-deeper-into-neural.html

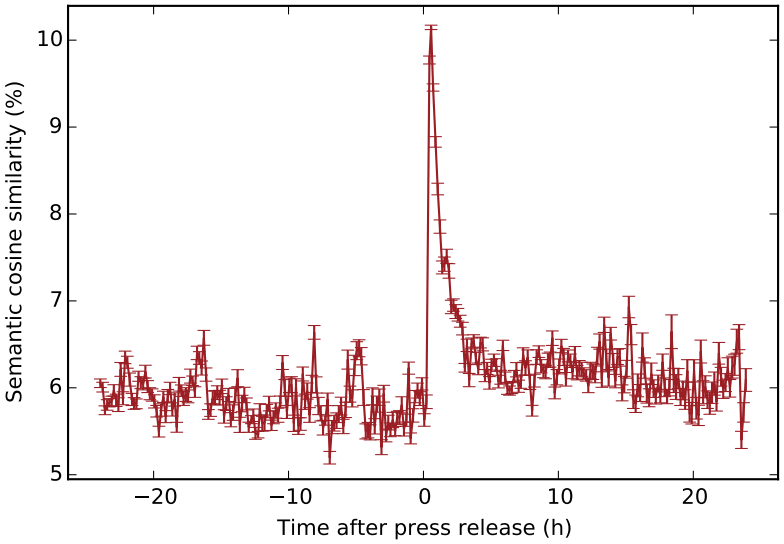
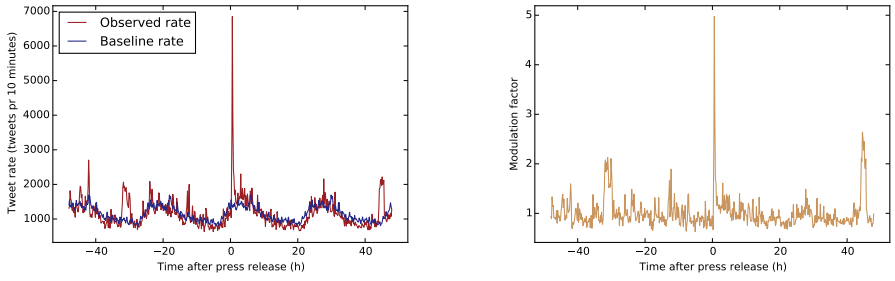


Figure 3.3: Graph of the mean semantic cosine similarity of tweets around the Deep Dreams press release. There is a clear peak around $t = 0$ and the similarity appears increased in the period following the release. The error bars represent the true standard deviation of the sample mean $\sigma/\sqrt{N}$ for each time bin, each representing a 10-minute interval.



(a) (b)

Figure 3.4: Tweet activity around the time the Deep Dreams blog entry was posted. The signal tweet rate increases with a factor of about 5 relative to the baseline rate around the post. Figure 3.4(b) shows the resulting modulation factor to the signal from in figure 3.3.

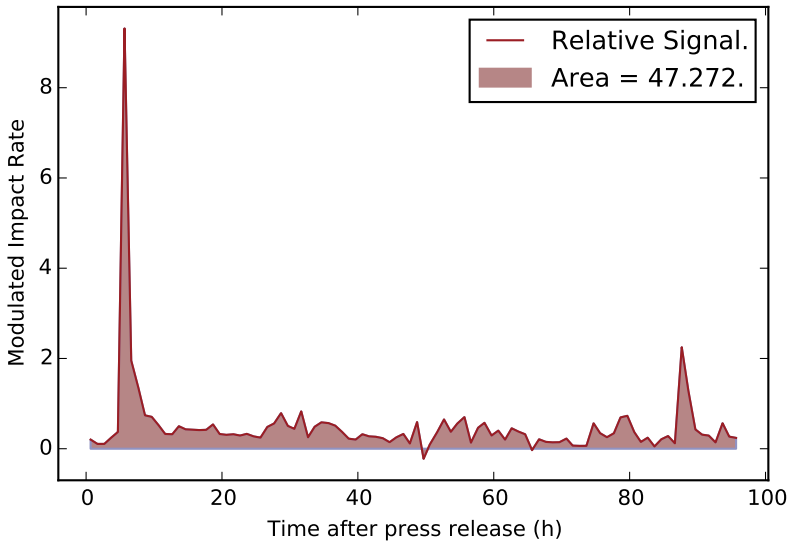


Figure 3.5: A measure of the total impact on Twitter by a press release can be obtained by integrating over the difference between a signal corresponding of the average tweet cosine similarity modulated by the relative activity, and a corresponding baseline. This measure does not depend on the typical Twitter activity and so can be used to compare the effectiveness of campaigns of varying size.

