Investigating the Impact of Hubness on SVM Classifiers

February 20, 2011

Submitted to the University of the Aegean

Department of Information & Communication Systems Engineering

In partial fulfillment of the requirements for the Degree of

Master in Information Management

Kouimtzis Georgios

Supervisors

Efstathios Stamatatos

Assistant Professor

Department of Information & Communication Systems Engineering

University of the Aegean

Committee Members

Alexandros Nanopoulos

Professor

Institute of Computer Science

University of Hildesheim, Germany

Georgios Vouros

Professor

Department of Information & Communication Systems Engineering

University of the Aegean

Preface

Support Vector Machines is a well known classification algorithm. Its very good performance has been proven theoretically and observed in real life data sets. One of its disadvantages is that it can be time consuming with regard to training time, especially when the training data set consists of a lot of samples, even though the final solution can depend only on a small fraction of all the training samples.

Recent works have demonstrated the existence of hubs in high dimensional spaces. Hubs are points that are very popular among other points. That is in a set of data points in a high dimensional space, a big number of these have the hub points as their nearest neighbors.

This work concentrates on the study of the relation between hubs and support vector machines and especially on the differences between high and low dimensionality.

In chapter 1 we give background knowledge on support vector machines and hubs. At the end of the chapter we introduce the data sets that we use throughout this work. In chapter 2 we explore how bad hubs relate to support vector machines and especially support vectors. In chapter 3 we introduce hubness ratio and look closer at the importance of samples with hubness ratio one. Chapter 4 exploits the knowledge gained in the previous two chapters and defines two instance selection methods. Chapter 5 shows results of the comparison between our methods and the random instance selection method. Finally chapter 6 summarizes the main contributions of this work and points directions of future work.

acknowledgements

I would like to thank my supervisors Alexandros Nanopoulos and Efstathios Stamatatos for their assistance. Without their help and guidance it would not be possible to complete this work. I would like also to thank the Aegean University and all my professors for giving me the opportunity to attend the MSc program. It has been a wonderful experience. Finally i would like to thank my friends in Karlovasi Samos and my family for their support.

Contents

Chapter 1

Introduction

Machine learning is a scientific discipline that is concerned with the design of algorithms that allow computers to learn from data. Examples of learning problems include:

- Identify handwritten numbers based on digitized images
- Predict whether a patient suffers from a certain disease based on her clinical exams
- Designate an email message as spam or not based on its subject and text
- Predict the price of a stock based on company performance measures

In all of these cases we have an *output*, usually quantitative (stock price) or categorical (spam email/not spam email), that we wish to predict based on a set of *inputs*. If the output variable is categorical then the problem is called *classification.*

In supervised learning one has a *training set* of data that consists of a set of objects (email messages) along with their input and output measurements. Using this data we build a prediction model, or *learner*, which will enable us to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome. The ability of the learner to categorize correctly new objects is known as *generalization.*

Support Vectors Machines (SVMs) is a classification algorithm that achieves great results, which are justified both theoretically and in practice.

Both inputs and outputs are usually expressed as vectors in some, possibly high, dimensional space. The *Curse of Dimensionality* generally refers to the fact that what intuition and practice suggest for two and three dimensions usually breaks down in higher dimensions. One aspect of the curse of dimensionality is the appearance of hubs in high dimensions, that is points that are popular nearest neighbors.

This essay is about exploring and exploiting the impact of hubness on SVM classifiers.

