Category Clustering

Exploring feature creation and similarity in clustering of cancer patients

Kristoffer Langerød

Thesis submitted for the degree of
Master in Programming and Networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2016

Category Clustering

Exploring feature creation and similarity in clustering of cancer patients

Kristoffer Langerød

Category Clustering

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The human DNA is a 3.1 billion long string of organic molecules, represented by four unique letters, one for each type of molecule in the chain. This string is physically divided into 23 separate pairs, folded to save space and protect against damage. When a cell is dividing through mitosis, this folded structure change to make it possible, but also exposing itself to alterations or damage. Changes are in most cases dealt with by defence mechanisms, but some times they are more severe and can lead to cancer, an uncontrollable growth of cells. Extensive research has been put forward to find better treatment, identifying cancer earlier and to identify the cell of origin. The latter being in the scope of this thesis, as I introduce different approaches, trying to find sub groups of cancer in affected patients.

This was done by using machine learning, a subset of artificial intelligence, where the goal is to find patterns or build models to identify objects. Since the ideal result is to find something that does not yet have a definitive answer, clustering, the group of machine learning algorithms that tries to identify patterns with unlabelled data, was used.

In it's essence, a clustering algorithm take in representations for each object and returns a grouping. For this to be possible, the representation of objects usually has to be numeric values with a possibility of distinguishing them by their shared attributes. And for several reasons, this representation is stored as a vector which can be used with distance measures such as Euclidean- and Manhattan-distance to calculate similarity between objects.

Traditional distance measures have three rules attached to them. One of the rules is that identical vectors should have zero distance. I argue that this does not always make sense. In the case of two people living 0km from Oslo and two other people living 1,000km from Oslo, any distance measure obeying this rule would mark both pairs of people as identical to each other. But not having a property does not make for as strong of a connection as actually having this property. Thus I have implemented distance measures to accommodate the idea that sharing a property is a stronger indication of similarity.

This also spurs out of how the numeric properties of objects are calculated, by using a reference set of information about DNase I Hypersensitive sites, which relates to active sites. So the objects are not compared directly but by how they relate to certain parts of the reference set.

Another rule for traditional distance measures is that the distance from object A to C is always smaller or equal to the distance of A to B plus B to C. I also argue that this does not always make sense, as objects can be

similar in different ways. A and B can be similar in one way, B and C in a different way and A and C can be completely dissimilar. To make this effect possible, the reference set is divided into sub sets and similarity between objects can be in one or more of these sets. When a similarity is established within a part of the reference set, there are locking mechanics that stops the transitive effect from occurring.

The methods developed was used to cluster 1889 donors, each with number of mutations ranging from 1,000 to 10,000, and 163 reference files of DNase I Hypersensitive site data to help create the representations. Results show that clustering with implemented methods yield a significantly higher probability than by chance to group donors by their cancer type. Also, the probability of donors being grouped so that their cancer type matches the tissue type of reference sub-sets, blindly chosen by the clustering algorithm, was with certain distance measure and arguments shown to be significantly higher than by chance.

Contents

List of Figures

List of Tables

Preface

The original title of this thesis was "Making sense of the human genome using machine learning", a broad take on bioinformatics with machine learning. This was what initially drove me to pick the topic, since I wanted to do something of potential importance, not just informatics for informatics sake. I had also at that point just been introduced to machine learning, and loved the idea of working with it further. As the topic narrowed, clustering and experimentation with unorthodox techniques related to features and similarity was the main driver.

Not just building on established methods, and due to the combination of biology and informatics related topics, made the process challenging. I had almost no knowledge of biology and my experience with machine learning and mathematics was fairly limited. But it has been a great learning experience altogether, when looking back.

First of all I want to thank my supervisor Geir Kjetil Sandve, for always being positive and available, even at inopportune times with a busy schedule. I also want to thank Niklas Jacobsen and Snorre Lærum for the best lunch breaks and conversations.

My parents deserve a particular thanks for supported me through the more difficult times of the process.

Part I

Introduction

0.0.1 The problem

The goal of this thesis is to explore how to incorporate data sets with relevant information and using machine learning to try to identify sub clusters of cell type from donors with somatic mutations.

Identifying the cell of origin for cancer is not yet completely accomplished, and the goal for this project is not to completely get there, but to see if a different approach will yield meaningful results.

Target group

The thesis is written as to be understood by a master student with general knowledge of informatics. Biologically concepts are introduced and discussed, but no former knowledge in biology should be necessary to understand these parts.

Motivation

Making a computer program that learns something new, even if the program doesn't know what it knows, is an enticing idea in itself. Even more so when making a computer program that learns something useful. In this day and age of information and rising cancer rates, this seems like a meaningful and necessary combination to explore further.

The end goal with trying to identify cell types and cell of origin for a cancer patients is very important, as it will make it easier to choose and develop treatment more suited for that exact type of cancer.

Research method

Throughout the project, tools has been implemented both on the a Galaxy instance and locally, to test and explore the concepts.

0.0.2 Thesis structure

Background

This part explains basic biology and machine learning which forms the basis of the thesis. These concepts are needed to understand the project and the motivations behind the decisions made.

Distance and similarity

Ways to represent objects are introduced, ranging from simple to more intricate. Then distance measures with different interpretations of similarity, related to object representation are discussed. Finally concepts of non transitive relation between objects and feature importance is introduced.

Implementation

This summarises how the implementation was done and what different parts of the code base does. It also explains the final algorithm and how to use the code.

Results

Results is a discussion of how different methods performed. Plots of the resulting clustering is shown to highlight different features and behaviour of the algorithm, feature creation and distance measures.

Chapter 1

Background

1.1 Genome

The genome is the entire heritage information of an organism, stored in the organism's deoxyribonucleic acid (DNA).

The physical structure of DNA molecules in humans are double stranded helices, built up by four types of nucleotides. Each consist of one of the bases; guanine, adenine, thymine or cytosine, which is usually denoted by their first letter when working with strings of DNA.

Each of the strands are directional, specified by the fifth and third carbon in the sugar molecule the bases are attached to, called 5′ (five prime) and 3′ (three prime). 5′ signals the start of a nucleotide sequence and 3′ signals the end. The strands run in opposite directions and usually contain the same information, so only one of the strands are needed to code genetic information, and only one side is stored after sequencing the genome (the genetic material of an organism).

1.1.1 Genome structure

The genome includes both genes, which is the part of the DNA that codes into proteins, and the parts in between, called non-coding sequences. The non-coding sequences makes up the wast majority of the human genome, but this does not mean that the non-coding sequences are junk. The ENCODE project found that 80 percent of the human genome serves some purpose, more than just coding proteins [1].

The genome is not just a long continuous straight string of base pairs, if it was it would not fit into the cell nucleus, but it is divided into physical folded structures as shown in figure 1.1. A part of this structure is called chromosomes and consists of the DNA string, wrapped around protein complexes (nucleosomes) of histones. Healthy humans have 23 pair of chromosomes, with the difference between male and females being that males have YX pairs and female XX pairs. Other than making it fit into cells, the packing of DNA also makes it more protected against outside interference, like solar radiation.

Chromosomes are only visible with a light microscope when the cell

undergoes mitosis (cell division). The reason is that the physical structure change during replication. The folded and packed DNA is hard to copy, so the DNA has to get more accessible for the RNA to do its work.

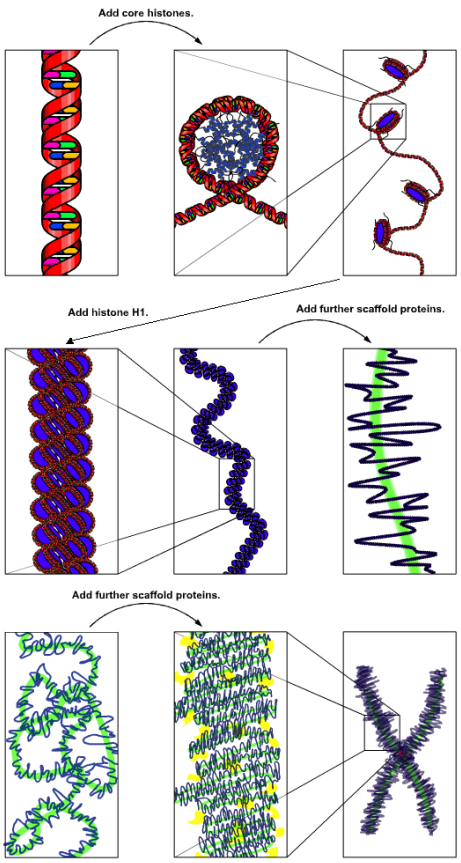


Figure 1.1: The packing of DNA from double helix to chromosome

Chromosomes, its proteins and macromolecules are together called chromatin and has the job of folding to make the DNA fit into the cell, protect the DNA against damage and to control gene expression and replication.

1.2 The Genomic HyperBrowser

The Genomic HyperBrowser (HB) is a web-based free software system for statistic analysis of genomic data. It is based on the Galaxy Project[2] and started out as a collaboration between the University of Oslo and The Norwegian Radium Hospital. It was released in 2010 and has a wide range of genomic data in form of annotated genomic tracks, or just "tracks", from basic track types with points to extended track types with linked function. HB also supports the use of user-uploaded files or files from external databases such as ENCODE, Cancer Genome Atlas and ICGC, among other [3].

1.3 Machine Learning

Machine learning is a branch of artificial intelligence (AI) with the purpose of learning from data and deriving models that tells us something about the structure of the data or recognize new unseen objects. There are several types of machine learning algorithms available, with supervised- and unsupervised learning as the most general and common.

1.3.1 Features

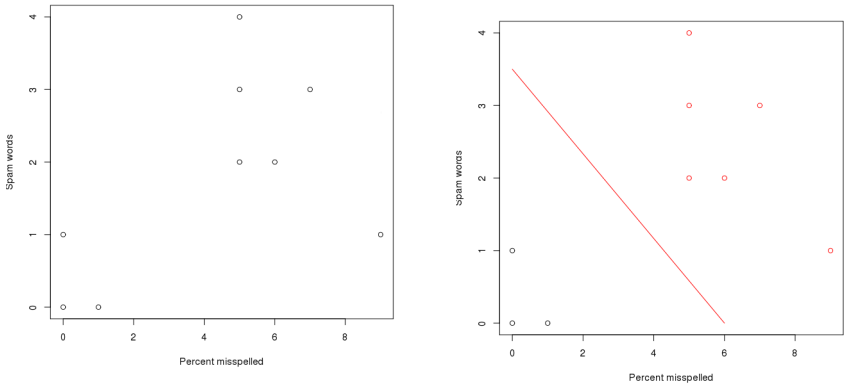
Machine learning algorithms needs specific inputs to be able to work with the objects being used. These are usually called features. A feature is a measurable property of an object, meaning features represent different

aspects of an object. If the objects were animals and the goal was to make a model which identifies baby elephants. The features could be; number of legs, weight, height, age, ear circumference, nose length and so on. They are numeric so general numeric functions can work on them, and together they are described as feature vectors, one vector for each object. This makes it easier to imagine objects as points in an n-dimensional vector space. The basic idea is that points that are close together are more similar than points that lay far apart and should be put in the same cluster or class. Most clustering methods take advantage of this representation.

1.3.2 Supervised learning

Supervised learning methods, usually called classification, works in two steps. The first step, called training, revolves around analysing a set of data with labelled examples to adjust functions, then use the acquired generalized knowledge to label new unseen data. This requires the desired output to be known, and the labelled data the algorithms train on usually has to be labelled or constructed by humans to some extent.

A traditional example of supervised learning is to build a model for deciding if e-mails are spam or not. To simplify the example, let the feature vectors representing each object be [x,y] where x is the percentage misspelled words in the mail and y is the number of spam words used (such as Viagra, Nigerian prince, weight loss etc.). Assigning imaginary example e-mail with this feature vector template, lets them be represented in a two dimensional plane as shown in figure 1.2a.



(a) Spam e-mail objects with two features

(b) With decision boundary

Figure 1.2: E-mail spam example

Supervised learning models use labelled example to build the model. In figure 1.2a there is a visible gap separating the upper right and the lower left points, so for convenience sake, say the upper right points are examples labelled as spam, while the lower left points are not spam. The gap between the classes will usually contain some kind of boundary. In some centroid based algorithms this boundary is imaginary, represented as

one point in each class, where the the boundary is defined as the centre of centroid points. In other algorithms only the boundary is represented in the final model, as one or more lines. The figure 1.2b shows the final model with the decision boundary marked.

After the model is crated, deciding if new mails are spam or not is only a task of figuring out if the e-mail, after assigned a feature vector, lays in the spam area or not.

This is a simplified example, but the same principles applies for more complex models with more objects and higher dimensions.

1.3.3 Unsupervised learning

Unsupervised learning is a set of algorithms that does not require a training set to be labelled with the correct interpretation, but instead uses unlabelled data. The main goal with unsupervised learning is not to generalize mapping from input to output but to discover structures of the data. This includes clustering (k-means, hierarchical etc.), where objects that are similar (close in the n-dimensional vector space) gets clumped together, and hidden Markov models (HMM) that makes probabilistic models, among other.

Choosing number of clusters

Classification has a fixed predefined number of classes, given by the number of unique class labels of the training data. Clustering on the other hand has not. This can be a real challenge as many clustering algorithms require the number of clusters to be set in advance, such as the commonly used k-means where the k refers to the number of clusters. Setting this variable is not trivial, as too high numbers will reduce the error but introduce artificial sub clusters. Too low numbers will on the other hand not capture the structures of the data. There exist heuristic methods to minimize this problem, such as the use of gap statistics [4], but in general it makes runtime longer and the creation of models more complex.

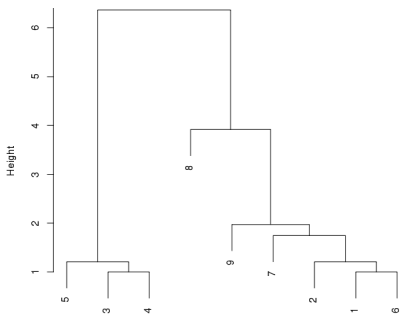


Figure 1.3

Hierarchical clustering algorithms omits this entirely. The agglomerative variant starts with each object as it's own cluster, then combining clusters until only one remains, storing information about what objects/clusters gets grouped together, at what point and the distances between them. This way the number of clusters can be set afterwards, cutting the tree result as desired.

Clustering the training data from the spam example in subsection 1.3.2 using hclust (Hierarchical

clustering) in the statistical language R, gives dendrogram plot 1.3. The y-axis (height) shows the distance between clusters. The cluster cut-off might be at distance 5 as it gives two fairly similar clusters, or at distance 2.5 if it is desired to have two clusters with more inner cluster similarity, and one single cluster/object. It all depends on the data, results, personal preference, goals with the clustering and so on.

1.3.4 Machine Learning in Biology

Today one of the big challenges involving large amount of data is within the field of biology. A sequence of one human's genome is over 3.2 billion base pairs and takes up at least 800MB of disk space (3.2 billion letters, each coded as two bits). Generally the size of the files are way larger, since the genome is often stored in many different formats and usually are annotated.

In general most analysis does not concern itself with complete genomes, but rather information about certain interesting features or functions. For example has the ICGC Data Portal [5] lots of data sets available containing information related to mutated genes and simple somatic mutations, ignoring the base pairs which is not relevant.

Machine learning advantages in bioinformatics

Data in bioinformatics tend to be noisy and have missing information. This is troublesome for traditional statistical methods. Machine learning on the other hand has the ability to deal with this much better due to the adaptive nature of the algorithms being used. Also the wast amount of data is not necessarily a problem for machine learning methods, but can rather be beneficial for the models to be accurate.

1.3.5 Data imbalance

The genomic data available in bioinformatics can also cause problems, even for machine learning, with possibly 3.2G data points in a single file and lacking information on relevant cases. Most machine learning techniques rely on the data being relatively balanced to get a satisfying performance from the resulting model.

If the only data available for training the spam classification model in subsection 1.3.2 were the mails sent to our e-mail account. This would probably be thousands of e-mails accumulated over several years, with the majority of them being labelled as not spam. Assuming 99 percent of all emails received are not spam, the classifier would be correct in 99 out of 100 cases by just classifying everything as not spam. This might seem like a good performance, but then the model has no real predictive properties. If on the other hand the classifier labelled 80 out of 99 of the normal e-mails as spam and every spam e-mail as spam, the classifier would be correct 81 percent of the times and there would be no spam in the inbox, but

	Class A	Class B
Correctly labelled	**T**rue **P**ositive	**T**rue **N**egative
Wrongly labelled	**F**alse **P**ositive	**F**alse **N**egative

Figure 1.4: Explanation of TP, FP, TN and FN.

potentially lots of important messages could get lost due to the aggressive spam filtering.

This leads to the four main methods of analysing the performance of models that cover binary classification like the one in our example. The acronyms used are explained in figure 1.4.

- Accuracy is a measure of overall correctness in labelling the objects.

  $Accuracy = (TP + TN)/(TP + TN + FP + FN)$

- Precision, or positive predictive value, measures the correct labels of a class, ignoring the other class(es).