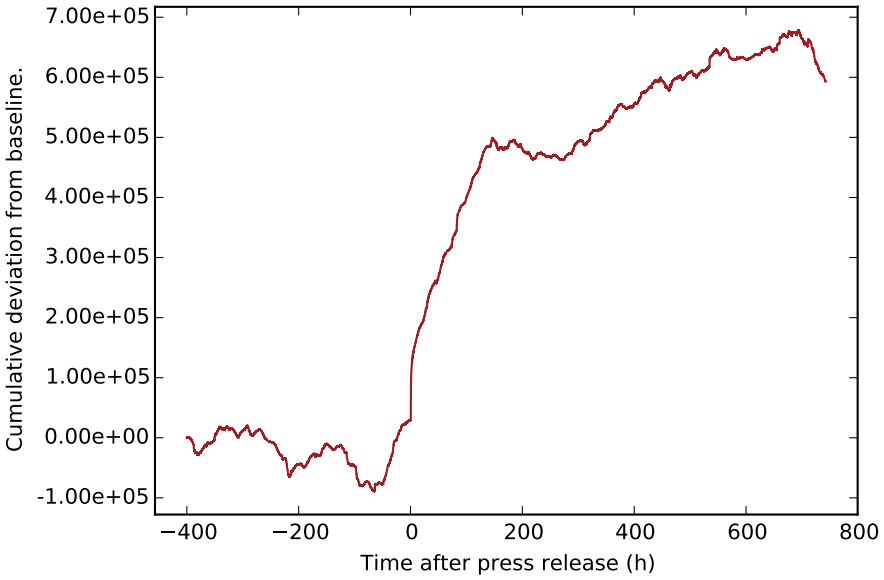


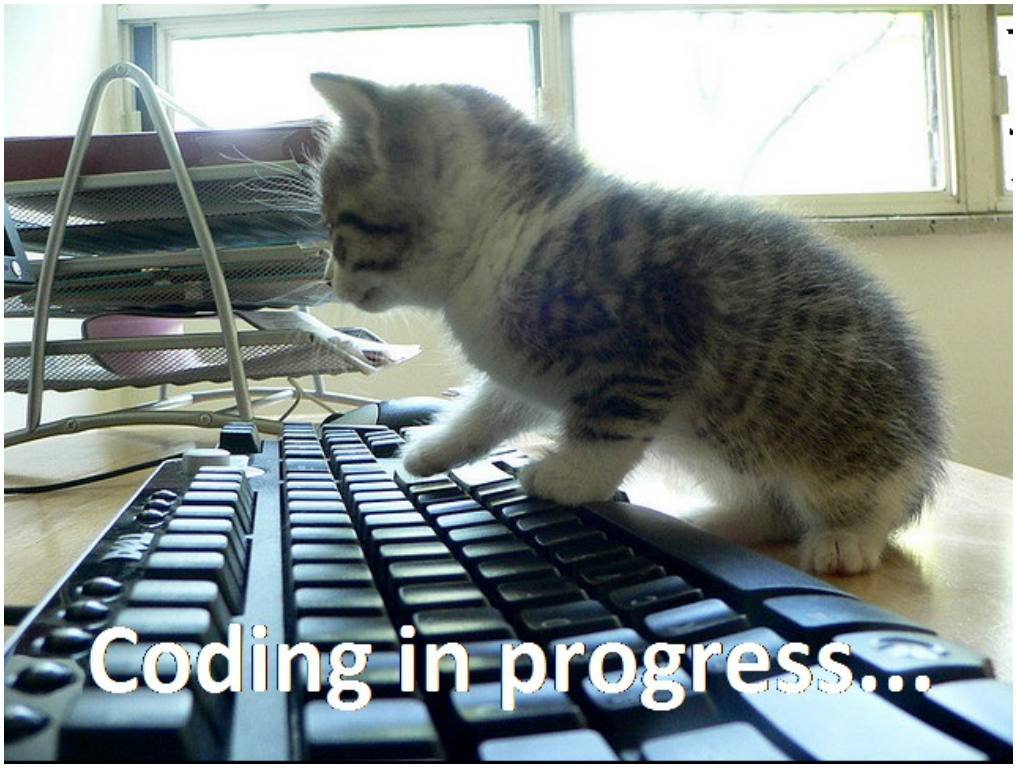
Figure 3.6: The cumulative deviation from the mean cosine similarity between the Deep Dreams post and tweets in the period around its release.

Appendix

A.1 Social Fabric-related Code

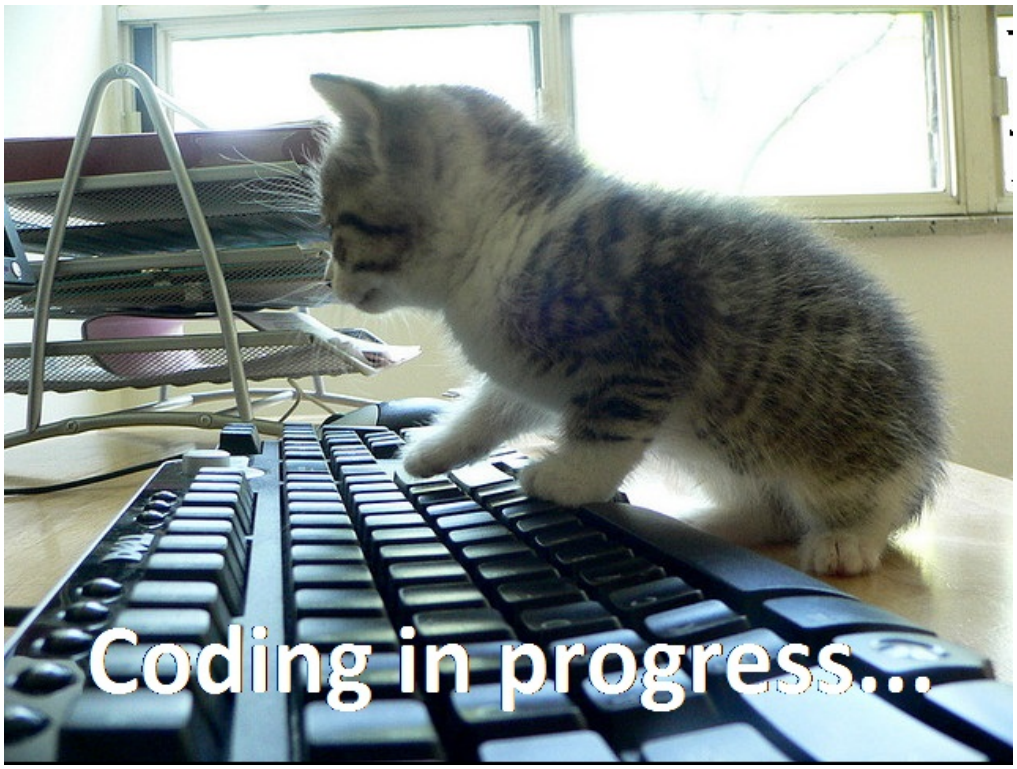
This section contains the code referred to in part I of the thesis.