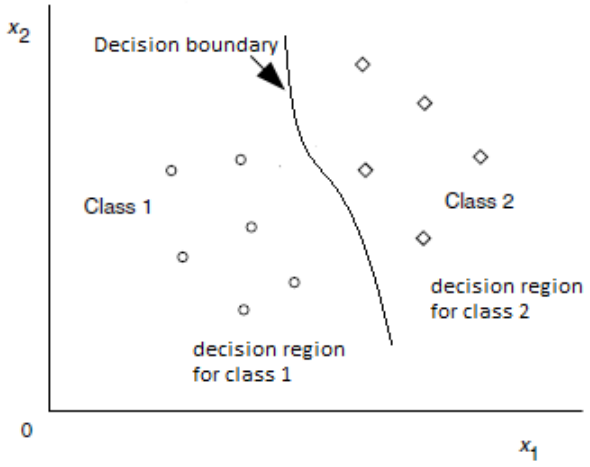


Figure 1.1: Decision function in two dimensional input space

1.1 Support Vector Machines

As already stated SVMs are classification algorithms that achieve high generalization and this ability has been established both theoretically and in practice.

In this section we develop SVMs, starting from the linearly separable case, moving to the case where the classes overlap and continuing to the case of non linear decision boundaries. Finally we consider training time issues and introduce instance selection methods.

Here we consider only the two class SVM algorithm.

1.1.1 Linearly separable classes

In classification the input space is divided into *decision regions* whose boundaries are called *decision boundaries*, or *decision surfaces* (see figure 1.1). In *linear classification models* the decision boundary is a linear function of the input vector and hence defines a (D-1)-dimensional hyperplane within the D-dimensional inputs space. Data sets whose classes can be separated exactly by linear decision surfaces are said to be *linearly separable*.

For two-class, separable training data sets, such as the one in figure 1.2, there are lots of possible linear separators. Intuitively, a decision boundary drawn in the middle of the void between data items of the two classes seems better than one which approaches very close to examples of one or both classes. While some learning methods find just any linear separator, others, search for the best linear separator according to some criterion. The SVM in particular defines the criterion to be looking for a decision surface that is maximally far away from any data point. This distance from the decision surface to the closest

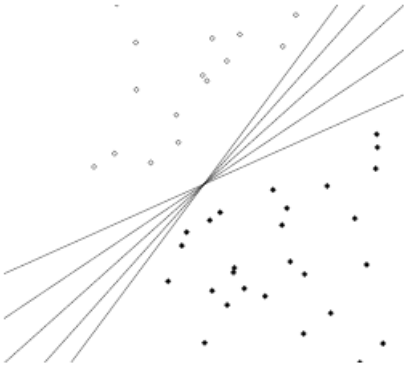


Figure 1.2: There are an infinite number of hyperplanes that separate two linearly separable classes.

data point determines the *margin* of the classifier. This method of construction necessarily means that the decision function for an SVM is fully specified by a (usually small) subset of the data which defines the position of the separator. These points are referred to as the *support vectors*. Figure 1.3 shows the margin and support vectors for a sample problem. Other data points play no part in determining the decision surface that is chosen.

We formalize more mathematically the SVM. Let $\{(\vec{x_i}, y_i)\}$ be a set of training data, consisting of pairs of a point $\vec{x_i}$ and its class label y_i. Class label takes the values $+1$ and -1.

Any linear decision boundary can be defined as $\vec{w}^T\vec{x} + b = 0$ (see figure 1.4). The distance from a point $\vec{x'}$ to the hyperplane is:

$$r = \frac{\vec{w}^T\vec{x'} + b}{\|\vec{w}\|}$$

where $\|\vec{x}\|$ is the Euclidean norm of vector $\vec{x}$.

Assuming that all training data are at least distance one from the hyperplane, then:

$$\begin{aligned} \vec{w}^T\vec{x_i} + b \geq 1,\ y_i &= 1 \\ \vec{w}^T\vec{x_i} + b \leq -1,\ y_i &= -1 \end{aligned}$$

or $y_i\left(\vec{w}^T\vec{x_i} + b\right) \geq 1$ for all training samples, and the support vectors are at distance one from the hyperplane. The geometric margin ρ is then twice as big, or

$$\rho = \frac{2}{\|\vec{w}\|}$$

To maximize the margin we can formulate the following optimization problem:

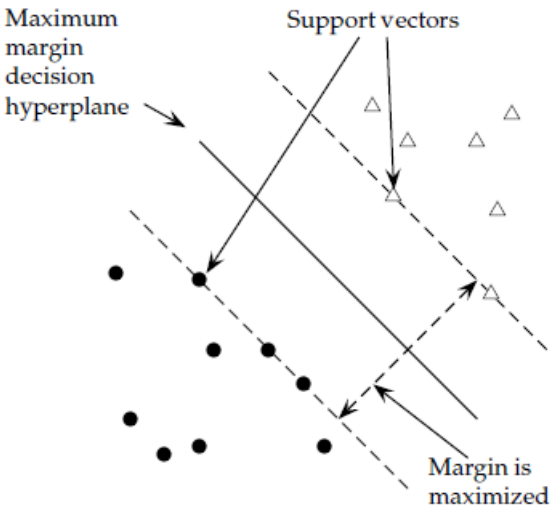


Figure 1.3: Margin and support vectors for a two dimensional problem. The support vectors are the five points right up against the margin of the classifier.

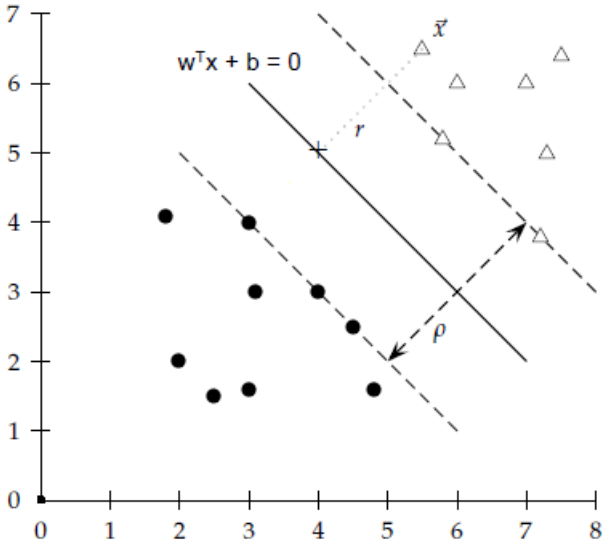


Figure 1.4: The geometric margin of a point (r) and a decision boundary (ρ)

- maximize $\frac{2}{\|\vec{w}\|}$
- such that $y_i \left(\vec{w}^T \vec{x_i} + b\right) \geq 1$ for all training data.

Changing the above maximization problem to a minimization one gives:

- minimize $\frac{1}{2}\vec{w}^T\vec{w}$
- such that $y_i \left(\vec{w}^T \vec{x_i} + b\right) \geq 1$ for all training data.

We are optimizing a quadratic function subject to linear constraints. This is a *quadratic optimization* problem and many algorithms exist for solving them. The solution involves constructing a dual problem where a Lagrange multiplier a_i is associated with each constraint $y_i \left(\vec{w}^T \vec{x_i} + b\right) \geq 1$ in the primal problem, and it has the following form:

Find $a_i, \ldots, a_N$ such that $\sum a_i - \frac{1}{2}\sum_i \sum_j a_i a_j y_i y_j \vec{x_i}^T \vec{x_j}$ is maximized and:

- $\sum a_i y_i = 0$
- $a_i \geq 0$ for all $1 \leq i \leq N$

The solution is then of the form:
$\vec{w} = \sum a_i y_i \vec{x_i}$
$b = y_k - \vec{w}^T \vec{x_k}$ for any $\vec{x_k}$ such that $a_k \neq 0$
In the solution most of the a_i are zero. Each non zero a_i indicates that the corresponding $\vec{x_i}$ is a support vector. The classification function is then:

$$f(\vec{x}) = sign\left(\sum_i a_i y_i \vec{x_i}^T \vec{x} + b\right)$$

1.1.2 Soft margin classification

If the training set is not linearly separable, the standard approach is to allow the fat decision margin to make a few mistakes (some points – outliers or noisy examples – are inside or on the wrong side of the margin). We then pay a cost for each misclassified example, which depends on how far it is from meeting the margin requirement given by the equation $y_i \left(\vec{w}^T \vec{x_i} + b\right) \geq 1$. To implement this, we introduce slack variables ξ_i. A non-zero value for ξ_i allows $\vec{x_i}$ to not meet the margin requirement at a cost proportional to the value of ξ_i. See figure 1.5.