  $Precision = TP/(TP + FP)$

- Recall, known as sensitivity, measures the ability to select objects of a specific class.

  $Recall = TP/(TP + FN)$

- F-score is the harmonic mean of precision and recall.

  $F - score = 2 * ((Precision * Recall)/(Precision + Recall))$

With two classes, A and B, figure 1.4 shows the cases of true positive, true negative, false positive and false negative, where "Class A" means that the element is labelled by the model as a member of class A, and the same for class B, when we look at the classifiers ability to predict class A.

In the lazy case where the spam model labels everything as not spam the scores would be:

- 0.99 accuracy
- 0.00 precision
- 0.00 recall
- 0.00 F-score

In the aggressive case where the spam model gets 81 out of 100 correct the scores would be:

- 0.81 accuracy
- 0.05 precision
- 1 recall

- $\sim$0.095 F-score

The scores given are not that great, even if all the cases of spam are correctly predicted and the majority of the non-spam cases are correctly predicted. One underlying problem in this case is the data imbalance, ie. the number of members in each class is very different.

1.3.6 Handling data imbalance

To simplify things, I will focus on data imbalance in classifying problems like the spam or not-spam problem, where there are just two classes.

There are different ways to handle imbalanced data sets. The simplest way is to either up-sample data from the minority class by randomly duplicating data called Random minority Over Sampling (ROS), or down-sample data from the majority class by randomly discarding data called Random majority Under Sampling (RUS). A conceptual problem with both solutions is that there is really no information being added with ROS, there is even an danger of the model over fitting the data as some objects are represented more than once. In the case of RUS, information is actually being removed.

Other more clever methods have been developed, like One Sided Selection (OSS) which removes objects from the majority class that are redundant or noisy. To define what is redundant can be problematic, and removing noise can also result in over fitting.

A more complex method is the Synthetic Minority Oversampling Technique (SMOTE) where you find the k nearest neighbours of each element of the minority class and add new elements by combining the pre-existing examples.

It has been reported that the "naive" sampling techniques generally works best but that no sampling technique works best in every case, it all depends on the data sets [6]. There has also been suggested that tweaking the parameters of the machine learning method itself could yield better result then sampling, and having the benefit of needing no sampling before training the model.

1.4 Machine learning usage in bioinformatics

There are may techniques within the field of machine learning that is interesting in the context of genome analysis. Artificial Neural Network (ANN), a supervised learning algorithm, which has been one of the dominant forms of machine learning from the 70's until today, is a model that is based on the structure of the brain. It uses interconnected neurons for sending signals forward and also usually backwards to calibrate or correct the network to recognizing patterns, and has been used to analysing DNA sequences [7]. It has also been used to predict the sequence of the human TP53 tumour suppressor gene based on a p53 Gene Chip [8].

1.4.1 OncodriveCLUST

The goal of this research project was to distinguish genomic alterations that are involved in making tumors and those that occur stochastically as a by-product of cancer development, by using the positional clustering of somatic mutations to identify cancer genes [9].

Their method consists of:

- Retrieve single-nucleotide protein-affecting mutations
- Find potentially meaningful cluster seeds by identifying positions with a number of mutations above a background rate threshold (those with a <= 1% probability of occurrence, according to the binomial cumulative distribution function, which takes into account both the gene length and the overall number of gene mutation)
- Group these positions to form clusters, joining positions that fall within distances of five or less amino-acid residues
- Complete these clusters by including the positions within or adjacent to each cluster that contains mutations in addition to those considered in the second step
- Compute a score for each cluster. This score is directly proportional to the percentage of mutations grouped within the cluster and inversely proportional to it's length, as shown in the equation

$$ClusteringScore = \sum_{i} \frac{fractionMutations}{(\sqrt{2})^{distance}}$$

  where i represents protein positions within the cluster, fraction mutations is the percentage of mutations falling in that position (out of the total observed in the protein across samples) and dist is the number of amino acids spanning between i and the position of the cluster with the largest number of mutations, i.e. its peak.

The end result is a method aimed at identifying genomic alterations that acts as drivers for cancer cells and the method can be found at **http://bg.upf.edu/group/projects/oncodrive-clust.php**.

1.4.2 Mutational heterogeneity in cancer

By examining the whole-genome sequence from 126 tumour–normal pairs across ten tumour types, they found marked variation in mutation frequency across the genome, with differences exceeding fivefold; the profile of the genomic variation was similar across and within tumour types. It was also found that mutation rate varies widely across the genome and correlates with DNA replication time and expression level as shown in figure 1.5 [10].

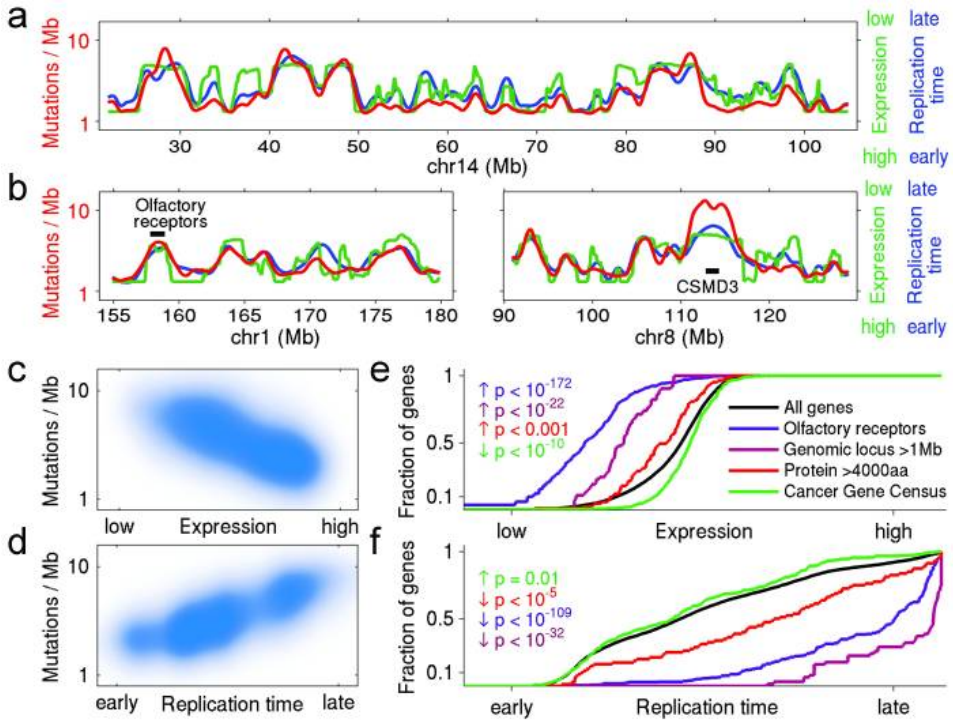


Figure 1.5: Mutation rate, replication time and expression level plotted across selected regions of the genome. Red shows total non-coding mutation rate calculated from whole-genome sequences of 126 samples (excluding exons). Blue shows replication time. Green shows average expression level across 91 cell lines in the Cancer Cell Line Encyclopedia determined by RNA sequencing. (Figure from article "Mutational heterogeneity in cancer and the research for new cancer-associated genes" [10])

1.4.3 Cell-of-origin chromatin organization

In the article "Cell-of-origin chromatin organization shapes the mutational landscape of cancer", they investigated the distribution of mutations in multiple independent samples of diverse cancer types and compared them to cell type specific epigenomic features. It was shown that chromatin accessibility and modification, together with replication timing, explain up to 86% of the variance in mutation rates along cancer genomes. The best predictors of local somatic mutation density was epigenomic features derived from the most likely cell type of origin of the corresponding malignancy. They also found that cell of origin chromatin features are much stronger determinants of cancer mutation profiles than chromatin features of matched cancer cell lines. Furthermore, it was shown that the cell type of origin of a cancer can be accurately determined based on the distribution of mutations along its genome. Thus, the DNA sequence of a cancer genome encompasses a wealth of information about the identity and epigenomic features of its cell of origin.

Epigenomics features indicative of active chromatin and transcription were associated with low mutation density, where repressive chromatin features were associated with regions of high mutation density. (Low mutation rate at open chromatin and high mutation rate at closed chromatin.)

The results suggest that the cell of origin of the individual tumour sample could be predicted from its mutation pattern alone [11].

Using mutation patterns is one path towards being able to identify the cell of origin for cancer, but in this thesis I have chosen another approach by using more traditional clustering with different feature creation.

Part II

The project

1.5 Methodology

The objective of this thesis is to explore how to interoperate information about functions and states of the DNA and cluster somatic mutations from cancer patients using this information. This has been done by building on existing methods for clustering and feature creation while expanding their capabilities within the scope of the problem. More specificity by exploring different distance functions, similarity between somatic mutations/donors that is not necessarily constrained by transitivity/the triangle inequality and feature creation. Also to allow for these properties during clustering a version of the hierarchical clustering algorithm has been implemented.

Chapter 2

Distance and similarity

2.1 Similarity

Usually in machine learning, the similarity between objects is the inverse function of distance. The smaller the distance between objects, or points in n-dimensional vector space, the greater the similarity. In clustering this is done internally when the machine learning algorithm tries to minimize the within cluster distance. A typical similarity function would be something like
$s_{ij} = 1 - \frac{1}{d_{ij}+1}$.

This works very well in general as distance between objects when using feature vectors means that objects close together in the vector space, have similar values for the same features. But dependent on how the features are created, and what they represent, the relation between distance and similarity isn't always this straight forward.

2.1.1 Object representation

Positions

Representing objects along the human genome could be done by positions. Using one position inside the whole genome is not advised, as in theory, the distance between the end of chromosome 1 and the start of chromosome 2 is not define, since DNA sequences are not continuous, but separated by chromosomes. So the points should be separated by chromosomes. But x and y coordinates is also not ideal, in the case of y for the chromosome, 1 to 24 (including the X and Y for males or X and X for females in healthy human beings, as humans has 23 pairs of chromosomes, but adding up to 24 unique chromosome pairs) and x for the position within the chromosome itself, since the distance between chromosomes is not defined.

In practice, it is possible to represent objects with one value for within chromosome positions and one value for chromosome. But then the distance measure has to account for the 24 separate spaces and always returning a maximum values when comparing positions within separate chromosomes. Another possible solution which allows for traditional

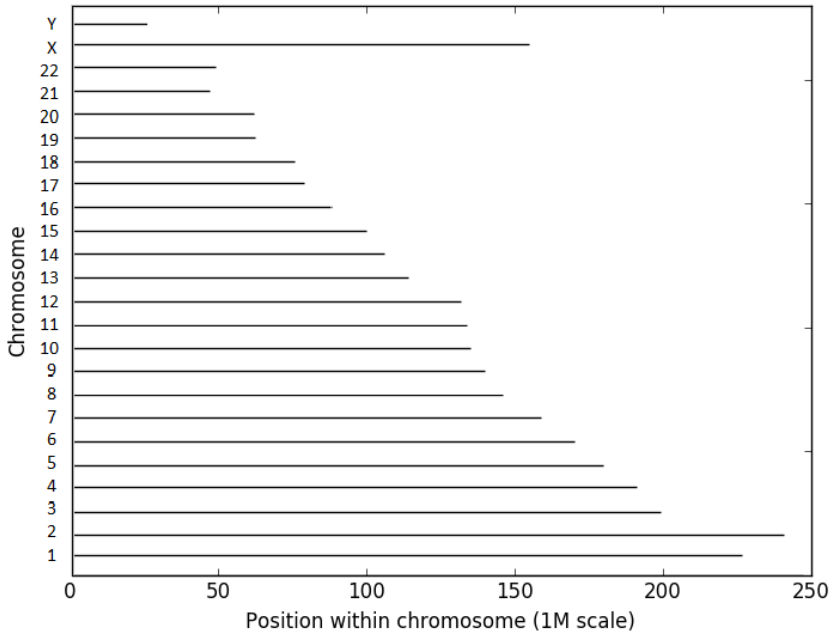


Figure 2.1: Possible position within the humane genome

	Chr. value range	Lower most dissimilar	Upper most dissimilar
Chr. 1	$[0, 0.25]$	$[0, 0, 0, 0, 0]$	$[0.25, 0.25, 0.25, 0.25, 0.25]$
Chr. 2	$[0.51, 0.76]$	$[0.51, 0.51, 0.51, 0.51, 0.51]$	$[0.76, 0.76, 0.76, 0.76, 0.76]$

Distance between most dissimilar vectors in chr. 1 and chr. 2 = **0.559**
Distance between upper limit vector in chr. 1 and lower limit vector in chr. 2 = **0.581**

Figure 2.2: Illustration of feature interval concept

distance measures is to separate the within chromosome position into different spaces, using only one value.

If the features were separated into intervals, one for each chromosome, where the lower limit of the first interval was 0 and the upper limit was for example 0.25, then the next interval could have lower limit of anything greater than 0.5 and maximum value of lower limit + 0.25. The actual lower and upper limit of each chromosome interval isn't that important, as long as the gap between intervals is always larger then the intervals themselves. This way the objects within different chromosomes would always lay closer together in the vector space, shown by the worst case in figure 2.2

Even if it works, it is highly unpractical. The theoretical resolution of the features is greatly reduced since it is divided by number of intervals, and over half the potential resolution is unused to make room for spacing between chromosomes. If the perfect one to one resolution should be preserved, the amount of disk space and memory usage would be high by holding floating point numbers with very high precision. Since the entire genome has approximately 3.1 billion base pairs, and after adding

the spacing, 6.2 billion unique values needs to be stored.

Overlaps

Another method of representing objects is to use relevant information about segments of DNA. Some sequences have known properties and relate to different kind of functions in areas of the body. Combined with the assumption that what lies close together along the genome are related, these segments can be used to enrich the representation of the objects. To integrate this enrichment, a form of relation to the segments needs to be established.

The most basic relation between tracks of positions along the genome is overlaps. For each object, if it consists of one or several positions, it either overlaps a track of segments or it does not. The dimension of the n-dimensional vector space/feature vector will be the number of segment tracks used to form the relation.

A conceivable problem with this representation is the limited number of unique representations an object will have. With an upper boundary of the faculty of the number of segment tracks, since the feature vectors has binary values and not floating point values as is more common. Which seems a lot, but in practice might not be this high due to the data sets being of a biological nature and will not evenly distribute along the genome.

Distances

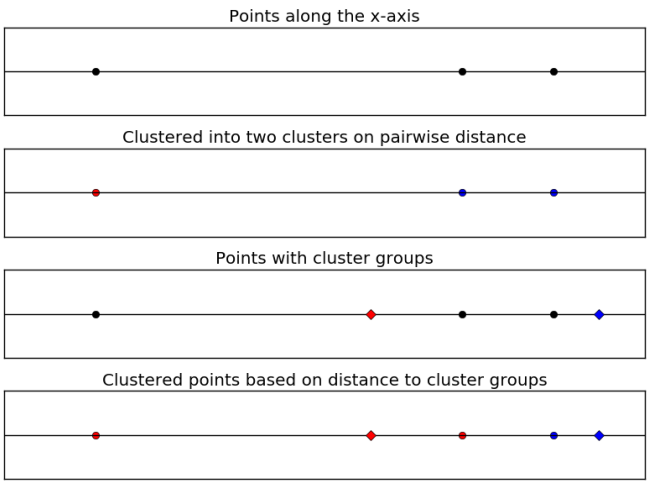


Figure 2.3: The alteration of clustering with added information

Distance is another obvious choice when forming a relation between tracks, again with the assumption that what lies close together are related. Now, each point is no longer just a point, but a list of distances to segments in the same chromosome the point is located, or a feature vector where each feature is a distance to some segment or point. This way each object got information about possibly related sequences of DNA with known properties, as well as the position within the chromosome. The theoretical clustering of objects might be altered by this representation, as shown in 2.3.

Note that the the points in the last two sub plots with cluster groups 2.3 does not have features with positions, but rather distances to the cluster groups. So clustering on this would not take into account where the points are in relation to the start of the chromosome, but where the points are in relation to the cluster groups.

Comparing with distances from the start of the chromosome 2.1.1, this representation has some advantages. The resolution of the features is not limited by dividing the total resolution up into segments and added padding. Also the total distances to relevant information about segments of DNA will generally be smaller than the distance to the start of the chromosome, as we shall see later.

DNase and chromatin accessibility

When areas of DNA are active, it is hypersensitive to DNase 1 (deoxyribonuclease 1), an enzyme that cleaves DNA, called DNase 1 hypersensitive sites. These accessible parts of DNA are more prune to mutations, as shown in [12] where "...the majority of S1-hypersensitive sites detected were not randomly distributed over the genome but apparently were clustered in damage-sensitive regions."

Since the goal is to look into somatic mutations, which is a result of damaged DNA, data sets on DNase I hypersensitive sites and accessibility of DNA are interesting.

Annotated sequences as centroids

In k-means [13] like clustering the number of clusters is predefined and the initial cluster centres are more or less randomly chosen. In contrast to hierarchical clustering k-clustering has the problem of being sensitive to initial conditions. The number of classes can lead to over- or under fitting and the initial cluster centres might not be close to the correct/desired clustering.