A.1.1 Phonetools

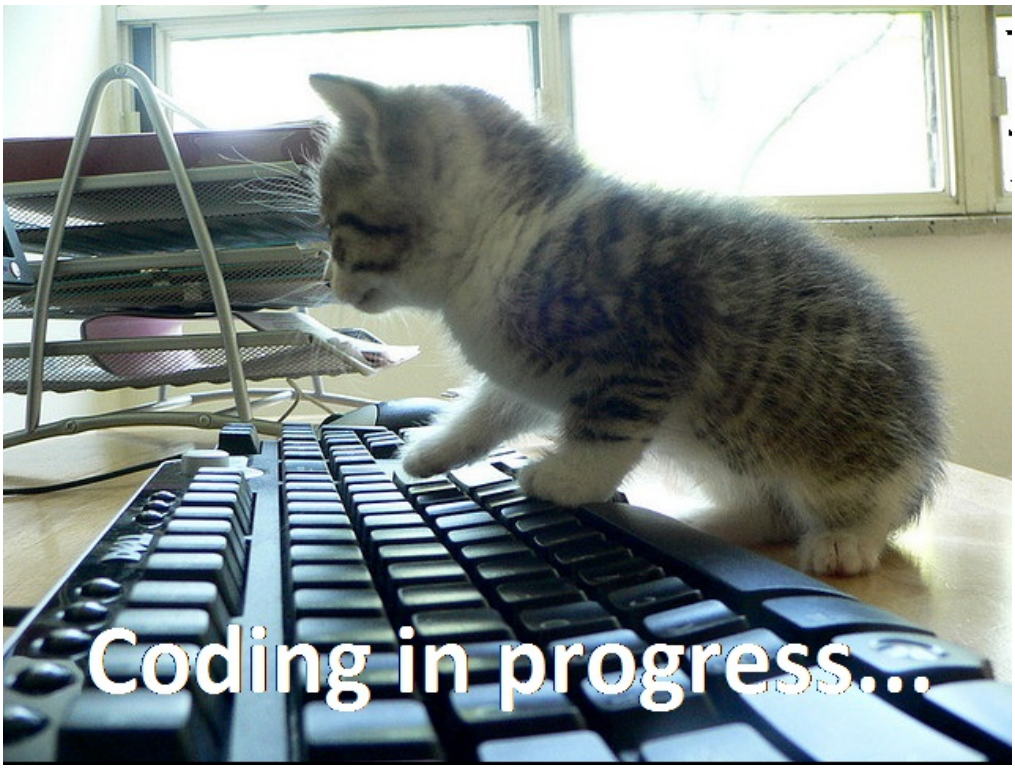


A.1.2 Code Communication Dynamics

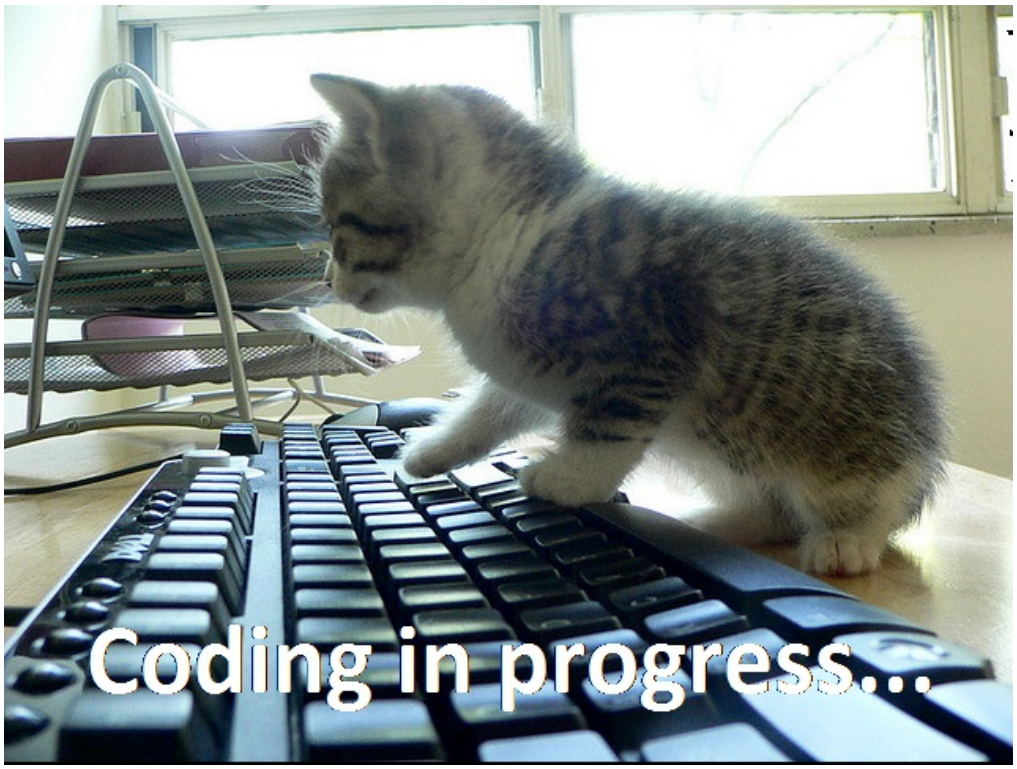
Code for extracting Bluetooth signals



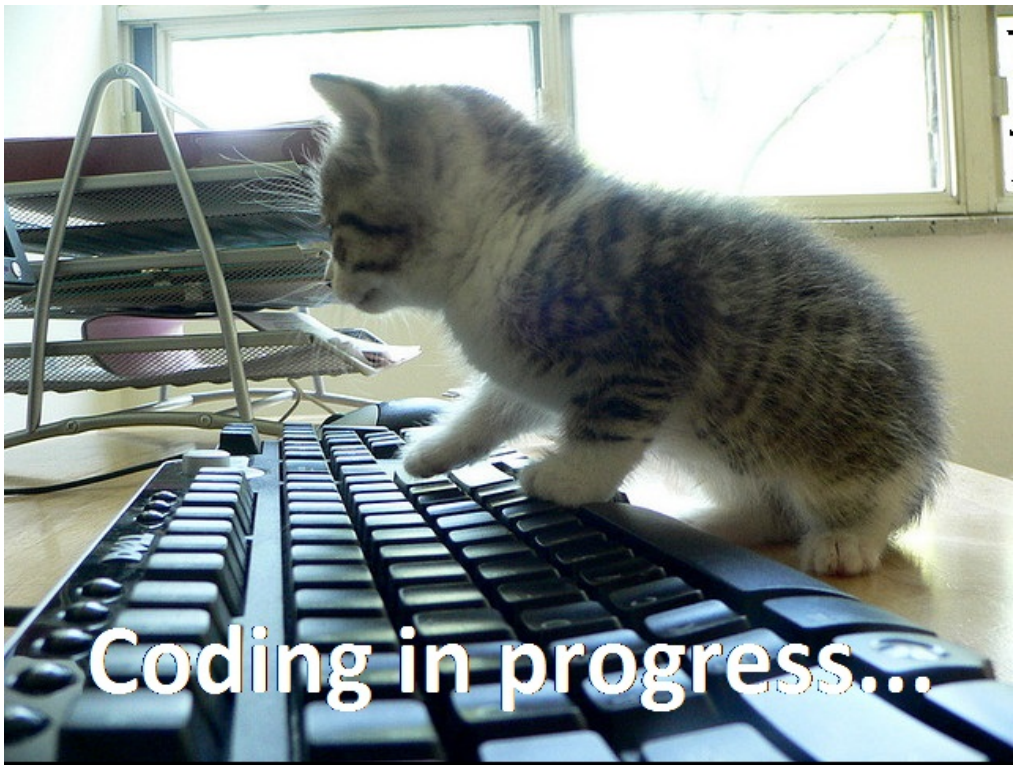
Code for loading and plotting Bluetoot data


Coding in progress...