The formulation of the SVM optimization problem with slack variables is:

Find $\vec{w}$, b, and $\xi_i \geq 0$ such that:

- $\frac{1}{2}\vec{w}^T\vec{w} + C\sum_i \xi_i$ is minimized
- and for all $\{(\vec{x_i}, y_i)\}$, $y_i \left(\vec{w}^T \vec{x_i} + b\right) \geq 1 - \xi_i$

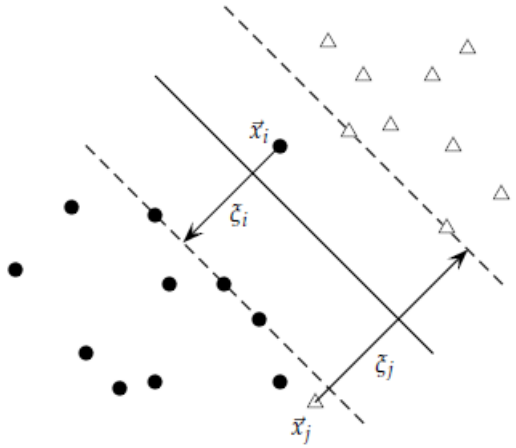


Figure 1.5: Maximum margin classification with slack variables

The optimization problem is then trading off how fat it can make the margin versus how many points have to be moved around to allow this margin. The margin can be less than 1 for a point $\vec{x_i}$ by setting $\xi_i > 0$, but then one pays a penalty of $C\xi_i$ in the minimization for having done that. The sum of the ξ_i gives an upper bound on the number of training errors. Soft-margin SVMs minimize training error traded off against margin. The parameter C is a regularization term, which provides a way to control overfitting: as C becomes large, it is unattractive to not respect the data at the cost of reducing the geometric margin; when it is small, it is easy to account for some data points with the use of slack variables and to have a fat margin placed so it models the bulk of the data.

The dual problem for soft margin classification becomes:

Find $a_i, \ldots, a_N$ such that $\sum a_i - \frac{1}{2} \sum_i \sum_j a_i a_j y_i y_j \vec{x_i}^T \vec{x_j}$ is maximized and:

- $\sum a_i y_i = 0$
- $0 \leq a_i \leq C$ for all $1 \leq i \leq N$

Neither the slack variables ξ nor Lagrange multipliers for them appear in the dual problem. All we are left with is the constant C bounding the possible size of the Lagrange multipliers for the support vector data points. As before, the $\vec{x_i}$ with non-zero a_i will be the support vectors.

The solution of the dual problem is of the form:
$\vec{w} = \sum a_i y_i \vec{x_i}$
$b = y_k (1 - \xi_k) - \vec{w}^T \vec{x_k}$ for $k = argmax_k a_k$

Typically, the support vectors will be a small proportion of the training data. However, if the problem is non-separable or with small margin, then every data point which is misclassified or within the margin will have a nonzero a_i.

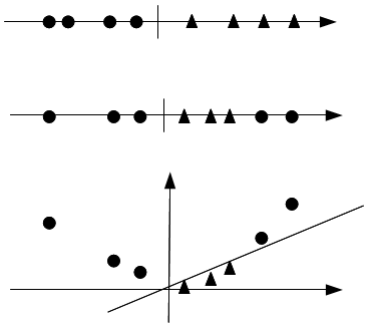


Figure 1.6: Projecting data that is not linearly separable into a higher dimensional space can make it linearly separable

1.1.3 Nonlinear SVMs

With what we have presented so far, data sets that are linearly separable (perhaps with a few exceptions or some noise) are well-handled. But what are we going to do if the data set just doesn't allow classification by a linear classifier? Let us look at a one-dimensional case. The top data set in Figure 15.6 is straightforwardly classified by a linear classifier but the middle data set is not. We instead need to be able to pick out an interval. One way to solve this problem is to map the data on to a higher dimensional space and then to use a linear classifier in the higher dimensional space. For example, the bottom part of the figure shows that a linear separator can easily classify the data if we use a quadratic function to map the data into two dimensions. The general idea is to map the original input space to some higher-dimensional *feature space* where the training set is separable.