Using the positions of annotated sequences of DNA as the initial clustering centres, a standard k-means clustering algorithm will work just fine to find groupings around the annotated sequences. But the number of clusters is predefined by the number of groups of annotated sequences, and the goal is to also find sub clusters within the initial groups, so standard k-clustering is not an ideal option.

Multi point objects

In the case of somatic mutations leading to cancer, the patient does not only have one mutation in their DNA, but ranging up to hundreds of thousands. SSM-tracks from the ENCODE project report several mutations per donor, and since the original mutations is not know (which is part of the problem this thesis is working towards), all mutations per donor is treated as equals. Either way, several mutation points per donor gives an even more precise

description of a donor/cancer patient and should be used together when the information is available.

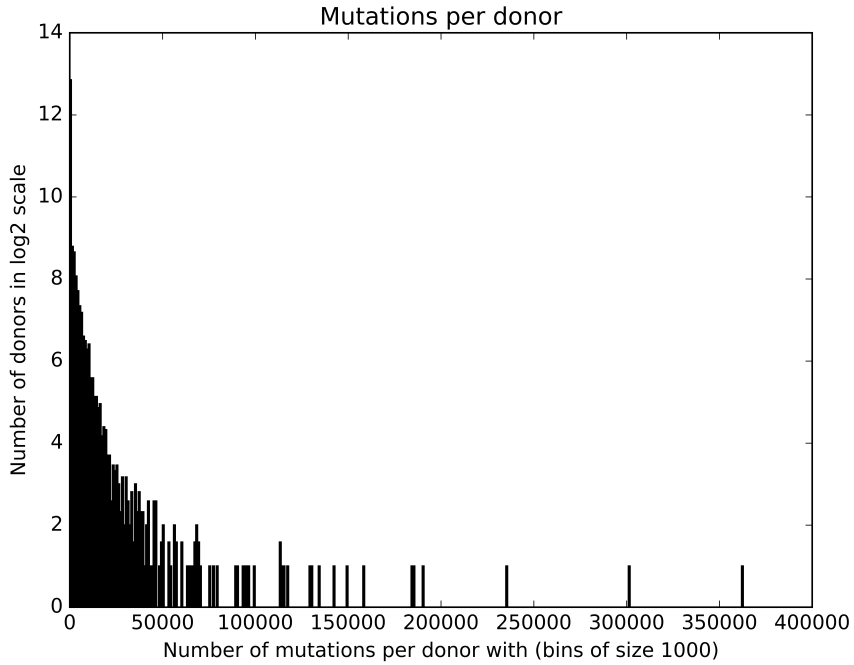


Figure 2.4: The distribution of donors by numbers of simple somatic mutations, compiled from the ICGC data portal.

Binned features

A challenge with multi point objects is how to create feature vectors for each object. Using the same method as with relevant segments of DNA, each feature vector would instead be a matrix, "distance to relevant segments of DNA"-feature vectors for each point.

A common way to look at similarity between tracks is to use bins, usually of a decent size like 1M (which result in a little over 3080 bins over the entire genome) or 10M. This way it is faster and easier to compare tracks, and tracks with very different number of points or segments can easily be compared.

Using bins, it is possible to calculate how much a positional track (donor-track) overlaps with a segment track within each bin. This still gives a feature matrix of dimensions (3080,#tracks). But since the goal also is to use how a positional track relates to segment-tracks, and the segment-tracks are annotated with tissue type, the feature vectors related to the same types of segment-tracks can be grouped together, and the resulting feature matrix has dimensions (3080,#types), which is easier to handle later on when calculating distances and similarity.

j / i	Presence	Absence	Sum
Presence	$a = i \bullet j$	$b = \bar{i} \bullet j$	$a + b$
Absence	$c = i \bullet \bar{j}$	$d = \bar{i} \bullet \bar{j}$	$c + d$
Sum	$a + c$	$b + d$	$n = a + b + c + d$

Figure 2.5: Expression of binary instances i and j

Bin presence similarity

Bins can also be used to represent an objects position within the genome as a boolean presence array for each bin, where true values means that the object has a point within this bin.

It is also possible to compare objects using bin presence with the Forbes formula (variables are explained in 2.5):

$$S_{FORBES} = \frac{n * a}{(a + b) * (a + c)}$$

The result of Forbes gives a similarity score between the bin presences of the objects, to use it as a distance measure, it can simply be converted to distance with: $forbesdistance = \frac{1}{1+S_{FORBES}}$.

Feature imbalance

A danger when comparing similarity to segment-tracks, is the total size of the segments in the segment-track. If a segment-track has a very high total overlap of DNA, every point-track compared to that one would look as if it was very similar. And when comparing positional-tracks against each other every positional-track would be very similar in the feature derived from this segment-track with high coverage. This is not ideal, and would make this information irrelevant in a clustering that takes into account the similarity with cell types/tissue types.

A way to deal with this is to weight each feature, depending on the probability of an overlap between a point track and this segment-track. Generally it's enough to look at the probability of an randomly distributed set of point overlapping, so the problem boils down to using total segment length and total genome length when creating features.

Note that when using Forbes and correlation, weighting of features is not necessary, as the formulas takes into account the total overlap of DNA for both tracks.

2.1.2 Reference correlation

Normally when calculating the similarity between segment tracks, overlaps and bin sizes are calculated. This gives the same values as used when evaluating the performance of classifiers (TP, TN, FP, FN) 1.3.5. True positive is the sum of the segment tracks overlapping, true negative is the sum of positions where neither tracks has any segments/values. False positive is where the second track has segments, but the first tracks does not, and

false negative is where the first track has segments, but the second track hasn't.

These values are useful not only when comparing segment tracks, but also in our case when comparing point tracks to segment tracks. Converting the positional tracks to segment tracks where all the segments have length one, the same function can be used.

The resulting values is interesting in itself, but in a clustering/feature context values should generally be a single float. Fortunately correlation coefficients can be calculated using this formula [14]:

$$coeff = \frac{tp * tn - fn * fp}{\sqrt{(tp + fn) * (tn + fn) * (tp + fp) * (tn + fn)}}$$

2.1.3 Correlation coefficients as features

Correlation coefficients is in the range [-1,1] where 1 means perfect correlation between the tracks, ie. the tracks perfectly overlap, and -1 means that they do not overlap at all but span the entire bin/chromosome.

Note that this is not normal correlation coefficients between objects being clustered. They are correlation between an object being clustered and a reference data set. If they were simply the correlation between objects, the distance would be a function of that number, but when both object reference a third track, there are two numbers and the two correlation coefficients needs to be combined in a sensible way.

When using correlation coefficients as features to calculate distance between objects it is important that the following conditions are met:

- Two positive coeffs. have a low distance
- Two negative coeffs. have a low distance
- Two zero coeffs. have a medium distance
- A negative and a positive coeff. have a high distance

It is also preferred that the resulting distance is positive and for simplicity in the range [0,1] as a floating point number. The maximum value of the product of two correlation coefficients is 1 and the minimum value is -1. So subtracting the product from 1 and dividing it by 2 gives a result in the preferred range and also fulfilling the conditions laid out over.

It turns out that most of the correlation coefficients generated in this thesis lay very close to 0, which is not surprising as each chromosome is large, and the data is in relation sparse. To evade the resulting floating point precision problem by taking the product of two very small numbers (ie. numbers very close to 0), the entire matrix of correlation coefficients can be scaled up so that the product of the two largest, or two lowest, numbers is less or equal to 1. This can be done by adding the two smallest and two lowest numbers together, dividing them by two and then dividing the entire array with the absolute value of the two.

The python code for this is:

```
# with cc being np array of shape (n,m), dtype float
pos = abs(np.sort(cc.flatten())[-1] \
      + np.sort(cc.flatten())[-2]) / 2
neg = abs(np.sort(cc.flatten())[0] \
      + np.sort(cc.flatten())[1]) / 2
cc = cc / max(pos, neg)
```

Norm

Calculating distances between vectors can be seen as doing it in two steps: First subtract one vector from another, then find the length of the resulting vector. Commonly used distance measures differ in the way they calculate the length of a vector. In mathematics a norm is a function for assigning a real positive length or size to a vector. Norms have three axioms.

- Zero vector: If the function maps a vector to 0, the vector is the zero vector
- Triangle inequality: The length of u + v is always less or equal to the length of u + the length of v
- Absolute homogeneity: If the vector is multiplied by a factor, the length of the vector is multiplied by a fraction of that factor

Euclidean distance is calculated by the Pythagorean formula

$$d(u,v) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2} = \sqrt{(u-v)\cdot(u-v)}$$

where u and v are vectors $u = (u_1, u_2, ..., u_n)$ and $v = (v_1, v_2, ..., v_n)$ and Manhattan distance is given by the sum of the differences of their corresponding components.

$$d(u,v) = \sum_{i=1}^{n}|u_i - v_i|$$

Manhattan distance is using $norm_1$ and Euclidean distance is using $norm_2$. There exists an infinite number of norms but $norm_1$ and $norm_2$ is most commonly used.

2.2 Distance matrix

A distance matrix is a collection of the pairwise distance between all observations. Given distance matrix dm, dm[i,j] is the distance between object i and object j. Some times the distance matrix is formatted as a vector to save space and computation time as a squared matrix contains duplicated information, ie. dm[i,j] = dm[j,i]. The order of the vector is usually (0,1), (0,2), ..., (0,m), (1,2), ..., (1,m), ..., (m-1,m).

In general the distance between i and j in the condensed distance matrix found at index

$$\binom{m}{2} - \binom{m-i}{2} + (j - i - 1)$$

or

$$m * i - \frac{i * (i+1)}{2} + (j - i - 1)$$

where $i < j$ and m is the number of columns along the squared distance matrix.

Computing a distance matrix is a computationally intensive task and optimizations is often used. This can include taking advantage of the triangle inequality, so that when the distance between vector u and v and v and w is known, but not the distance between vector u and w, the triangle inequality says that the distance between u and w has to be less or equal to the distance between u and v + the distance between v and w. Also, implementing vectorization can speed up the computation of pairwise distance.

2.3 Method

2.3.1 Distance

As described in subsection 2.1.3, normal distance measures always gives a distance of zero when measuring the distance between equal vectors. This makes perfectly sense as the two points are at the exact same place in the vector space, but it implies that equal vectors are always perfectly similar and thus there is no natural clustering order between the objects represented by these features.

Figure 2.6 shows three basic cases of vectors with 1s and 0s and their distances using Euclidean- and Manhattan distance. The vectors u and v are illustrations of feature vectors resulting from points overlapping segments. The distance between two vectors with only 0s and the distance between two vectors with only 1s are identical, thus implying they are equally similar. But sharing the property of not overlapping does not necessarily have the same indication of similarity as the property of overlapping. The standard distance measures does not take into account the underlying properties of the feature vectors.

u	v	Euclidean distance	Manhattan distance
[0,0,0,0,0]	[0,0,0,0,0]	0	0
[0,0,0,0,0]	[1,1,1,1,1]	2.236068	5
[1,1,1,1,1]	[1,1,1,1,1]	0	0

Figure 2.6: Distances between vectors with 1s and 0s

The assumption that objects that has a property is more similar than objects that does not, gives the basic similarity ranking:

- Equal vectors with only 1s should always have the distance of zero.
- Equal vectors with only 0s should have a greater distance than equal vectors with only 1s.
- The distance between a vector with only 0s and a vector with only 1s should have the greatest distance.

Weighted threshold

"Weighted threshold" is a distance measure with these three assumptions in mind, introducing weight and threshold to differentiate between vectors that normally would get a distance of zero but in this context are not equally similar. The threshold variable is not necessary when dealing with vectors only containing 0s and 1s, but to generalize the function the variable is introduced.

Figure 2.7 shows a simplified example with vectors of range one in the range from 0 to 1, where the resulting distances are plotted in a wire frame plot. With threshold set to 0.5 and weight set to 0.25.

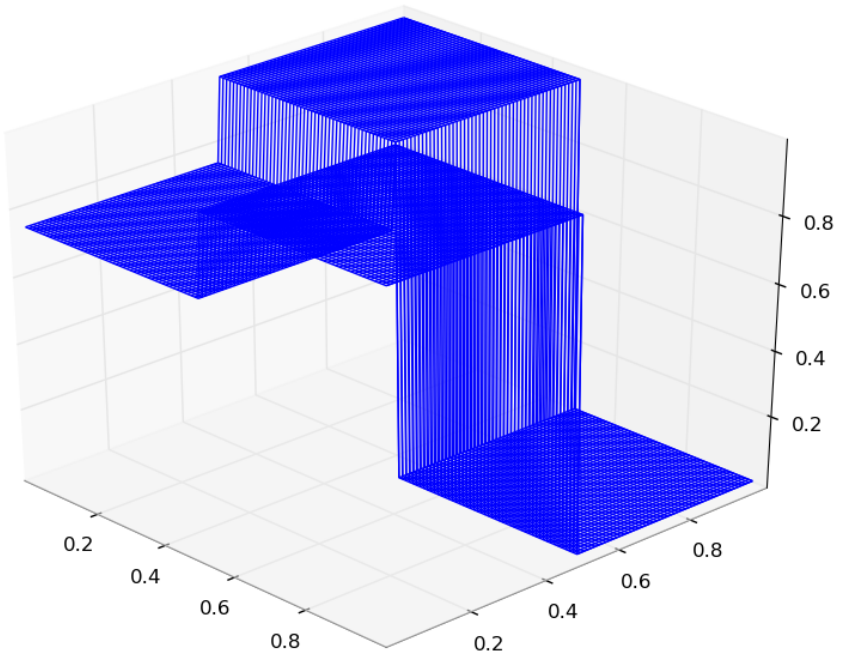


Figure 2.7: Weighted threshold plot with one valued vectors. Horizontal axes are input values and vertical axis is distance.

This plot shows three different resulting distances with 1-length vectors:

$$dist(u,v) = \begin{cases} 0 & \text{if } u \text{ and } v > \text{ threshold} \\ 1 - weight & \text{if } u \text{ and } v <= threshold \\ 1 & \text{if } u \text{ or } v > \text{ threshold, but not both} \end{cases}$$

More generally the distance between two vectors u and v, using weighted threshold, is

$$\frac{||u|| - \sum_{i=1}^{n} (u_i > t \wedge v_i > t) - \sum_{i=1}^{n} (u_i <= t \wedge v_i <= t) * w)}{||u||}$$

Where t is the threshold and w is the weight. Note that the resulting distance will always be in the interval [0,1] no matter the values in the vectors and how many dimensions the vector represent. This is an important factor when comparing vectors with different number of dimensions.

Weighted threshold deals with the basic problems of standard distance measures, but an apparent problem is the limited resolution of the distances produced. As the dimensions rise, the resolution will get higher, but not theoretically infinite as it counts occurrences of vector index pairs over and under a threshold.

Relevant distance

With weighted threshold being a variant of Manhattan distance, modified to capture the underlying properties of the feature vectors, "Relevant distance" uses parts from Euclidean space to get a theoretical infinite resolution. "Relevant distance" does not use a threshold, but instead three parts that insure the three properties.

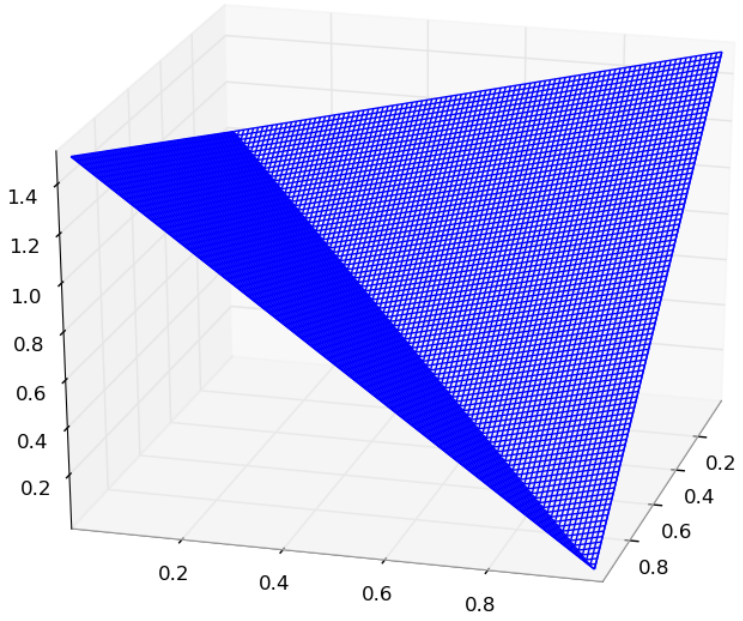


Figure 2.8: Relevant distance plot with one valued vectors. Horizontal axes are input values and vertical axis is distance.

- The first part is $\sum_{i=1}^{n} 1 - (u_i + v_i)/2$ where 1 - the average of the values are added together. This gives the inverse magnitude of the vector in Euclidean vector space scaled to a maximum value of 1, ie. higher values gives a lower result.
- The second part is $\sum_{i=1}^{n} |u_i - v_i|$, the Manhattan distance between the vectors.
- The third part is dividing the sum of the first and second result by the length of the vector. Again to be able to compare distances where one pair of vectors have n dimensions, another pair of vectors have m dimensions and $n \neq m$.