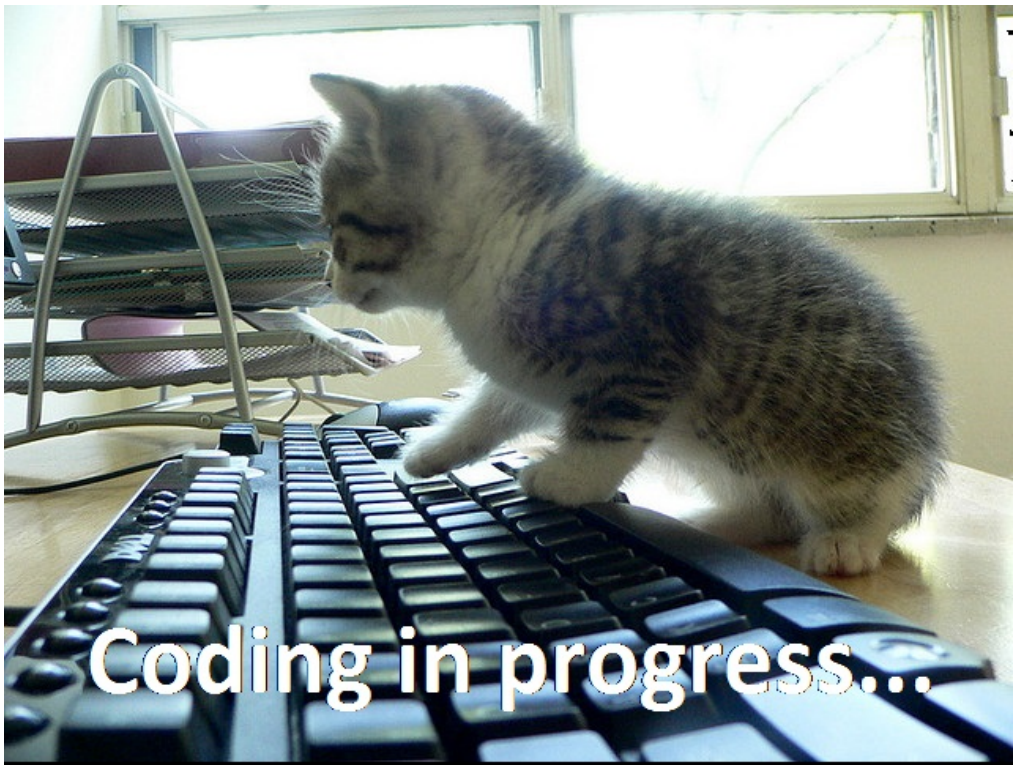
A.1.3 Preprocessing



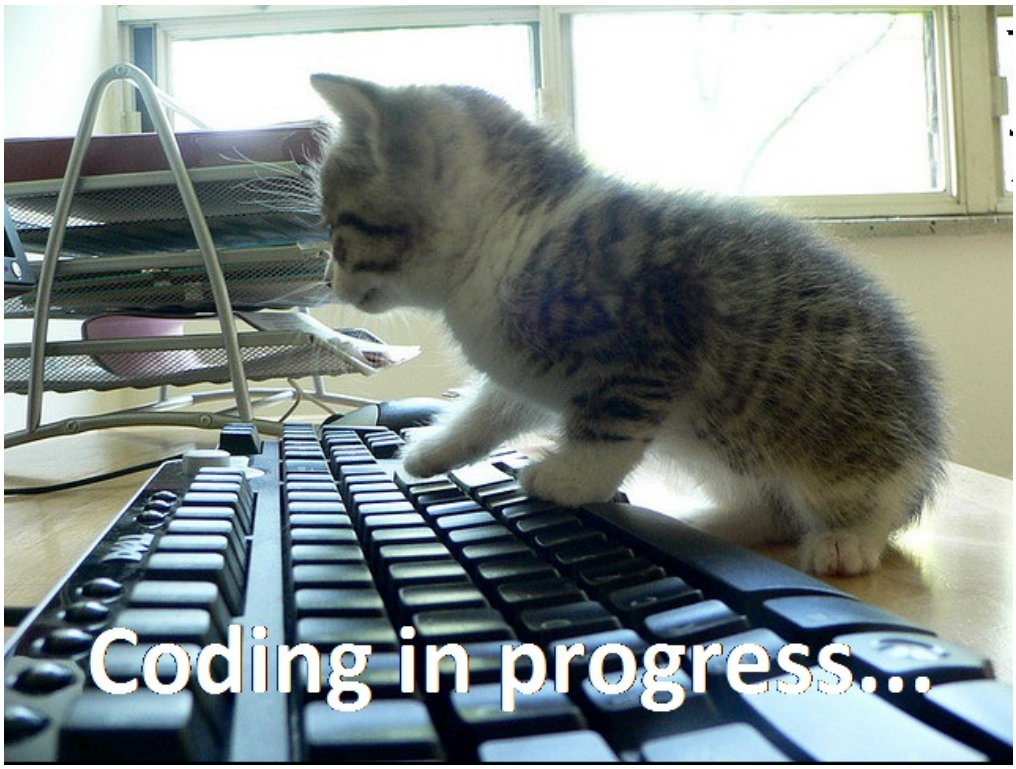
A.1.4 Social Fabric Code