SVMs provide an easy and efficient way of doing this mapping to a higher dimensional space, which is referred to as "the kernel trick". The SVM linear classifier relies on a dot product between data point vectors. Let $K(\vec{x_i}, \vec{x_j}) = \vec{x_i}^T \vec{x_j}$. Then the classifier we have seen so far is:

$$f(\vec{x}) = sign\left(\sum_i a_i y_i K(\vec{x_i}, \vec{x}) + b\right) \tag{1.1}$$

Now suppose we decide to map every data point into a higher dimensional feature space via some transformation $F : \vec{x} \mapsto \phi(\vec{x})$. Then the dot product becomes $\phi(\vec{x_i})^T \phi(\vec{x_j})$. If it turned out that this dot product (which is just a real number) could be computed simply and efficiently in terms of the original data points, then we wouldn't have to actually map from $\vec{x} \mapsto \phi(\vec{x})$. Rather, we could simply compute the quantity $K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i})^T \phi(\vec{x_j})$, and then use the function's value in Equation 1.1. A kernel function K is such a function that corresponds to a dot product in some expanded feature space.

The two commonly used families of kernels are polynomial kernels and radial basis functions. Polynomial kernels are of the form $K(\vec{x}, \vec{z}) = \left(1 + \vec{x}^T \vec{z}\right)^d$. The

case of $d = 1$ is a linear kernel, which is what we had before the start of this section. The case of $d = 2$ gives a quadratic kernel, and is very commonly used.

The most common form of radial basis function is a Gaussian distribution, calculated as:

$$K(\vec{x}, \vec{z}) = e^{-\frac{(\vec{x}-\vec{z})^2}{2\sigma^2}}$$

A radial basis function (rbf) is equivalent to mapping the data into an infinite dimensional Hilbert space.

A polynomial kernel allows us to model feature conjunctions (up to the order of the polynomial). Simultaneously we also get the powers of the basic features. A radial basis function allows us to have features that pick out circles (hyperspheres) – although the decision boundaries become much more complex as multiple such features interact. Beyond these two families, there has been interesting work developing other kernels.

1.1.4 Instance selection

In training a SVM, we need to solve a quadratic programming problem with the number of variables equal to the number of training data. Computational complexity is of the order of M^3, where M is the number of training data. Thus when M is large, training takes long time. To speed up training, numerous methods have been proposed. One is to extract support vector candidates from the training data and then train the support vector machine using these data.

According to the architecture of the support vector machine, only the training data near the decision boundaries are necessary. In addition, because the training time becomes longer as the number of training data increases, the training time is shortened if the data far from the decision boundary are deleted. Therefore, if we can delete unnecessary data from the training data efficiently prior to training, we can speed up training. Several approaches have been developed to preselect support vector candidates.

Almeida et al (2000) conducted k-means clustering first on the entire training set regardless of patterns' class-membership.. Those clusters which contain patterns from one class are called homogeneous, while those which do not are called heterogeneous. All the patterns from a homogeneous cluster are replaced by a single centroid pattern, while the patterns from a heterogeneous cluster are all selected. The k-means clustering algorithm is not a requisite. It may be substituted by other clustering algorithms. The drawback of this research is that it is not clear how to determine the number of clusters. and that if classes are well separated, it may not extract boundary data.

