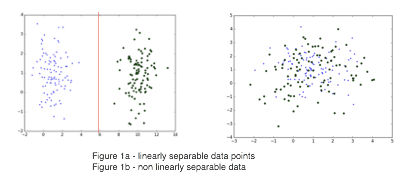
**2. Project background chapters**

**2.1 Background Concepts**

2.1.1 - Classifiers

The field of machine learning is all about giving computers the power to classify and predict on data in a similar fashion to us humans, via means of pattern recognition and data similarity. The task of classification involves categorising sets of data instances, such as sensory data or digital images, into separate well defined classes, for example classifying images of animals into dogs, cats, horses etc. Classification can be multi-class or binary, with certain algorithms (e.g SVM) limited to binary classification between two class values. These binary classifiers can be effectively used as multi-class classifiers however, by employing a one-versus-one scheme, where each class is compared to every other class one by one, or a one-versus-all scheme, which compares each class with all the other classes as one. Classification is an example of supervised learning in machine learning, which is to say the classifier is trained based on labelled training data, and learns an approximation of a truth function to compute the class of unlabelled data.

The general approach with classification algorithms is to define a feature space vector , with d features, as a representation of the classification data. For binary classification, this set of feature vectors across the training data is then input to the classification algorithm and a positive or negative class value (often +1 or -1) is chosen if is above or below a certain threshold. In an alternate view, this means we want data to be linearly separable in Euclidean space, wherein a hyperplane (i.e a flat line) can split all data points of each class. This can be visualised in figure 1a. The data points for the 2 classes form 2 single, easily linearly separable clusters, separable by a hyperplane such as the red line. Whereas, the data in figure 1b has no simple linear boundary which can separate it.

When data is not linearly separable in its feature vector representation, it can often be transformed into a different dimensional space, one where the data is able to be linearly separated. This is a mapping which transforms the data into a new dimension, where linear classification can be performed. The linear boundary in the induced space will become a complex boundary in the original feature space. This method gives rise to some problems however; transforming to greatly increases the computational load of the classifier, especially in situations where we are transforming to much higher dimensions.

To exploit the utility of mapping to a higher-dimension space without incurring such high computational penalties, an implicit mapping into can be achieved without having to explicitly compute the induced space feature vector. This is known as the kernel trick, which replaces dot products in classification algorithms with kernel functions, allowing the dot products of a high dimensional space to be computed without explicitly computing the transformed feature vectors.

2.1.2 - Kernel Methods

A kernel function is defined in this paper as a symmetric function where is a finite set. For well-grounded theoretical results, the kernel function must also be positive semi-definite

(psd), which means to satisfy the condition

,

which is to say, all values in the kernel matrix must be non-negative.

Kernels compute dot products of vectors in some transformed space:

(1)

where *X* is a non-empty finite set, is a transformation on , and the kernel function is thus symmetric, . Such functions arise naturally in many linear classification and regression models when a dual representation is derived; this is why they are such a powerful tool in machine learning. If we consider linear regression, we can see how a dual form can be derived, given purely in terms of kernel functions.

Linear regression models a continuous variable on a function of its data inputs:

(2)

and learns its parameters by minimising a sum-of-squares error function :

(3)

with . If we set the derivative of with respect to equal to zero, the solution for is a linear combination of the vectors (the transformation on each ), with coefficients

(4)

where is the matrix whose nth row is given by , and is the vector , with

(5)

Now we can redefine in terms of the parameter vector , by substituting . This gives us the dual representation , as below, with the square multiplied out:

(6)

where . The term is the form of a Gram matrix, as described in section 2.1.3; i.e , hence we can define . This means that can be written using

(7)

----- TODO finish dual rep derivation -------------

Kernel functions are used as a measure of similarity in the induced space between the two inputs, and . There are generally two classes of functions, polynomial and radial basis kernels. The simplest kernel example is the linear kernel, where is simply the identity mapping i.e , and thus the kernel is .

The quadratic kernel is another simple kernel, which is be shown to be a kernel below.

For some vectors , and , the quadratic kernel is defined as . To show this is a kernel, it must be shown that for some.

Defining the mapping we can then expand, letting :

Hence, we have a valid kernel, as per (1).

Given a valid kernel, it is easy to construct other valid kernels based on some known properties, as summarised in table 1 in the appendix. One can construct a complex kernel to fit the needs of a problem, while being sure that it is still positive semi-definite.

We can take the Gaussian kernel as an example of a valid kernel, verifiable using the kernel construction rules:

If we expand the square term :

  and rewrite :

Using 5.10 from table 1 and the linear kernel we can see that is a valid kernel, 5.13 shows is a valid kernel, and 5.11 shows the whole kernel is valid.