A.1.5 Lloyd's Algorithm



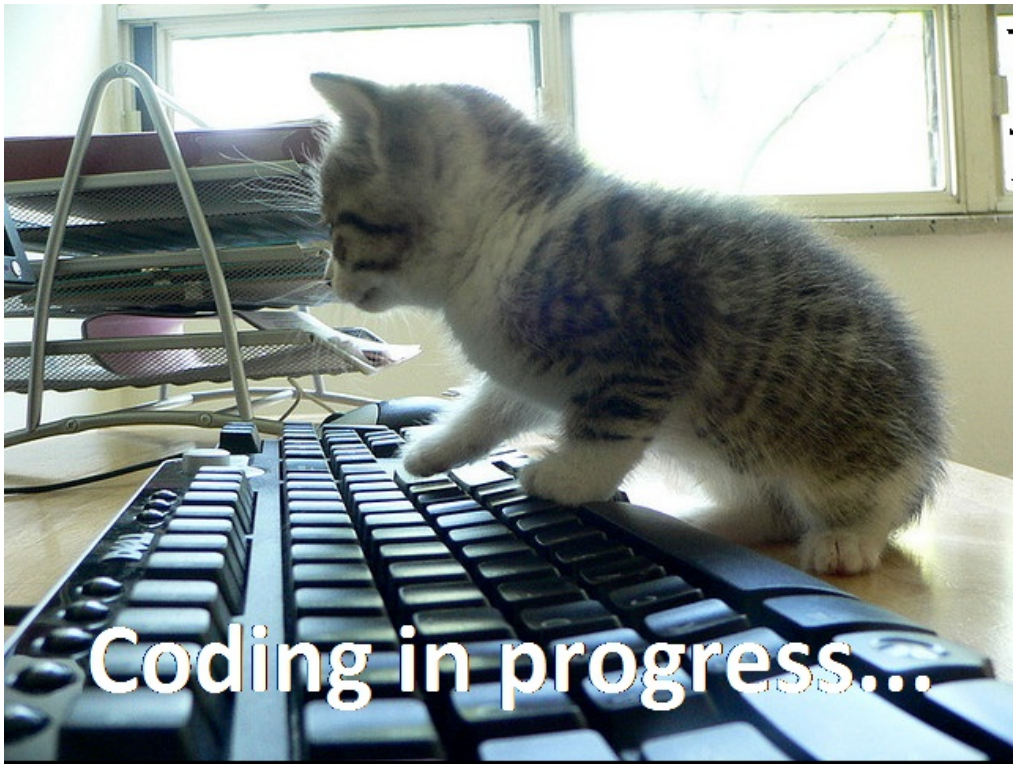
A.1.6 Smallest Enclosing Circle


Coding in progress...