The complete equation looks like this

$$\frac{(\sum_{i=1}^{n} 1 - (u_i + v_i)/2) + (\sum_{i=1}^{n} |u_i - v_i|)}{||u||}$$

Plotting the result in a wire frame plot with the same inputs as 2.7. This plot 2.8 is produced. Looking at the two plots it is clear that the same ideas applies, but the "relevant distance"-plot has the ability to differentiate vectors much better than the "weighted threshold"-function. It also scales the distances to a maximum value of 1.5. The maximum value should not have an impact on clustering but is only a result constructing the functions in different ways.

2.4 Category

2.4.1 Feature importance

In machine learning, feature selection can be an important step to reduce training time, reduce over fitting, increase generalization and in general make the model less complex for further analysis. If there is some sort of labelled data available, one way is to train a model with all the features and get the importance of the features used. Methods such as Random Forest classifier and Logistic Regression in Python can calculate the feature importance of features used in training and will generally give a good indication the ability of features to predict different classes.

My assumption with the aggregated dataset of somatic mutation points from ICGC data portal[15], combined with data sets from ENCODE[16] and RoadMap[15] with information about chromatin accessibility and DNaseHS sites, is that some features will have a higher predictive properties when classifying certain classes. Since each feature is created in such a way that it relates to one of the data sets from ENCODE or RoadMap.

The plot 2.9 shows the feature importances of each group of features in a classification of two classes (for example colon and not colon, lung and not lung an so on) with 4302 mutation points. The classification was done by a Random Tree Classifier with previous results showing an accuracy

of over 90% in all the classifications. The 204 features are along the x-axis, grouped on related type of tissue into 30 groups. The y-axis shows the relative importances of the different groups when classifying different tissue types. The blue line shows standard deviation between the features within each group.

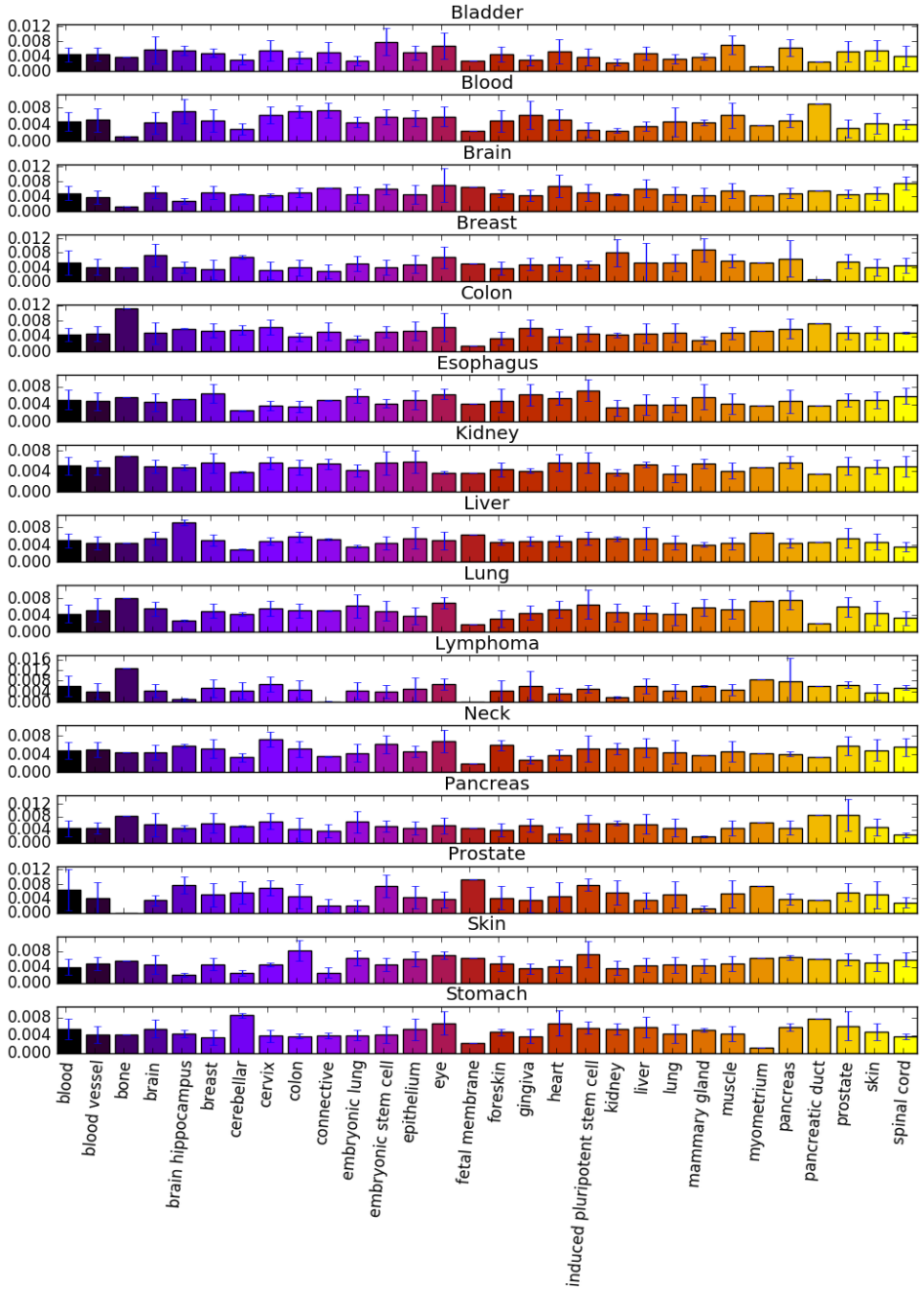


Figure 2.9

These results are not perfect, in the sense that each class does not have overwhelming important features of groups which in some way are related

1. wgEncodeUwDnaseAg10803PkRep1
1. wgEncodeUwDnaseAg09309PkRep1
2. wgEncodeUwDnaseGm06990PkRep1
2. wgEncodeUwDnaseGm12865PkRep1

Figure 2.10: Reference set example

to the tissue type of the class. But at the very least it illustrates that when predicting different classes, the same features with the same values are used in different ways by the Random Forest Classifier, ie. having different importance numbers.

Normally, all features are equal in the sense that they all have values in different dimensions in the vector space, and the machine learning algorithm know of no relation between them. There may of cause be a relation between features or the data used to create them, as indicated by figure 2.9, but this is lost when feature vectors for each object is created. The idea of categories is to preserve some sort of relation between the features, and for machine learning algorithms to be able to use this during clustering/classification.

Preferable there should not be added complicated structures to represent relation. One way to accomplish this is to divide the vector space according to the grouping of features.

This gives non overlapping vector spaces that are subsets of the vector space (a partition of the vector space) the feature vectors normally would be placed in, with normal distance measures and clustering. For convenience, call each of these subsets a category.

But "a category is a vector space" is just one way of defining a category. In context of the actual clustering algorithm, it makes sense to look at a category as one of several distance matrices, since the algorithm is limited to a distance matrix (or several), not the data set. When creating feature vectors on the other hand, it makes sense to think of a category as a subset of the reference set or a subset of the features for an object. The idea of separate vector spaces for each category only comes into play when creating the distance matrices. Figure 2.10 shows a small example of a the file names in a reference set. The first two and the last two would be grouped together as the first two (ag10803 and ag09309) refers to skin and the last two (gm06990 and gm12865) refers to blood. This would result in two categories, with two features for each category.

For the clustering algorithm implemented in this thesis to potentially act different than ordinary hierarchical clustering, there has to be at least two categories. In the case of one category, only one distance matrix will be produced and the concepts of category, and other related concepts, will not be relevant.

2.5 Locking

When partitioning the vector space, the problem of how to handle several distance matrices during clustering arises. If the algorithm was free to choose the lowest distance from any of the matrices for each iteration, a form of feature importance would be generated by storing what category/distance matrix was used to form that cluster. But choosing freely between them all might not be desirable in all cases. Again with the results shown in figure 2.9, some sets of features have a higher predictive property when clustering or classifying certain mutation types. Using this knowledge, restricting the algorithm to only use a subset of the distance matrices, with maximum size, might be a good option.

This idea, which I call locking, works by trying to minimize the distance between pairs of objects, but using up to k distances from all of the available distance matrices.

2.5.1 Locking to one category

In the first case where a newly formed cluster only gets locket to one category, there are two requirements:

- Find the lowest value in the extended distance matrix to form a new cluster.
- Restrict the algorithm from choosing to cluster two objects the has used different categories.

2.5.2 Locking to several categories

When picking a locking number higher than 1, there are several choices presented that result in widely different behaviour. To choose locking numbers, understanding these effects are crucial.

Choosing the two closest objects

First is how to decide which two objects to choose. Setting the locking number to k means that there are k numbers of possible distances between every pair of objects. For the best resulting clustering, the ideal choice might be to always choose the two objects with the k categories which result in the lowest sum of distances between them, using all of the distances/categories. This gets highly impractical in practice, resulting in a complexity of O(m*k!*k) for finding the lowest distance, where m is the number of objects and k is the number of categories to be used.

Another more simple possibility is to first choose the two objects simply by the lowest distance in the distance matrix, as in the case where n is 1. Then the rest of the categories can be chosen by finding the categories where the distances between the two objects are the smallest. This way of doing it will not take advantage of all the information available, but the complexity is polynomial, not exponential.

Assigning categories

The second challenge is how to to deal with clustering of objects where categories partially overlap. Figure 2.11 shows three objects where 1 and 2 has overlap, and 2 and 3 has overlap, but 1 and 3 has no overlapping categories.

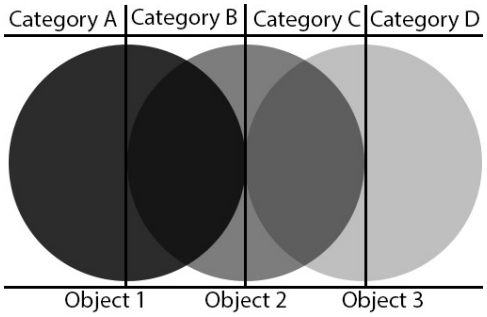


Figure 2.11

If no partial overlap was allowed, the clustering would stop very soon when using larger locking numbers. In some cases this is fine, but in general the premature stopping of a clustering is not desirable, and a cutting of the tree afterwards will yield more useful information. So the decision is between intersecting the set of categories from each cluster, or to join them together. With the objects from figure 2.11 the there are two options for the initial clustering. Either object 1 and 2 or 2 and 3 are clusters together, as they are the pairs that share a category. The different results show up when using either intersect or join to merge the categories. In the case of intersect, the clustering is finished in one step, as the two resulting clusters has no category in common. But in the case of joining, all the initial objects gets clustered together, even if there were objects with no overlapping category, as illustrated in figure 2.12.

2.6 Classification

Deciding labels for somatic mutations in an aggregated data file, using only positional information, could be seen as a simple classification problem. What prohibits this is that if the goal is to find the cell or origin for each somatic mutation, group of somatic mutations or donor, the cell of origin has to be known for the training to be possible. This is of cause not known, but for the sake of testing the abilities of a classifier with the limited information, artificially deciding the type of cancer for each somatic mutation, or group, a classifier could be used. Then the same data sets used in clustering can construct a standard feature vector for each of the mutation points, as true labels in terms of cancer type (not cell of origin) can be fetched using RESTful endpoints through the ICGC API.

2.6.1 MLP

A MLP or multi layered perceptron is a classifying algorithm that gets it's inspiration from how the brain works, with interconnected neurons and correction mechanisms. The end result would be inputs getting fed into layers of neurons which, dependent on weights, either fires and sends a signal to the next layer, or don't. The result is the pattern of the last layer.

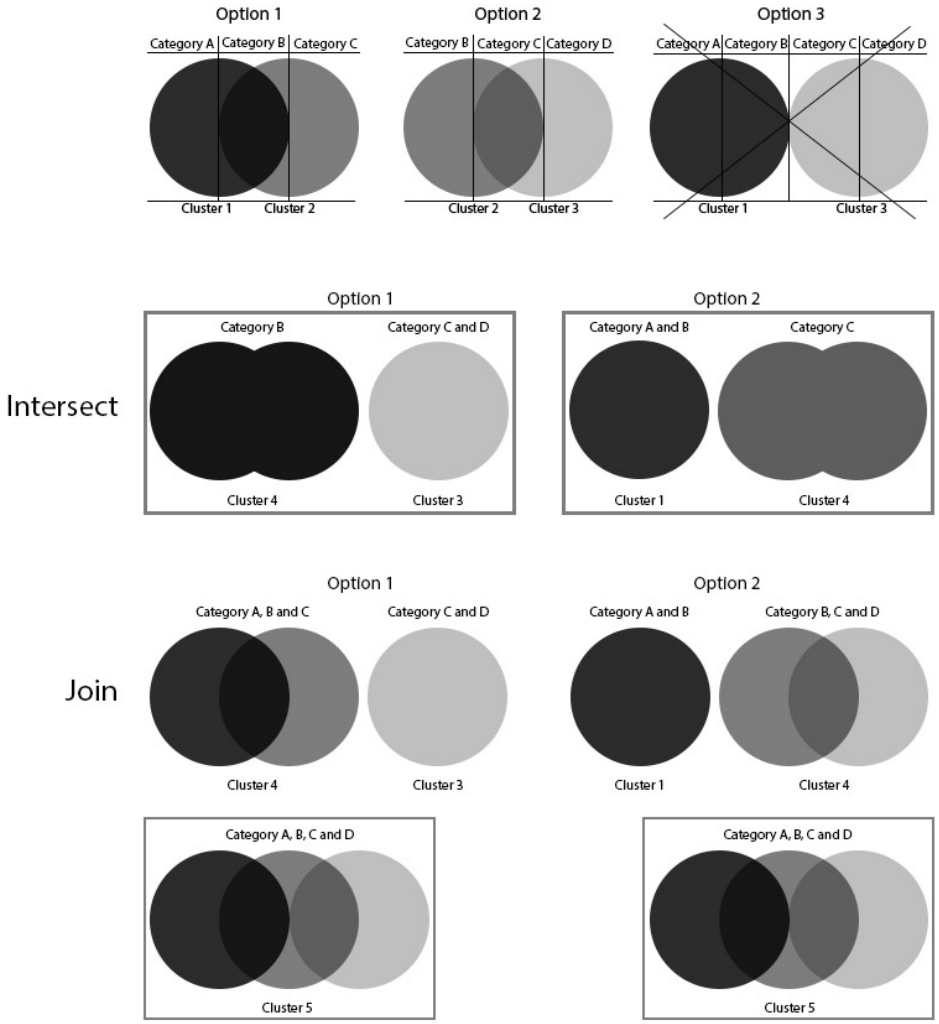


Figure 2.12: Different results when clustering objects with partial overlapping categories. The possible final results is marked by the grey squares.

In a MLP, the neurons are just logistics functions with weights that mimic this boolean behaviour Initially the weights of the functions are random numbers close to zero, so the weights can be updated and doesn't get trapped in local optima, too early. The MLP used is a standard batch variant, with input nodes equal to the number of features, one hidden layers and output nodes equal to the number of possible classes.

Balancing data

Since the aggregated data set of somatic mutations have classes of different sizes, the smallest classes are left out to make the learning easier on the MLP. The smallest classes are defined as having less than the median of the number of members in all the classes multiplied by 0.75. Then the largest classes are down sized so that every class has the same number of members. All the objects are also shuffled to help with proper training and the values are normalized to lay in the interval [-1,1] which is preferred for training an MLP with a standard logistic function.

Results

Several tests was ran with different data sets for constructing features and 5, 8, 12, 15, 20 and 29 nodes in the hidden layer. The MLP was trained with 10-fold cross validation using the change in residual sum of squares (measure of error) as the stopping criteria. A held out part of the data set is used for testing to create the confusion matrix. The test runs can be found at the HyperBrowser[17].

The feature data set that trained an MLP which yielded the highest percent correctly labelled classes with around 20% seemed to be the data set named 'dnase'. It contains files with segments related to DNase hypersensitive sites from RoadMap and ENCODE, that are regions of chromatin more sensitive to cleavage/cutting by DNase enzyme. The number of hidden nodes does not seem to affect the performance much, as the lowest percentage correctly classified was 12 hidden nodes with 19.14% and the highest was 20 hidden nodes with 20.17%. The difference in performance is small enough to be explained by the stochastic nature of the machine learning algorithm with small random negative and positive weights in initialization.