2.1.3 Gram Matrix

For a set of inputs , we can also define the kernel, or Gram matrix as , with  . This matrix is positive semi-definite for valid kernels. With this kernel matrix, we can effectively verify if the associated kernel is valid by computing all the eigenvalues of the kernel matrix for an input data set. The kernel is valid, at least for the particular data set, if all of the eigenvalues are non-negative. This allows for a quick check of kernel validity, and can be combined with the kernel construction techniques to rigorously prove if a kernel is valid.

2.1.4 Kernel PCA

Principal Component Analysis (PCA) is a technique to transform a number of possibly correlated variables into a smaller number of variables called principal components (PCA, Mark Richardson), i.e it transforms a feature vector space into a lower dimensional space, consisting of the principal components, wherein trends and outliers can be revealed for analysis. The principal components are the directions of the data with the greatest variance, the directions where the data is most spread out; the greatest variance lies on the first principal component, and so on.

PCA performs a diagonalisation of the covariance matrix defined as:

(2)

for data , zero-centred s.t . From , the associated eigenvalues and eigenvectors are found according to , with the -th highest eigenvalues corresponding to the first principal components.

If the data is mapped into a feature space via some mapping , we can perform kernel PCA on the mapped data to obtain a lower dimensional representation by virtue of C being defined in terms of dot products:

From this definition of we can obtain the principal components as above, or equivalently, we are finding the eigenvectors (principal components) of the normalized and centered version of the kernel matrix .

As an aside, it is also possible to present the kernel matrix as a summation of its eigenvectors and eigenvalues:

(3)

**2.2 Approaches to sequential data analysis**

A sequence is defined as an ordered set of symbols , with , and being a symbol space containing discrete or continuous symbols. Sequential analysis seeks to find similarity between sequences, on a symbolic and/or structural level.

2.2.1 Bag-of words model

The bag-of-words model is a simple technique used mainly in natural language processing (NLP) for text classification, with adaptations into other areas such as object categorization. A document gets represented as its ‘bag-of-words’, a vector containing the counts of the words or n-grams found in the document. An n-gram is a substring of length n, and can be placed as window across the document or string, to obtain a count vector. There are many other variants of the model; some count words with similar stems as the same word for example. Vectors of term frequencies for all the documents in a classification set can then be easily used to train a classifier, such as Naïve Bayes. This simple vector representation makes bag-of-words a good approach for suitable problems, providing a ready-made input for vectorial similarity measures.

Similarly, instead of defining a dictionary which words can belong to, we can define an alphabet of symbols in order to use the bag-of-words method on strings, treating individual symbols as we would a word in document. This method is clearly limited, in that it says nothing of the order of words/symbols, which is a big issue in dealing with sequences of symbols; words in a document have some semantic value when removed from their place in the document, but symbols within a sequence are likely to have most of their information residing in the relationship to other symbols in the sequence. Furthermore, it cannot deal with a continuous ‘alphabet’, as this could lead to an arbitrarily large term frequency vector; and sequential data where there is no obvious alphabet.

2.2.2 Edit distances

Edit distances are a technique to assess the similarity of two sequences based upon the cost of transforming one to the other. The edit distance is a dissimilarity measure metric, returning 0 if no transformation is required to make the sequences the same, and some positive value if transformations are required. Edit distances can be very useful if the transformations between sequences and their associated costs are well defined, making it once again more useful for strings or even text documents, where transformations between strings consist of one of three operations: *insertion* of a symbol into the string; *deletion* of a symbol in the string; or *substitution* of a symbol of the string for another symbol. Then, the similarity of two strings is the minimal total cost of operations to transform between the strings. As an example, the transformation between the strings ‘bath’ and ‘boat’ consists of an insertion and a deletion, giving them a similarity of *I + D,* with *I* and *D* the insertion and deletion costs, respectively. Clearly this method is unsuitable for wider sequential analysis, where operations to transform sequences into one another are less well-defined.

2.2.3 String Kernels

String kernels appear much the same as other sequential similarity measures, but can exploit the power of kernel methods’ inner product feature space representation. This implicit feature space allows for a wide selection similarity measures based on the geometry and characterisation of sequences or strings, offering more options than a simple representation such as bag-of-words. A simple example of a kernel offering geometric information about 2 sequences in a feature space is to define Euclidean distance between 2 strings in terms of kernels:

The only requirements for a function to be a kernel are that it is both symmetric and positive semi-definite, hence there exist a great number of possible kernels for which there are corresponding implicit inner-products in some unknown feature space; string kernels include kernels on n-grams or words, kernels involving string alignments, or kernels for mismatches and gaps.

One such kernel is the string subsequence kernel, which defines a feature space over all subsequences of length , giving the induced space a dimensionality of . The function mapping into the feature space is given by , with a subsequence frequency function, giving a value of how frequently a substring appears in the input string s. This gives a measure of similarity based upon how many substrings two sequences have in common, and prises greater structure from the sequences than words or n-grams can.