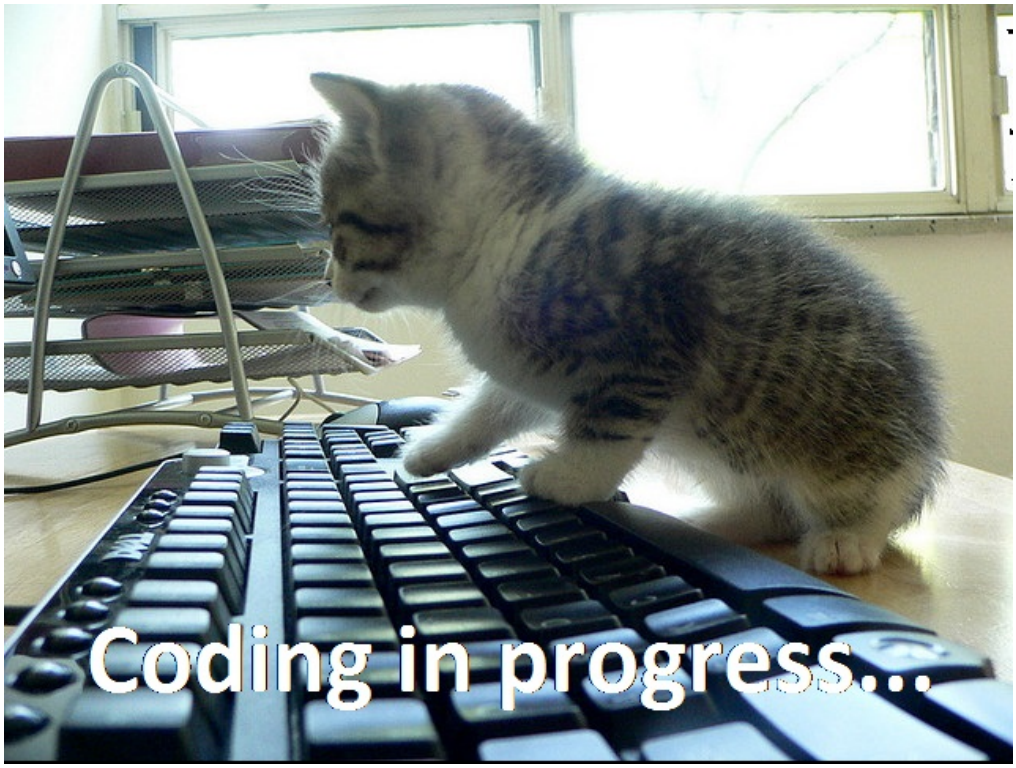
A.2 Source Code for Explicit Semantic Analysis

This section contains code pertaining to part II of the thesis.