24.95726496	21.88034188	22.05128205	21.36752137	20.85470085
20.17094017	19.82905983	20.17094017	21.02564103	19.82905983

Figure 2.13: Correct percentages for each fold

Standard deviation: 1.46% Average: 21.21% Correct labeled classes: 19.829060%

7	3	4	4	1	3	4	1	2	1
6	**16**	3	4	2	9	10	4	6	7
5	3	**7**	6	6	6	4	12	7	8
8	1	9	**8**	4	4	10	3	5	10
6	7	5	6	**26**	9	4	4	4	12
4	5	1	5	6	**10**	2	1	8	6
11	1	0	6	2	1	**9**	2	1	3
7	10	12	5	5	9	6	**18**	12	4
6	2	0	6	6	5	1	0	**5**	2
5	3	12	8	3	8	8	7	10	**10**

Figure 2.14: The confusion matrix from the last fold using 10 classes, 8 hidden nodes, 1755 objects for training and 585 objects for testing. Arguments to the algorithm were: beta=1.0, eta=0.1, momentum=0.9, iterations=100

Deep learning

It is worth mentioning that deep learning algorithms, which as become increasingly more popular as hardware is able to cope with more complex algorithms, could be used for this kind of classification. Simply put deep learning is a variant of an MLP with many more hidden layers and more input nodes. This way more information per object can be fed into the algorithm, and the added layers might catch patterns or ways to distinguish objects in a way that a shallow MLP won't. A drawback with added complexity is the amount of time it takes to train models, which can be months for complex models.

For hand writing recognition, speech recognition, face recognition, object identifications in still pictures and video, iris identification, to mention some, deep learning generated models have been shown to provide good performance in classifying objects [18].

Chapter 3

Category clustering

3.1 Implementation

The more advanced and untraditional topics discussed in chapter 2 is inconvenient or impossible to use with standard libraries. So functions for clustering, data manipulation, feature creation and distance measure has all been implemented and are openly available as a Github repository at https://github.com/flyvekristoffer/Category-Clustering.git. The code base can be viewed online or cloned using "git clone".

The entire project is implemented using Python 3.5 and uses packages from the SciPy Stack. To run the code it is advised to install the SciPy stack. Alternatively Numpy, Matplotlib, Scipy and sklearn can be installed independently. Note that to get from data sets to linkage matrix without any of the other functionality, just numpy outside standard Python language is used, but other packages are imported and the code has to be manually tweaked if other packages is not present.

The code base is highly modular, to easily mix and match functions and functionality. This section will be mostly about the main parts of the code.

3.1.1 Distance

distance.py is a collection of methods for calculating distance between vectors. About half of the functions are from the Scipy package [19], with some removed and some custom functions added.

pdist is the function for calculating the pairwise distance between all vectors given as an argument to the function. It will return the distances as condensed matrix with length n * (n-1) / 2 where n is the number of objects in the array containing vectors. If the distance measure is given as a sting, and the string is in the collection of C-implemented functions, the C-version of the pdist function is used.

Euclidean, squared euclidean, correlation and city block are all implemented in C while weighted threshold, relevant distance and Forbes-Correlation (and a few more) are implemented in python. Using the C-implemented distance functions will give a significant speed up.

For pdist with Forbes-Correlation and categories,

$forbes\_corr\_pdist(presence, corr\_coeff, k)$ is the best option as it is vectorized across categories and will be significantly faster than individual calls on $forbes\_corr\_dist(u\_pres, u\_cor, v\_pres, v\_cor, k)$.

3.1.2 Feature methods

feature.py does all the calculations necessary to create feature vectors and reading tracks. The three main types of features supported are overlaps, inverse distance and parts used in Forbes-Correlation, discussed in subsection 2.1.1, with a variety of arguments available to change their behaviour. It also contains a functions for manipulating tracks, plotting results and evaluating a clustering.

3.1.3 Clustering algorithm

The clustering algorithm used is a bottom up hierarchical variant with some significant variations, mainly the ability to cluster objects across distance matrices and locking to them in further progression.

Representation

The algorithm uses a tree structure to represent the clustering and to keep track of which objects are a result of clustering with which category. This way information about category counts in a cluster is easily available. Normally a simple linkage matrix would be enough, but a tree structure helps visualize and keep track of potentially important information. This representation can also be used outside clustering as it is returned by the algorithms. It contains methods to extract information about ids, categories and is able to cut the tree. When only the linkage matrix is present, a tree can be built by using static function $Z\_to\_tree\_dict(Z, cats)$ from class Tree in CatClust.py. Z is here the linkage matrix and cats is a list of lists where each sub list contain integer(s) referencing to categories used to form the cluster at that step. Ie. cats[i] is categories used to form Z[i].

Linkage

After each iteration in agglomerative hierarchical clustering, linkage has to be calculated. Linkage is a way to calculate the distances from a newly formed cluster to all other clusters, combining distances available in the distance matrix. Centroid- and medium variance linkage also uses the size of the clusters to calculating new distances.

In Algorithms for clustering data [20], Jain and Dubes shows a modified version of the general formula first proposed by Lance and William in 1967, to express SAHN (Sequential, Agglomerative, Hierarchical, Nonoverlapping) clustering methods. The distance between newly formed clusters (i,j) and existing cluster k is given as

$$d((i,j),k) = \alpha_i d(i,k) + \alpha_j d(j,k) + \beta d(i,j) + \gamma |d(i,k) - d(j,k)|$$

were d(i,j) is the distance from i to j and $\alpha_i, \alpha_j, \beta$ and γ are constants or size fractions for each linkage method. The general formula gives six formulas for linkage supported by my clustering implementation.

The most basic linkage methods is single linkage. When calculating the distance from the newly formed cluster (i,j) to any cluster k, the distance from (i,j) to k is the shortest distance from either i or j to k. This traditionally results in long chains of clusters, where bigger clusters gets combined with single objects clusters:

$$d(i+j,k) = \frac{d(i,k)}{2} + \frac{d(j,k)}{2} - |\frac{d(i,k)}{2} - \frac{d(j,k)}{2}|$$

Complete linkage is similar to single linkage but instead of using the shortest distance from (i,j) to k, it uses the greatest distance. In contrast to single linkage, complete linkage usually results in a more evenly distributed tree:

$$d(i+j,k) = \frac{d(i,k)}{2} + \frac{d(j,k)}{2} + |\frac{d(i,k)}{2} - \frac{d(j,k)}{2}|$$

Average linkage takes the average of the distance from i to k and j to k. For it to work correctly when combining clusters of different size, the size difference is taken into account:

$$d(i+j,k) = \frac{|i|}{|i|+|j|} * d(i,k) + \frac{|j|}{|i|+|j|} * d(j,k))$$

Centroid linkage, also known as Unweighted Pair-Group Method using Centroids (UPGMC), is the distance between the centres of the clusters:

$$d(i+j,k) = \frac{|i|}{|i|+|j|} * d(i,k) + \frac{|j|}{|i|+|j|)} * d(j,k) - \frac{|i|*|j|}{(|i|+|j|)^2} * d(i,j)$$

Median linkage, or Weighted Pair-Group Method using Centroids (WPGMC), is the Euclidean distance between the **weighted** centers of of the clusters:

$$d(i+j,k) = \frac{d(i,k)}{2} + \frac{d(j,k)}{2} - \frac{d(i,j)}{4}$$

Minimum variance, called Ward after H. Ward, Jr., minimizes the total within cluster variance:

$$d(i+j,k) = \frac{|i|+|k|}{|i|+|j|+|k|} * d(i,k) + \frac{|j|+|k|}{|i|+|j|+|k|} * d(j,k) - \frac{|k|}{|i|+|j|+|k|} * d(i,j)$$

Extended distance matrix

In a standard hierarchical clustering method a standard distance matrix is used. Some algorithms implement this as a priority queue, sorting on distances. This can drastically improve run times as priority queues usually are implemented as heaps, giving initial build complexity of O(n), and insertion and removal a complexity of O(log n). This means that the biggest time consumer, finding minimum values, is much lower. But when dealing

with several distance matrices at once, with the possibility of locking, a priority queue is not practical.

The initial implementation used an array of squared distance matrices. Each matrix representing pairwise distance within one vector space/category. This way updates after each iteration of the algorithm is don in near constant time with vectorized operation. Current implementation utilizes condensed distance matrices to save space and help with the biggest time consumer, finding minimum values. This way the complexity of identifying minimum values is halved from O(n*n*c) where n is the number of objects being clustered and C is the number of categories.

3.1.4 Algorithm

When ignoring the all extra functionality as locking, number of end clusters, weighting, deciding categories, different linkage methods and distance measures, the algorithm can be broken down to three steps for each iteration:

- Finding the lowest value in the extended distance matrix, which gives four values, one for category, the two clusters with the lowest distance between them and the distance found.
- Updating the extended distance matrix with new values from the new (x,y)-cluster to all other clusters, as well as removing entries in the distance matrix that is no longer relevant. The function for finding x and y always returns x as having a lower index then y, so the updated values are stored at index x, setting values at index y to ∞. This way the values related to the old x- and y-cluster can no longer be chosen in *get_min*().
- The last step is to update the linkage matrix and store what happened during that iteration.

Listing 3.1: Simplified clustering algorithm

```
def linkage(X)
  dM = make_dM(X)
  Z = []
  while hasValues(dM):
    x, y, category, dist = get_min(dM)
    dM = fix_dm(dM, x, y, category)
    Z.append([x, y, dist, size(x + y)])
  return Z
```

3.1.5 Locking implementation

For a more detailed explanation of locking, see section 2.5. This subsection is about how it is implemented.

No locking

lock_to is an optional argument with a default value of 3. If set to 0, there will be no locking to categories, or restricting partially formed clusters from using parts of the extended distance matrix. No locking means in practical terms that only the values in the distance matrix referencing cluster y, as explained in subsection 3.1.4 will be set to ∞, so all of the distance matrices related to categories not used will also be updated.

Locking set to 1

When *lock_to* is set to 1, the behaviour is partially the same as with no locking, except that only the distance matrix where the lowest value was found is updated as normal. The x and y entries in the other sub distance matrices are set to ∞, to prevent clustering of the newly formed cluster in any other category than what was used, from happening, ie. **locking**.

Locking set to n > 1

In the general case where *lock_to* is set to $n > 1$, $get\_mins()$ is called instead of $get\_min()$. This first calls $get\_min()$ to find the lowest value and indices for that value. Then it calls $get\_min\_cat\_dist()$, a vectorized function to find the lowest values between x and y for each category different than what was returned from $get\_min()$. It then returns those n categories, which has values at index (x,y) lower than infinity, index (x,y) and distance. Note that only the distance between x an y initially found is returned as described in subsection 2.5.2 "Choosing the two closest objects". There is also a function in CatClust.py which takes weights and returns a result of using all the distances between x and y for categories used. This is in the code base not in use as it can result in weird looking dendrograms where the hight of the tree can decrease over time.

3.1.6 Category inheritance

When *lock_to* is set to $n > 0$ partial overlap of categories can occur, as discussed in subsection 2.5.2. So when the join option is selected, categories of the new cluster is the categories from both children clusters. Without the join option selected, only the categories returned from $get\_mins()$ is used, resulting in the intersect behaviour. In implementation this is just an extra step by combining and using the children's categories instead of the result from $get\_mins()$. By default, join is set to false as it highly increases run time and breaks with the idea of not transitive relation between objects.

3.1.7 Using the code

The file "clustering.py" in the code base contains the function "cluster" which combines the most important functionality. This takes file names to donors, reference files and arguments, and returns a clustering using CatClust.

Distance measure supported is:

- Euclidean (C implementation) - $'euclidean'$
- Squared Euclidean (C implementation) - $'sqeuclidean'$
- Correlation (C implementation) - $'correlation'$
- Manhattan (C implementation) - $'cityblock'$
- Relevant distance (custom implementation) - $'relevant\_distance'$
- Weighted threshold (custom implementation) -$'weighted\_threshold'$
- Forbes-Correlation (custom implementation) - $'forbes - corr'$
- Experimental Relevant distance (custom implementation) - $'experimental\_relevant\_distance'$

Linkage supported:

- single
- complete
- average
- centroid

Feature types supported:

- Overlap between donor mutations and reference files - $'overlap'$
- Distance to closest reference file - $'dist\_to\_closest'$
- Forbes-Correlation (used when forbes-corr is distance measure) - $'forbes - corr'$

When Forbes-Correlation is used, bin size and k can be changed from default values of 1M and 0.1. Lock to and join can be set to change the behaviour of the algorithm. Finally *print_progress* can be set to *False* for progress to be muted.

3.1.8 Unit testing

Doctest from the Python library has be used for critical functions and modules in the project. These tests are by default ran when the file is the source of an execution, and not when importing through other files.

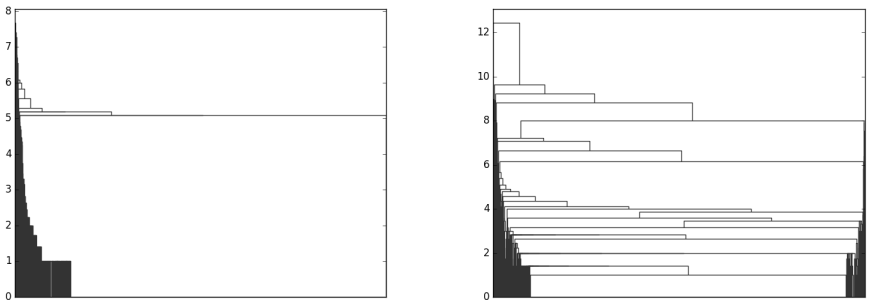
Chapter 4

Results

4.1 Feature creation

4.1.1 Overlaps as features

As discussed in 2.1.1, overlaps can be used to form a relation between mutation points and some segment tracks, which is then used as features in clustering. As expected, the resolution is a problem, and the number of unique combinations of overlaps- and not overlaps values is too small to differentiate all the objects, as shown in 4.1.



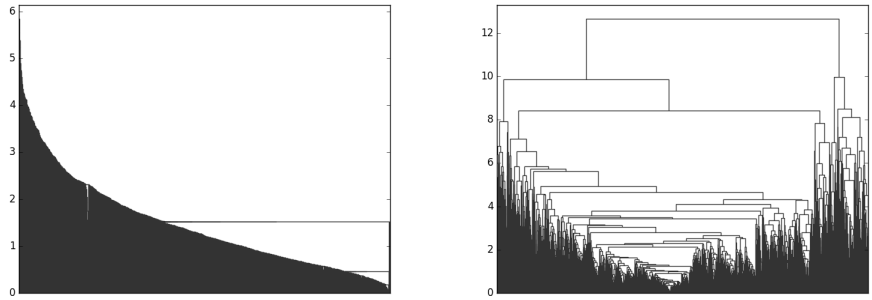
(a) Single linkage (b) Complete linkage

Figure 4.1: Dendrogram plot generated by clustering 1887 mutation point objects with features created from calculating overlaps to 163 DNaseHS-tracks. Hierarchical clustering from python's sciPy package with euclidean distance was used to generate the linkage matrix. Note that 1659 out of 1887 objects had a minimum distance between them of 0.

4.1.2 Distance to nearest segment

With complete linkage, using distance to nearest segment in each of the reference files for each features, seems to do a fairly good job of differentiating the objects in the final clustering. It suffers a bit from size skewness if the tree was to be cut by hight with 7 or more end clusters. This

is of cause even more apparent with single linkage, showing the traditional chain from using minimum values. Plots are shown in figure 4.2



(a) Single linkage (b) Complete linkage

Figure 4.2: Dendrogram plot generated by clustering 1887 mutation point objects with features created from calculating with log of distance + 1, to closest segment in each of the 163 DNaseHS-tracks. Hierarchical clustering from python's SciPy package with euclidean distance was used to generate the linkage matrices.

4.1.3 Bin presence

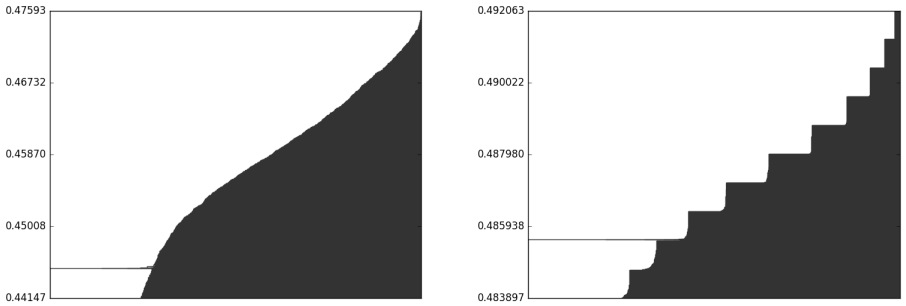
Each object has essentially two sets of features, one for their overall position inside the genome and one for their relation to segment tracks. Their overall position inside the genome is represented as presence within bins as a boolean array for each object.

The two bin sizes used when clustering is 1M base pairs and 10M base pairs, resulting in 3091 and 320 bins. The reason the number of bins for 1M does not work out to $genomesize/1M$ is because bin presence for each chromosome has to be calculated individually and then be combined.

Using single linkage, no locking and only bin presence for clustering, the difference between bin sizes is apparent. With 10M bin size the typical single linkage chain looks like stairs, which is not as apparent with 1M bin size. This means that there are less unique distances between objects, and the clustering algorithm can't properly differentiate objects. So in terms of only using bin presence as features, 320 bins is not enough.