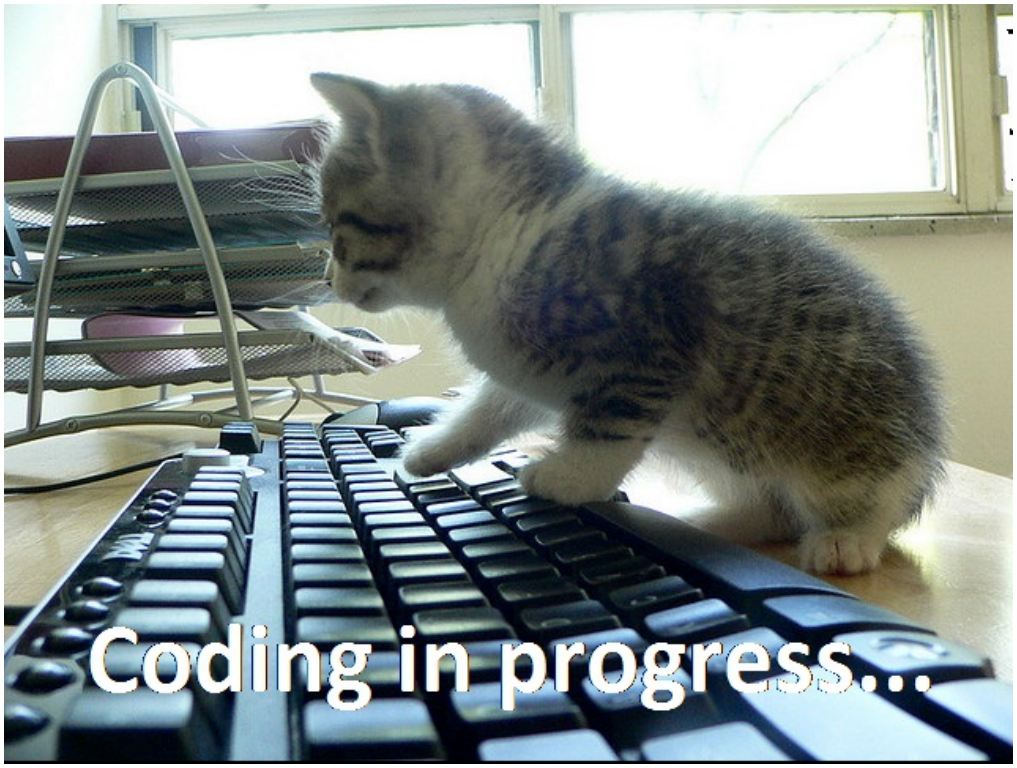
A.2.1 Parser



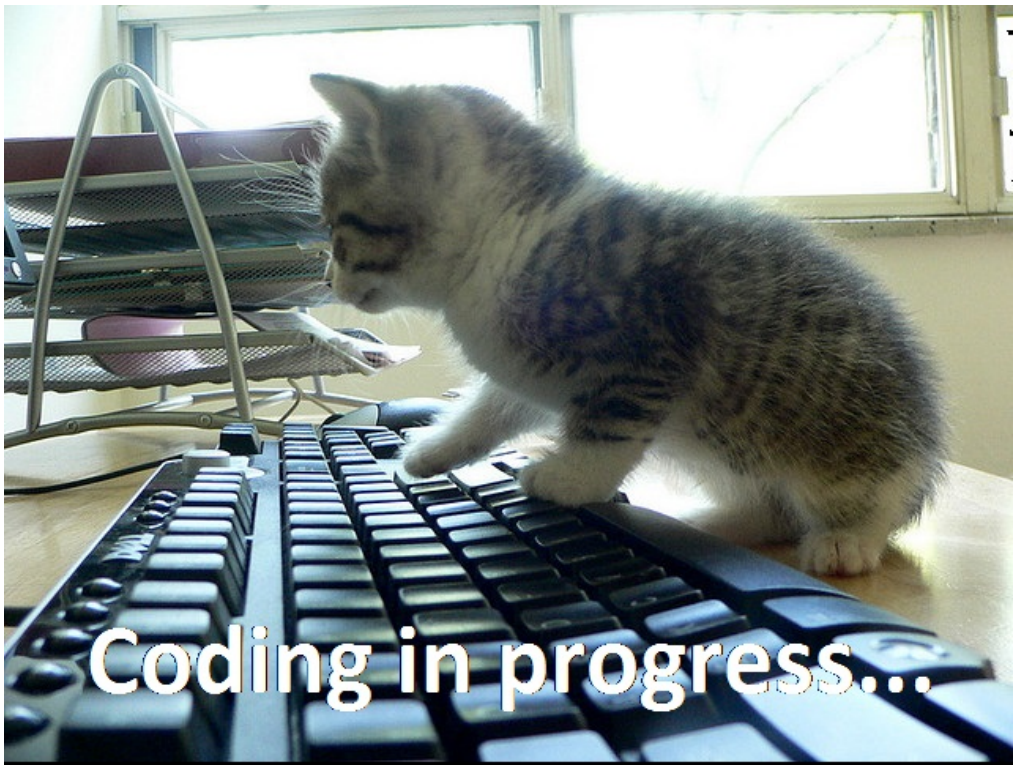
A.2.2 Index Generator



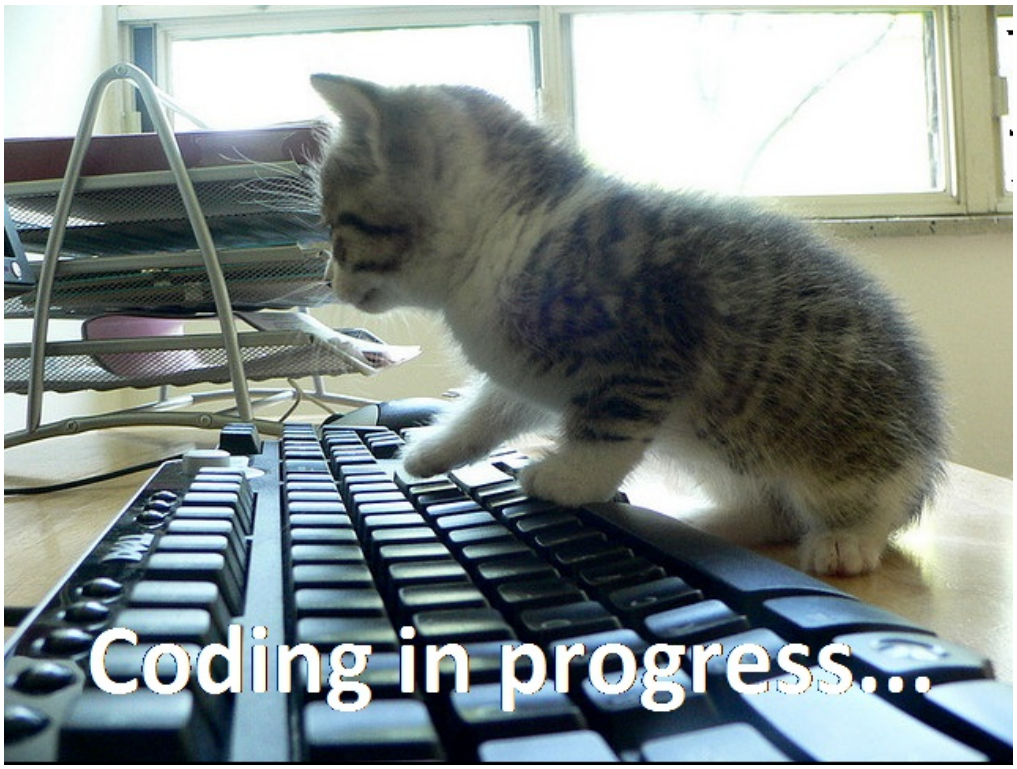
A.2.3 Matrix Builder



A.2.4 Library for Computational Linguistics



A.2.5 Wikicleaner

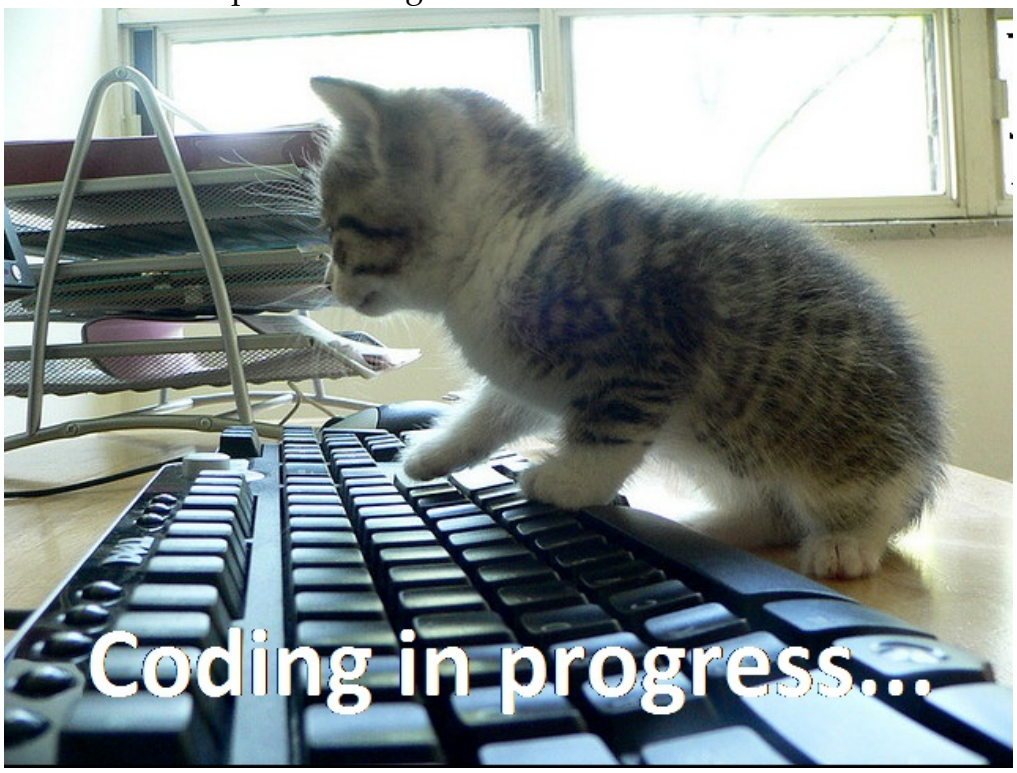


A.2.6 Application Examples

This section contains code used for the examples of applications of explicit semantic analysis described in section 3.3.

Trend Discovery and Monitoring

This is the script used to generate the results seen in section 3.3.1.



Social Media Impact

This is the code used to extract and analyse the data described in section 3.3.2.

Bibliography

[1] Yoshua Bengio and Yves Grandvalet. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. 5:1089–1105, 2004.

[2] Avrim Blum, Adam Kalai, and John Langford. Beating the Hold-out: Bounds for {K}-fold and Progressive Cross-Validation. *Proceedings of the 12th Annual Conference on Computational Learning Theory*, (c):203–208, 1999.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] L Breiman. Random forests. *Machine learning*, pages 5–32, 2001.

[5] Cjc Christopher J C Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[6] Yves Alexandre De Montjoye, Jordi Quoidbach, Florent Robic, and Alex Pentland. Predicting personality using novel mobile phone-based metrics. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7812 LNCS:48–55, 2013.

[7] JM John M Digman. Personality structure: Emergence of the five-factor model. *Annual review of psychology*, 41:417–440, 1990.

[8] RA Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 1936.

[9] E Gabrilovich and S Markovitch. Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. *IJCAI*, 2007.

[10] Isabelle Guyon. Gene Selection for Cancer Classification. pages 389–422, 2002.

[11] Marti a. Hearst, Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Schölkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[12] Thomas K Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. An Introduction to Information Retrieval. *Online*, (c):569, 2009.

[14] Preslav Nakov. Latent semantic analysis of textual data. *Proceedings of the conference on Computer systems and technologies - CompSysTech '00*, pages 5031–5035, 2000.

[15] AD Pietersma. Kernel Learning in Support Vector Machines using Dual-Objective Optimization. *Proceedings of the 23rd . . .* , 2011.

[16] G. Salton, a. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[17] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[18] AJ Smola and B Schölkopf. *Learning with kernels*. 1998.

[19] AJ Smola and B Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 2004.

[20] Philip Wolfe. A duality theorem for nonlinear programming. *Quarterly of applied mathematics*, 19(3):239–244, 1961.

[21] PA Zandbergen and SJ Barbeau. Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. *Journal of Navigation*, 2011.