4.1.4 Correlation coefficients

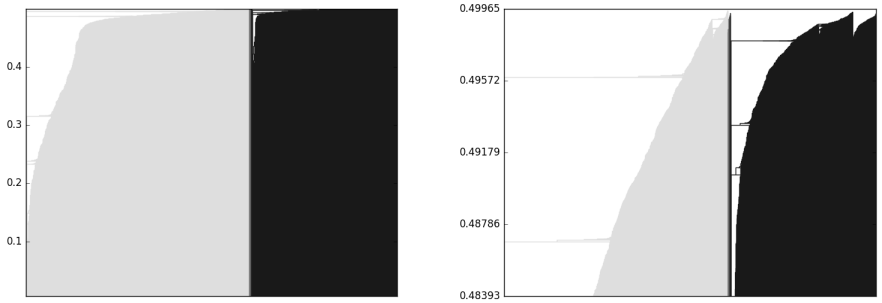
With Forbes-Correlation distance and k set to 0, only correlation between the donor and reference tracks are used. The plots can be viewed in figure 4.4 and show resulting trees with sub clusters coloured by the category most used to form that cluster. It is apparent that the range of values at where objects gets grouped together is very small, all close to 0.5. When comparing donor's correlation numbers to reference sets the formula takes the product of those values and scales them to the range of 0 to 1. This



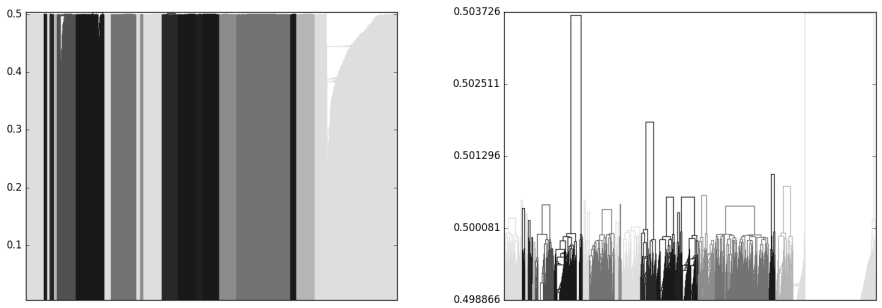
(a) 1M base pair bin size (b) 10M base pair bin size

Figure 4.3: Top 75% plot of clustering with Forbes-Correlation distance and k=1, linkage=single

lacks the ability to properly distribute all the values between 0 and 1, and the majority of distance lays close to 0.5.



(a) Single linkage (b) Top 75% of tree with single linkage



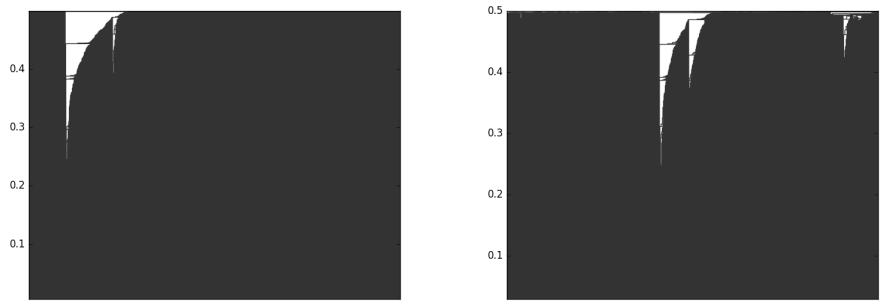
(c) Complete linkage (d) Top 75% of tree with complete linkage

Figure 4.4: Forbes-Correlation distance with no locking and only correlation, coloured by majority category in each sub cluster.

4.1.5 The power of k

Forbes-Correlation distance uses a variable to balance the two part of the equation. When k is 0 only correlation is used and when k is 1 only Forbes, ie. a function of bin presence, is used. $k * forbes + (1 - k) * correlation$

The consequence of changing the scaling is how much the individual parts influence the final distance. As shown in the previous subsection, the correlation part gives a very flat clustering. So the lower the k, the flatter the clustering.

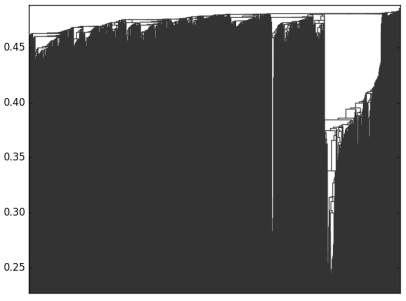


(a) k=0.0

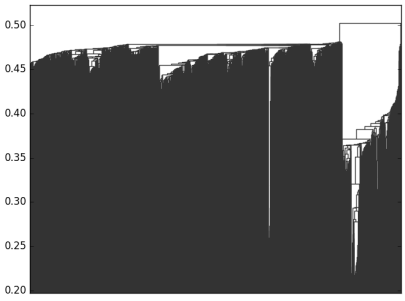
(b) k=0.05

(c) k=0.1

(d) k=0.2



(e) k=0.9



(f) k=1.0

Figure 4.5: Forbes-Correlation with different values of k, complete linkage and no locking.

4.2 Distance measures

In this section I will look at the result of using weighted threshold and relevant distance. Features are the minimum distance from any of the mutation points in each donor, to any of the segments in each of the reference files. The distances then are grouped together by reference set type, and individual distance measures are calculated.

4.2.1 Weighted threshold

Using the weighted threshold function to construct the distance matrix turns out to be problematic. First of all, the function is sensitive to variations in the threshold argument, which is partially revealed when looking only at the number of unique values in the resulting distance matrix when changing threshold.

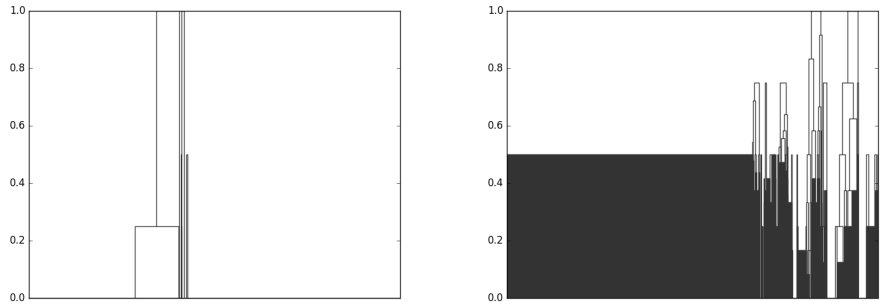
Edge thresholds

The edge cases when using threshold of 0.25 and 0.75 yields a very low amount of unique values. Even when comparing to bin presence which has only 2 unique values and at least some sort of ability to differentiate some of the objects, this turns out to not be the case with weighted threshold. Using 0.25 the resulting clustering has an within cluster distance of 0, which has no value what so ever. With threshold at 0.75 the resulting clustering is similar to 0.25, but instead every object has a distance of 1 - weight (or 0.75 with default arguments) to every other object, so the results are practically the same.

The reason for these results are in how the function works. The core is that it sums up the number of places where both the vectors are over the threshold, or where both are under. The first part being $len(u) - np.sum((u > t)\&(v > t))$, counts the number of positions where both vectors are over a threshold and subtract that from the length of the vector, ie. how many positions have values under the threshold in one or both of the vectors. This means that when a great majority of values are over the threshold "t", this gives a result of 0. The second part ($np.sum((u <= t)\&(v <= t)) * w)/len(u)$) does almost nothing when the first part catches almost all values in all vectors. The end result is that there is enough 0-distance between objects that only 0-distances are used. The same applies for a too high threshold, as the second part catches the great majority of vectors, so the function acts as if it were $1 - 1 * w$.

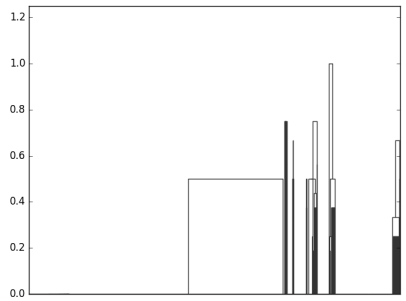
When threshold gets closer to 0.5, the results are more interesting as shown in figure 4.6. The trees still suffer from very flat clustering and both sub plot 4.6a and 4.6c has a majority of sub trees with inner distance of 0. With threshold set to 0.6, the flat part gets raised to 0.5 in the same way seen with too high threshold.

The number of unique values peeks at 245, but even then

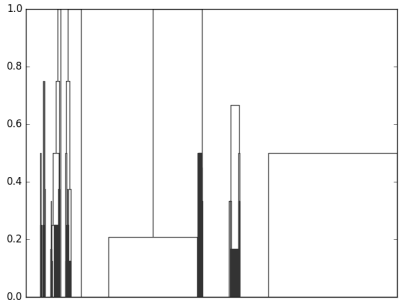


(a) Threshold=0.4

(b) Threshold=0.6



(c) Threshold=0.5



(d) Threshold=0.5085

Figure 4.6: Complete linkage and locking to 3 categories with weight set to 0.5

Threshold	0.25	0.4	0.5	0.6	0.75
Unique values	20	133	245	112	6

Figure 4.7: Number of unique values in distance matrix using weighted threshold

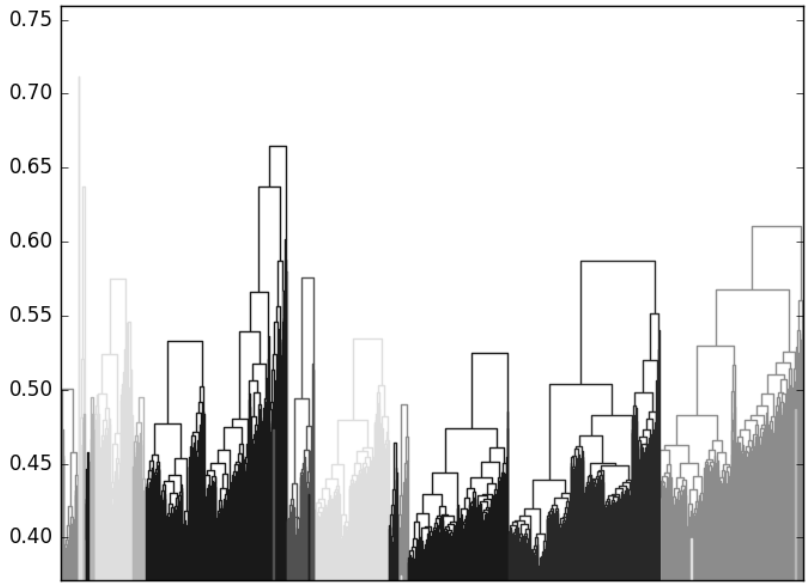
4.2.2 Relevant distance

Overall, using the relevant distance function for pairwise distance fares better than weighted threshold. The problems with almost all minimum values for each clustering iteration being the same, is not present.

4.3 End cluster performance

An important question is if the resulting clusters can catch any signal, and how much better they are able to distinguish donors than what one should expect from just random assignment.

For locking with complete linkage, end clusters are convenient to identify in most cases. This is because the algorithm will stop at the point where there are no more overlapping categories between any of the remaining clusters. When locking was 0 and in table 4.11 when the



clustering algorithm from python's scipy package was used end clusters are not given, since a complete tree is is formed with every object being clustered together. So the trees was cut at a hight where the numbers of end clusters was at least 20, which is similar to most cases where end clusters was given by the algorithm stopping.

Another thing which is not set, is to give each end cluster a "true" label, as clustering has no true class/cluster labels, in contrast to classification. So I have used two methods for labelling the end clusters. Fist the naive way, by the majority of the donor labels in each cluster, both by taking the most occurring group of donor labels and scaling the donor label occurrence so that the total occurrence of each unique donor label adds up to 1. In other terms, when using scaled occurrences, each cluster is labelled to maximize the equivalent of precision since $TP/(TP+FP)$ is the actual numbers in the scaled occurrence matrix. The second way is by using the category which was most used to form every sub cluster in the end cluster, given by the clustering algorithm implemented and the Tree class in CatClust.

The resulting numbers can be viewed in figures 4.9, 4.10, 4.11, 4.12, and 4.13. The tables contains the overall percentage matches from the result of clustering and the average out of 100,000 iterations of randomly shuffling the labels independently of the clusters but keeping the size of each cluster the same. They also contain fold change and p-values. Fold change is defined in different ways, but here it is defined as how many times more than by random chance leaf nodes assigned to the end clusters matched with the end cluster label. For example would fold change 0.5 mean that the number of clustering matches was 50% higher than the average of 100,000 iterations with shuffled leaf nodes, and -1 means that it was half of the

average with Monte Carlo estimation. P-value is the chance of getting a more extreme result by chance, ie. the chance of randomly shuffled leaf nodes yielding a higher overall match with the end cluster label than the result of clustering.

Note that even if the p-values and fold change could be calculated by using the actual probability of a random "clustering" getting better matching numbers than what the actual clustering yielded, these values are approximated by Monte Carlo estimations. So the probabilities might not be completely precise, but with 100,000 iterations, it gives a very close approximation.

4.3.1 Matching results

The actual results are divided by the method of labelling end clusters, and there are several reasons for this, as I shall describe in this sub section.

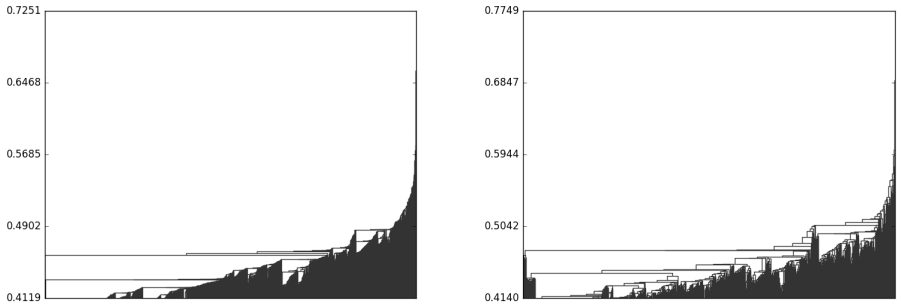
Labelling by leaf label occurrence

When using the highest occurrence of leaf labels for labelling, the results show that both methods of calculating pairwise distance between objects, yields very low p-values and high fold change which indicate that the clustering done with these methods catches some signal in the objects being clustered. Since the end cluster labels are in the set of leaf labels, this shows how well the grouping of donors was done by their cancer type reported by ICGC. And to reiterate, the clustering algorithm had no information about leaf labels and their respectable cancer type. This was added when calculating matches, after the linkage matrices was constructed and clustering was done.

When looking at table 4.10 it is interesting to note that locking to 1 or 3 categories did better than without locking. With single linkage this makes perfect sense, as the number of matches divided by the total number of leaf nodes would be small with the traditional chain of clusters single linkage provides. When cutting the tree, the end clusters would be of greatly different size, with one big and many small. This results in the potential difference between a random "clustering" and the actual clustering to be very small, as the one big cluster will either way contain the majority of the leaf nodes. This imbalance of end cluster sizes can be observed in figure 4.8a, where the chain is apparent. The same also goes for complete linkage shown in figure 4.8b where the chaining is not as severe, but still very much apparent, and the same cluster size imbalance was present.

Labelling by most used category

When using the labels of the reference sets/categories to label the end clusters, the p-values are in most cases much higher than what was the result of using majority cancer type for labelling. One reason for this might be that the relation to the reference sets in not strong enough to properly group donors by the reference sets assumed to be related to cancer types,



(a) Single linkage (b) Complete linkage

Figure 4.8: Upper 75% of the result of using Relevant-Distance and locking to 0 categories

Arguments	Clust.	Avg. rand.	Fold change	p-value
linkage=complete k=0.0 Scaled	0.0706	0.0310	1.2778	0.0014
linkage=complete k=0.0 Not scaled	0.2833	0.2656	0.0668	0.0000
linkage=complete k=0.05 Scaled	0.0453	0.0306	0.4777	0.0349
linkage=complete k=0.05 Not scaled	0.2896	0.2656	0.0905	0.0000
linkage=complete k=0.1 Scaled	0.0879	0.0305	1.8849	0.0003
linkage=complete k=0.1 Not scaled	0.3007	0.2655	0.1323	0.0000
linkage=complete k=0.2 Scaled	0.1316	0.0303	3.3433	0.0000
linkage=complete k=0.2 Not scaled	0.3012	0.2652	0.1357	0.0000
linkage=complete k=0.9 Scaled	0.0800	0.0319	1.5099	0.0001
linkage=complete k=0.9 Not scaled	0.3023	0.2659	0.1369	0.0000
linkage=complete k=1.0 Scaled	0.2428	0.0286	7.4800	0.0000
linkage=complete k=1.0 Not scaled	0.4950	0.2660	0.8610	0.0000
linkage=single k=0.0 Scaled	0.0321	0.0123	1.6153	0.0294
linkage=single k=0.0 Not scaled	0.2586	0.2591	-0.0023	0.5056
linkage=single k=0.05 Scaled	0.0300	0.0134	1.2434	0.0493
linkage=single k=0.05 Not scaled	0.2617	0.2592	0.0096	0.0174
linkage=single k=0.1 Scaled	0.0348	0.0135	1.5814	0.0376
linkage=single k=0.1 Not scaled	0.2707	0.2596	0.0425	0.0001
linkage=single k=0.2 Scaled	0.0348	0.0137	1.5297	0.0338
linkage=single k=0.2 Not scaled	0.2665	0.2600	0.0247	0.0048
linkage=single k=0.9 Scaled	0.1143	0.0172	5.6360	0.0002
linkage=single k=0.9 Not scaled	0.2875	0.2601	0.1052	0.0000
linkage=single k=1.0 Scaled	0.0121	0.0127	-0.0456	0.0072
linkage=single k=1.0 Not scaled	0.2612	0.2655	-0.0161	0.9999

Figure 4.9: Matching numbers with results from using Forbes-Correlation, locking set to 3 and bin sizes of 1MB, compared to random "clustering" with same end cluster sizes. Labelling for each end cluster is done by the highest occurrence of leaf labels in each end cluster, both with scaled occurrence and not.

Arguments	Clust.	Avg. rand.	Fold change	p-value
linkage=complete l=0 Scaled	0.0137	0.0140	-0.0216	0.3274
linkage=complete l=0 Not scaled	0.2680	0.2661	0.0074	0.0498
linkage=complete l=1 Scaled	0.1585	0.0248	5.3871	0.0000
linkage=complete l=1 Not scaled	0.3138	0.2631	0.1928	0.0000
linkage=complete l=3 Scaled	0.1543	0.0235	5.5754	0.0000
linkage=complete l=3 Not scaled	0.3128	0.2625	0.1918	0.0000
linkage=single l=0 Scaled	0.0121	0.0127	-0.0463	0.0074
linkage=single l=0 Not scaled	0.2665	0.2654	0.0038	0.0951
linkage=single l=1 Scaled	0.1738	0.0242	6.1945	0.0000
linkage=single l=1 Not scaled	0.3191	0.2629	0.2140	0.0000
linkage=single l=3 Scaled	0.1748	0.0240	6.2989	0.0000
linkage=single l=3 Not scaled	0.3160	0.2628	0.2022	0.0000

Figure 4.10: Matching numbers with results from using the Relavant-Distance-distance measure compared to random "clustering" with same end cluster sizes. Labelling for each end cluster is done by the highest occurrence of leaf labels in each end cluster, both with scaled occurrence and not

Arguments	Clust.	Avg. rand.	Fold change	p-value
linkage=complete Scaled	0.0827	0.0279	1.9648	0.0005
linkage=complete Not scaled	0.4302	0.2647	0.6254	0.0000
linkage=single Scaled	0.0121	0.0127	-0.0462	0.0074
linkage=single Not scaled	0.2654	0.2655	-0.0002	0.4328

Figure 4.11: Matching numbers with results from using hierarchical clustering from the scipy package with euclidean distance measure, compared to random "clustering" with same end cluster sizes. Labelling for each end cluster is done by the highest occurrence of leaf labels in each end cluster, both with scaled occurrence and not

and that the bin presence is a stronger driver. For Forbes-Correlation-distance, this can be partially observed with complete linkage. K-values of 1 means that only bin presence is used, and the p-values here is almost at 0.5 and the fold change is negative. In contrast does k-values of 0.2 or less well, with the exception of k=0.05, even with a smaller fold change than observed with previous labelling. Another reason might be that the relation between tissue types of the reference sets and cancer types of the donors is not there. That the reference sets still can drive clustering but not towards grouping donors of cancer types that can be mapped to tissue types.

Arguments	Clust.	Avg. rand.	Fold change	p-value
linkage=complete k=0.0	0.0685	0.0570	0.2004	0.0094
linkage=complete k=0.05	0.0532	0.0524	0.0156	0.4048
linkage=complete k=0.1	0.0685	0.0522	0.3118	0.0004
linkage=complete k=0.2	0.0637	0.0516	0.2339	0.0054
linkage=complete k=0.9	0.0574	0.0482	0.1896	0.0219
linkage=complete k=1.0	0.0943	0.0944	-0.0019	0.4959
linkage=single k=0.0	0.0411	0.0415	-0.0112	0.5241
linkage=single k=0.05	0.0437	0.0386	0.1314	0.0382
linkage=single k=0.1	0.0421	0.0390	0.0792	0.1161
linkage=single k=0.2	0.0358	0.0397	-0.0972	0.9157
linkage=single k=0.9	0.0079	0.0326	-0.7578	1.0000
linkage=single k=1.0	0.0948	0.0938	0.0101	0.0000

Figure 4.12: Matching numbers with results from using Forbes-Correlation compared to random "clustering" with same end cluster sizes. Labelling for each end cluster is done by category/reference set label for each end cluster (most used category), given by the clustering algorithm itself.

Arguments	Clust.	Avg. rand.	Fold change	p-value
linkage=complete locking=0	0.0005	0.0001	3.3941	0.0124
linkage=complete locking=1	0.0274	0.0277	-0.0120	0.5031
linkage=complete locking=3	0.0290	0.0281	0.0295	0.3700
linkage=single locking=0	0.0000	0.0000	nan	0.0000
linkage=single locking=1	0.0247	0.0266	-0.0700	0.6794
linkage=single locking=3	0.0242	0.0248	-0.0221	0.5287

Figure 4.13: Matching numbers with results from using Forbes-Correlation compared to random "clustering" with same end cluster sizes. Labelling for each end cluster is done by category/reference set label for each end cluster (most used category), given by the clustering algorithm itself.

Part III

Conclusion

4.4 Conclusion

There was two goals of this thesis, to use data sets of relevant information to enrich the representation of cancer patients with cancer that originated from somatic mutations, and to use this representation to try to identify sub clusters of cell types. Data sets on DNase I Hypersensitive sites was used and methods was developed to form relations between the reference set and donors. Also distance measures and a clustering algorithm was developed to take advantage of this representation in different ways.

The clustering algorithm implemented in this thesis can be seen as a hybrid between clustering and classification. Not in the way that there is a model being trained with labelled data, but that categories is in a way predefined classes. So when the algorithm decides to cluster two objects together it does so within a category (or more than one in case of locking numbers higher than one) and by the classification analogy; labelling the new cluster as part of that class. Depending on the arguments, all the categories does not have to be used and newly formed clusters can be labelled with more than one category. In this sense, the term category is more suitable as it describes the cluster.

The interesting question is if methods developed in this thesis catches a signal and are able to identify sub clusters of cancer patients. But the answer is not straight forward. When testing the ability to cluster objects by cancer types it seems to give good results. Even more so in comparison to the results of randomly shuffling labels. In particular when using Forbes-Correlation for features and distance with complete linkage and k set to 1. Then the end cluster labels set by scaled majority occurrence in each end cluster, match 7.48 times more than by randomly shuffling, and a little over 24% of the donors end up in the correct cluster. In fact both Forbes-Correlation and Relevant distance with complete linkage and locking does better than what random shuffling provided, also in many cases beating out hierarchical clustering from python's scipy package.

But when labelling end clusters by the most used category to form that cluster, the matching is not that good. In most cases it seems to be on par with what is expected by random chance, with most p-values close to 0.5 and small fold change. The exception is Forbes-Correlation with k set to 0.0, 0.1 and 0.2, which has p-values ranging from 0.0004 to 0.0094, but still small fold change. This might be because there is not a strong enough relation formed between donors and cell types in the reference set. Some indication of this is shown by the very flat clustering when only using correlation with reference set as distance in Forbes-Correlation. Another interpretation is that DNase bound to specific cell types helps driving clustering, but not in the way that was expected, such that cell types in breast would group together breast cancer patients, cell types in brain would group together brain cancer patients and so on.

Since I have not configured for multiple testing, it is not possible to conclude whether or not these methods are able to identity sub clusters of cell types, even if there is some indication that it might be possible.

4.5 Future work

The current state of methods and concepts developed in this case are in parts experimental. If further work was to be done using these concepts that breaks with normal norms and similarity, a more theoretical approach to see the mathematical effects of diverging from the norms could be interesting.

It could also be interesting to explore the result of using different data sets as reference and in an environment configured for multiple tests, to better determine practical abilities of the methods in a different context.

Another approach is to optimize the code base by introducing parallellisation and possibly porting to CUDA code so it can run on Nvidia GPUs. As of now everything is running on a single thread and parts of the methods used can greatly benefit from parallellisation.

Bibliography

[1] Elizabeth Pennisi. 'ENCODE Project Writes Eulogy for Junk DNA'. In: *Science* 337.6099 (2012), pp. 1159–1161. ISSN: 0036-8075. DOI: 10.1126/science.337.6099.1159. eprint: http://science.sciencemag.org/content/337/6099/1159.full.pdf. URL: http://science.sciencemag.org/content/337/6099/1159.

[2] *The Galaxy Project*. URL: https://galaxyproject.org/.

[3] Geir Kjetil Sandve et al. *The Genomic HyperBrowser: inferential genomics at the sequence level*. 2010. URL: http://www.ncbi.nlm.nih.gov/pubmed/21182759.

[4] Robert Tibshirani, Guenther Walther and Trevor Hastie. 'Estimating the number of clusters in a data set via the gap statistic'. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423. ISSN: 1467-9868. DOI: 10.1111/1467-9868.00293. URL: http://dx.doi.org/10.1111/1467-9868.00293.

[5] International Cancer Genome Consortium. *ICGC Data Portal*. URL: https://dcc.icgc.org/.

[6] Jason Van Hulse, Taghi M. Khoshgoftaar and Amri Napolitano. 'Experimental Perspectives on Learning from Imbalanced Data'. In: *ICML* (2007), pp. 935–942.

[7] LiMin Fu. 'Knowledge discovery based on neural networks'. In: *Communications of the ACM* 42.11 (1999), pp. 47–50.

[8] Jeppe S. Spicker et al. *Neural network predicts sequence of TP53 gene based on DNA chip*. Tech. rep. Center for Biological Sequence Analysis - Technical University of Denmark - 2800 Lyngby DK, 2002.

[9] David Tamborero, Abel Gonzalez-Perez and Nuria Lopez-Bigas. 'OncodriveCLUST: exploiting the positional clustering of somatic mutations to identify cancer genes'. In: *Oxford Journals* 29 (2013), pp. 2238–2244. URL: https://dx.doi.org/10.1093/bioinformatics/btt395.

[10] M.S. et al. Lawrence. 'Mutational heterogeneity in cancer and the research for new cancer-associated genes'. In: *Nature* 499 (2013), pp. 214–218.

[11] Pax Polak et al. 'Cell-of-origin chromatin organization shapes the mutational landscape of cancer'. In: *Nature* 518 (Feb. 2015), pp. 360–364.

[12] Legault J1 et al. 'Clusters of S1 nuclease-hypersensitive sites induced in vivo by DNA damage'. In: *Molecular and Cellular Biology (MCB)* 17.9 (Sept. 1997), pp. 552–567.

[13] 'Some Methods for classification and Analysis of Multivariate Observation'. In: *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability* 1 (1967), pp. 281–297.

[14] Moisès Burseta and Roderic Guigóa. 'Evaluation of Gene Structure Prediction Programs'. In: *Elsevier* 34 (1996), pp. 353–367. URL: http://dx.doi.org/10.1006/geno.1996.0298.

[15] *International Cancer Genome Consortium (ICGC)*. URL: http://icgc.org/.

[16] *Encyclopedia of DNA Elements (ENCODE)*. An international collaboration of research groups funded by the National Human Genome Research Institute (NHGRI). URL: https://www.encodeproject.org/.

[17] Hyperbrowser. *Classification runs*. URL: https://hyperbrowser.uio.no/ml2/u/krilange/h/mlp-classification-runs.

[18] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. 'Deep learning'. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836. URL: http://dx.doi.org/10.1038/nature14539.

[19] SciPy. *SciPy*. URL: http://www.scipy.org/.

[20] Anil K. Jaoin and Richard C. Dubes. *Algorithms for Clustering Data*. Prestice Hall, 1988.

Appendices

.1 Run times

Run times for clustering, excluding the creation of distance matrices, is largely dependent on the amount of donors and categories. An Intel i7 3770k at 4GHz and 32GB of DDR3 at 1600MHz was used, with both bin sizes 1MB and 10MB, locking numbers of 0, 1 and 3, k-values of 0, 0.05, 0.1, 0.2, 0.9 and 1.0 with both single and complete linkage and 1889 donors. The run time was about 16 hours, or 13 minutes average per clustering with 163 categories (each feature as a category), and 2.5 hours, or 2 minutes average per clustering with 25 categories.

.2 Python tricks

Python standard method call stack size, called recursion limit, is set to 1000. When working with large trees and recursive methods, such as plotting Z through the dendrogram method from scipy, this might throw a runtime error and stop the program. Setting recursion limit with sys.setrecursionlimit(N) can solve this, but should be set cautiously. Depending on system and implementation, memory errors/segmentation faults can occur and the OS will kill the process.

.3 Working with large data

Simple somatic mutation (SSM) data from ICGC data portal has a compressed size of 2.28GB, or 75GB uncompressed.

The files contain a lot of information, but for this thesis only a relatively small amount of the data per mutation points was needed, so the original SSM files were filtered from 42 columns, as shown in table 1, down to the 3 columns sown in table 2. Note that the original files contains chromosome_start and chromosome_end, which is the start and end of a mutation. But since around 96% of all mutations were single based substitution, and the rest were insertion or deletions of under 200 base pairs (usually much fever than 200), I felt it was OK to skip the chromosome_end column and treat mutations as single points.

Due to size and competition constrains, I used the tools built into The Genomic HyperBrowser [3] to handle the initial filtering of the data. This can be done by uploading the file using an URL to the data file using "Upload Data" and cut columns away using "Cut". This reduces the total file size down to 470MB, which is more manageable.

The same goes for the set of reference tracks with segments. These also contain more information than needed, of varying degree depending on the source. Fortunately these tracks can be found in The HyperBrowser file repository, but another process is needed to cut columns. The GSuite branch of The HyperBrowser, which is a more convenient way of handling a large amount of files at the same time, was used. This was done by uploading a GSuite track of format "uri", "title", "file_type" and "track_type" with uris

referring to location of each tracks as in "hb:/Chromatin/Roadmap Epigenomics/DNaseHS/ENCODE_wgEncodeUwDnaseA549PkRep1". These GSuite tracks can also be generated by The HyperBrowser directly. Then The GSuite tracks has to be preprocessed to primary files to be able to manipulate the textual datasets referred to in the GSuite and lastly if the files are to be used locally/downloaded, exporting the primary tracks from GSuite to history and download history.

icgc_mutation_id	icgc_donor_id	project_code
icgc_specimen_id	icgc_sample_id	matched_icgc_sample_id
submitted_sample_id	submitted_matched_sample_id	chromosome
chromosome_start	chromosome_end	chromosome_strand
assembly_version	mutation_type	reference_genome_allele
mutated_from_allele	mutated_to_allele	quality_score
probability	total_read_count	mutant_allele_read_count
verification_status	verification_platform	biological_validation_status
biological_validation_platform	consequence_type	aa_mutation
cds_mutation	gene_affected	transcript_affected
gene_build_version	platform	experimental_protocol
sequencing_strategy	base_calling_algorithm	alignment_algorithm
variation_calling_algorithm	other_analysis_algorithm	seq_coverage
raw_data_repository	raw_data_accession	initial_data_release_date

Table 1: Columns in simple somatic mutation data set from ICGC data portal

icgc_donor_id	chromosome	chromosome_start

Table 2: Columns in simple somatic mutation data set after filtering

.4 The HyperBrowser tool

During the start and middle part of the thesis the code base was integrated into the HyperBrowser, mostly using standard Galaxy functionality. The tool is not at the same state as the code base on GitHub but can be found at https://hyperbrowser.uio.no/ml2/ under Machine learning. This includes both an early state of Category Clustering and MLP classification, focusing on cancer types of mutation points.

Classification

Choosing MLP Classification will show the window in figure 14

- **Genome build** - the genome of which the data files are generated from.
- **Chromosome** - the possibility to filter the data files by only using data entries in one or more chromosomes. The chromosomes should be given as a semicolon separated strings with chromosomes on the format chr* where * is a number, or x or y. Only leaving * will use the entire data files.

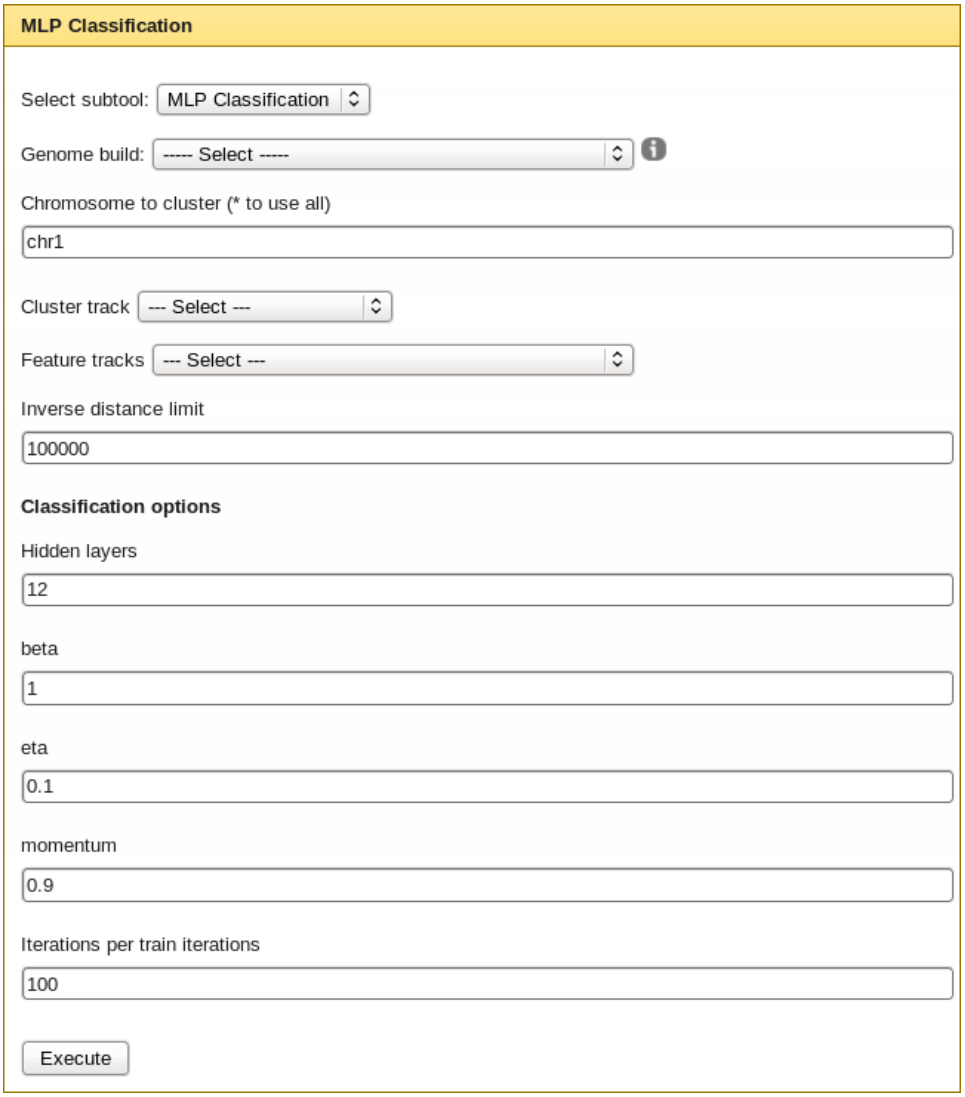


Figure 14

- **Cluster track** - a track from history with at least **chromosome**, **start** and **type**.
- **Feature track** - a GTrack file with uris to files in the HyperBrowser repository.
- **Distance limit** - the limit for creating inverse distance features.
- **Hidden nodes** - the number of nodes in the hidden layer.
- **beta** - adjusts the slope of the activation function. Higher numbers gives a more steep activation curve.
- **eta** - the teaching step works together with momentum to adjust the speed of the training.

- **momentum** - speeds up the training by adding part of the already occurred weight change to the current weight change.
- **iterations** - the number of forward and back propagations between checking the stopping criteria.

Standard values of hidden nodes, beta, momentum, eta and iterations are OK values to use and does not need to be change, but are added to have the option to tweak the training of the MLP. The inverse distance limit may be changed to suit the data sets being used. With a dense track a lower number may be beneficial but with sparse data a low number may not capture the structures the classification is after. Filtering on chromosomes is mainly a way to test the tool as it restricts the tool to only create features and training a model using data in that (or those) chromosomes. This is faster, as creating features can be time consuming.

Clustering

Choosing Category Clustering in the sub tool drop down menu will show the window in figure 15. The clustering tool has the option to cluster tracks instead of lines in a track. After checking "Cluster multiple tracks" under input choices, the tool will present a list of possibly usable tracks from history under "Cluster track". Each track selected will result in one object to be clustered by the algorithm.

- **Input choices** - contains the option to cluster tracks instead of observation in a single track.
- **Genome build**, **chromosomes**, **cluster track**, **feature tracks** and **distance limit** - same as in classification.
- **Feature type** - argument to feature creation, either overlap or inverse distance.
- **Distance measure** - the distance measure used to create the distance matrices. It is advised to use relevant distance.
- **Clustering method** - linkage method to use in clustering.
- **Extra options** - extra options for the tool, such as locking, remove and down sample, and converting segments to points.
- **Lock to** - when "Lock" is selected under extra options, a text box for selecting the number of categories a clustering should lock to is displayed
- **Min. end clusters** - the option to force the algorithm to stop when the total remaining clusters reaches this number.

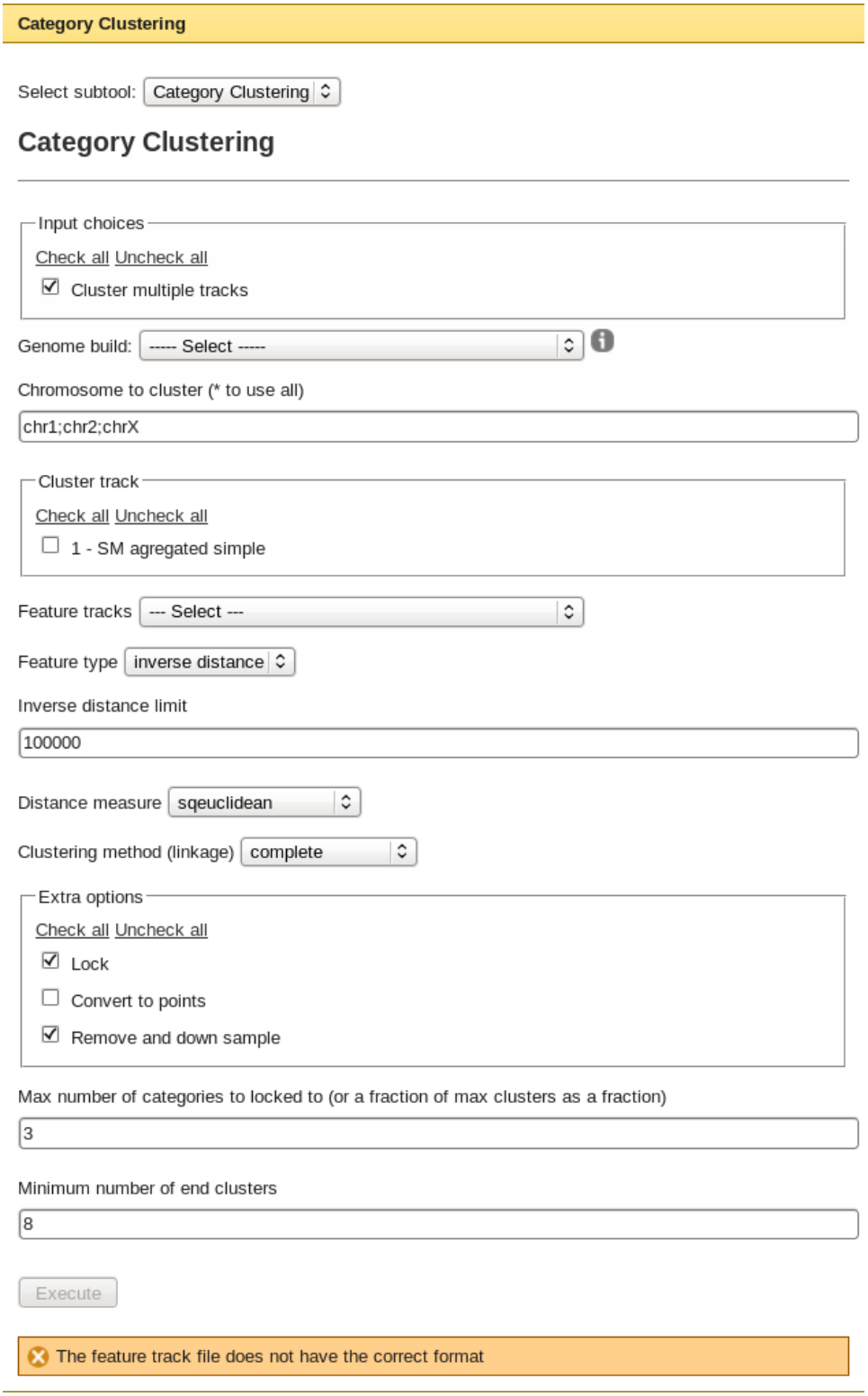

Category Clustering
Select subtool: Category Clustering
Category Clustering
Input choices
Check all Uncheck all
Cluster multiple tracks
Genome build: ----- Select -----
Chromosome to cluster (* to use all)
chr1;chr2;chrX
Cluster track
Check all Uncheck all
1 - SM agregated simple
Feature tracks --- Select ---
Feature type inverse distance
Inverse distance limit
100000
Distance measure sqeuclidean
Clustering method (linkage) complete
Extra options
Check all Uncheck all
Lock
Convert to points
Remove and down sample
Max number of categories to locked to (or a fraction of max clusters as a fraction)
3
Minimum number of end clusters
8
Execute
The feature track file does not have the correct format

Figure 15

.5 Prediction performance

The figures 16, 17, 18, 19, 20 and 21 are performance measures, seen as a unique binary classification for each cluster.

It is important to note that the labels in the left column is not a true label, but rather the label assigned after choosing the highest scaled occurrence of leaf node labels for each end cluster. The reason for scaled occurrences is to not let dominant groups of end labels dominate the entire labelling. As a result the labels are chosen to maximize precision, as TP / (TP / TN) is the actual result of the scaled occurrence matrix used.

This means that the performance numbers should not be seen as proper performance numbers. The underlying true positives are not really true positives since this is not a classification problem where the classifying knows the labels during training.

Sub clust lab.	Accuracy	Precision	Recall	F-score	Category used
prostate	0.8857	0.3750	0.0140	0.0269	pancreas
skin	0.9510	0.0952	0.0263	0.0412	foreskin
esophagus	0.9789	0.0357	0.0714	0.0476	embryonic lung
brain	0.9626	0.0882	0.0698	0.0779	cerebellar
gall bladder	0.9874	0.1429	0.1429	0.1429	breast
colorectal	0.9363	0.0909	0.0471	0.0620	connective
bone	0.8647	0.0039	0.3333	0.0077	muscle
ovary	0.9068	0.1074	0.2667	0.1531	spinal cord
uterus	0.8952	0.0365	0.3333	0.0657	brain hippocampus
lung	0.9847	0.4000	0.2308	0.2927	kidney
kidney	0.9289	0.2000	0.2024	0.2012	cervix
cervix	0.9595	0.0145	0.1000	0.0253	mammary gland
breast	0.7783	0.5891	0.4653	0.5200	induced pluripotent stem cell
bladder	0.8062	0.0054	0.5000	0.0108	embryonic stem cell
blood	0.8984	0.0667	0.0056	0.0103	eye
liver	0.8404	0.1374	0.1933	0.1607	prostate
pancreas	0.8120	0.3333	0.0028	0.0056	epithelium

Figure 16: Relevant distance

Sub clust lab.	Accuracy	Precision	Recall	F-score	Category used
bladder	0.9958	0.0000	0.0000	nan	kidney
blood	0.9036	0.0000	0.0000	nan	cerebellar
bladder	0.9958	0.1667	0.2500	0.2000	prostate
esophagus	0.9916	0.2500	0.0714	0.1111	heart
lung	0.9758	0.0455	0.0385	0.0417	blood vessel
cervix	0.9700	0.0392	0.2000	0.0656	brain hippocampus
bone	0.9858	0.0385	0.3333	0.0690	breast
brain	0.9726	0.0000	0.0000	nan	colon
gall bladder	0.9805	0.0400	0.0714	0.0513	connective
pancreas	0.8115	0.4545	0.0281	0.0529	embryonic stem cell
liver	0.9052	0.1250	0.0333	0.0526	eye
bladder	0.9932	0.0000	0.0000	nan	mammary gland
kidney	0.9521	0.2308	0.0357	0.0619	spinal cord
ovary	0.3481	0.0448	0.9667	0.0857	brain
colorectal	0.9516	0.1818	0.0235	0.0417	embryonic lung
breast	0.7383	0.3478	0.0163	0.0312	foreskin
stomach	0.9505	0.0816	0.0755	0.0784	skin
skin	0.8931	0.1104	0.2368	0.1506	induced pluripotent stem cell
uterus	0.9284	0.0400	0.2381	0.0685	muscle

Figure 17: Weighted threshold

Sub clust lab.	Accuracy	Precision	Recall	F-score	Category used
bone	0.9840	0.0385	0.5000	0.0714	kidney
esophagus	0.9852	0.1739	0.4444	0.2500	gingiva
kidney	0.9519	0.1818	0.0282	0.0488	cerebellar
esophagus	0.9704	0.0851	0.4444	0.1429	epithelium
breast	0.7375	0.7500	0.0344	0.0658	heart
ovary	0.9464	0.0600	0.0698	0.0645	brain hippocampus
brain	0.9643	0.0800	0.0541	0.0645	pancreas
gall bladder	0.9661	0.0943	0.4167	0.1538	spinal cord
uterus	0.9396	0.0449	0.2353	0.0755	prostate
skin	0.8965	0.0887	0.1667	0.1158	blood vessel
lung	0.9803	0.2353	0.1739	0.2000	foreskin
lung	0.9723	0.1333	0.1739	0.1509	liver
gall bladder	0.9298	0.0278	0.2500	0.0500	muscle
uterus	0.9439	0.0375	0.1765	0.0619	breast
bladder	0.9205	0.0079	0.2500	0.0153	induced pluripotent stem cell
cervix	0.9217	0.0161	0.2857	0.0305	brain
uterus	0.9760	0.1071	0.1765	0.1333	embryonic lung
lung	0.9587	0.1452	0.3913	0.2118	blood
skin	0.8226	0.1172	0.5152	0.1910	colon
bladder	0.8219	0.0069	0.5000	0.0137	skin

Figure 18: Forbe-correlation k=0.9, non restrictive labelling

Sub clust lab.	Accuracy	Precision	Recall	F-score	Category used
bladder	0.9817	0.0000	0.0000	nan	kidney
esophagus	0.9853	0.1739	0.4444	0.2500	gingiva
stomach	0.9639	0.0000	0.0000	nan	cerebellar
kidney	0.9352	0.1277	0.0845	0.1017	epithelium
breast	0.7396	0.7500	0.0344	0.0658	heart
brain	0.9493	0.0400	0.0541	0.0460	brain hippocampus
no label	0.0000	0.0000	0.0000	0.0000	pancreas
gall bladder	0.9664	0.0943	0.4167	0.1538	spinal cord
uterus	0.9401	0.0449	0.2353	0.0755	prostate
liver	0.8698	0.1200	0.1271	0.1235	blood vessel
no label	0.0000	0.0000	0.0000	0.0000	foreskin
prostate	0.8765	0.3333	0.0521	0.0901	liver
pancreas	0.7885	0.3426	0.1186	0.1762	muscle
ovary	0.9322	0.0750	0.1395	0.0976	breast
colorectal	0.9016	0.1339	0.2500	0.1744	induced pluripotent stem cell
cervix	0.9224	0.0161	0.2857	0.0305	brain
blood	0.8839	0.1000	0.0253	0.0404	embryonic lung
lung	0.9590	0.1452	0.3913	0.2118	blood
skin	0.8240	0.1172	0.5152	0.1910	colon
bladder	0.8233	0.0069	0.5000	0.0137	skin

Figure 19: Forbe-correlation k=0.9, restrictive labelling

Sub clust lab.	Accuracy	Precision	Recall	F-score
blood	0.9047	1.0000	0.0056	0.0110
prostate	0.8866	1.0000	0.0047	0.0093
colorectal	0.9569	1.0000	0.0122	0.0241
colorectal	0.9569	1.0000	0.0122	0.0241
colorectal	0.9569	1.0000	0.0122	0.0241
colorectal	0.9569	1.0000	0.0122	0.0241
lung	0.8159	0.0556	0.7692	0.1036
blood	0.9074	0.7500	0.0500	0.0938
blood	0.9095	1.0000	0.0556	0.1053
gall bladder	0.9159	0.0814	1.0000	0.1505
prostate	0.8984	1.0000	0.1075	0.1941
blood	0.9117	0.8182	0.1000	0.1782
prostate	0.9037	1.0000	0.1542	0.2672
breast	0.8994	0.8004	0.8184	0.8093
bone	0.6679	0.0048	1.0000	0.0095
pancreas	0.8584	0.8938	0.2845	0.4316

Figure 20: SciPy linkage, euclidean distance, complete linkage, non restrictive labelling

Sub clust lab.	Accuracy	Precision	Recall	F-score
bladder	0.9973	0.0000	0.0000	nan
brain	0.9766	0.0000	0.0000	nan
cervix	0.9941	0.0000	0.0000	nan
esophagus	0.9920	0.0000	0.0000	nan
kidney	0.9548	0.0000	0.0000	nan
liver	0.9196	0.0000	0.0000	nan
lung	0.8159	0.0556	0.7692	0.1036
colorectal	0.9532	0.2500	0.0366	0.0638
ovary	0.9627	0.0000	0.0000	nan
gall bladder	0.9159	0.0814	1.0000	0.1505
skin	0.9473	0.0000	0.0000	nan
blood	0.9117	0.8182	0.1000	0.1782
prostate	0.9037	1.0000	0.1542	0.2672
breast	0.8994	0.8004	0.8184	0.8093
bone	0.6679	0.0048	1.0000	0.0095
pancreas	0.8584	0.8938	0.2845	0.4316

Figure 21: SciPy linkage, euclidean distance, complete linkage, restrictive labelling