Koggalage and Halgamuge (2004) also employed clustering to select the patterns from the training set. It is quite similar to Almeida et al. (2000) approach in that they conducted clustering on the entire training set first and chose the patterns which belong to the clusters having heterogeneous members. For a homogeneous cluster, on the contrary, the patterns along the rim of cluster were

selected not the centroid. It is relatively a safer approach since even for homogeneous clusters there can exist the patterns near the decision boundary if the cluster's boundary is almost in contact with the decision boundary. On the other hand, it has a relative shortcoming as well in that the patterns far away from the decision boundary are also picked as long as they lie along the rim. And further, it is still vague how to set the radius and how to define the width of the rim from it.

Shin and Cho (2005) used k nearest neighbors and three different neighborhood properties of the samples in order to choose support vector candidates. The three neighborhood properties were: (i) a sample located near the decision boundary tends to have more heterogeneous neighbors in their class membership (ii) an overlap or a noisy sample tends to belong to a different class from its neighbors and (iii) the neighbors of a sample located near the decision boundary tend to be located near the decision boundary as well.The first property is used to identify samples located near the decision boundary, the second to remove samples located on the wrong side of the decision boundary and the third to skip calculations of unnecessary distances between samples, thus accelerating the samples selection procedure.

Yang and Ahuja (2000) defined a superset of support vectors called guard vectors. They used linear programming in order to extract this set. Assume that the set of training input–output pairs $\{(x_1, y_1), \ldots, (x_M, y_M)\}$ is linearly separable, where y_i are either 1 or -1. Then there exists a separating hyperplane that includes a data sample x_i:

$$\begin{aligned} y_1\left(\vec{w}^T\vec{x_1} + b\right) &\geq 0 \\ &\vdots \\ y_{i-1}\left(\vec{w}^T\vec{x_{i-1}} + b\right) &\geq 0 \\ y_i\left(\vec{w}^T\vec{x_i} + b\right) &= 0 \\ y_{i+1}\left(\vec{w}^T\vec{x_{i+1}} + b\right) &\geq 0 \\ &\vdots \\ y_M\left(\vec{w}^T\vec{x_M} + b\right) &\geq 0 \end{aligned} \tag{1.2}$$

We call the data sample that satisfies 1.2, a guard vector. Clearly the support vectors are guard vectors but the reverse is not true. Thus the set of guard vectors is a superset of the set of support vectors. To obtain the set of guard vectors, we solve 1.2 by linear programming, changing i from 1 to M. Unlike usual linear programming we need only to check whether x_i is a feasible solution of 1.2. If the training data set is not linearly separable, we need to map the original input space into the feature space. Instead, Yang and Ahuja (2000) in their work, the original input space is extended to the $(m + M)$-dimensional space where m is the dimension of $\vec{x}$ and the $(m + i)$th element is 1 for $\vec{x_i}$ and 0 otherwise.

Finally, Zechner and Granitzer (2009) use the following method to extract support vector candidates. First they grow two topological networks for each one of the two classes using their Sparsifying Neural Gas (SNG) algorithm. In the SNG algorithm the number of neurons, i.e. nodes in the network, is determined on the fly. Then they merge the two sets of neurons and construct the induced Delauny triangulation, that is they determine the nearest and second nearest neighbors of each training sample and join these two neurons with an edge. The final reduced training set is determined by the neurons that joined samples that belonged to the two different classes and the samples that has these neurons as nearest neighbors.

1.2 Hubness in high dimensional spaces

Radovanovic et al (2009) have introduced the hubness phenomenon, which is another aspect of the curse of dimensionality.

It has already been stated that the curse of dimensionality refers to the difficulties one has to confront with when working with high dimensional spaces. One aspect of the curse of dimensionality is *distance concentration.* This refers to the tendency of distances between all pairs of points in high-dimensional data to become almost equal.

Another aspect is related to the distribution of *k-occurrences,* that is the number of times a point appears among the k nearest neighbors of other points in a data set. It seems that, under certain conditions, as dimension increases this distribution becomes considerably skewed to the right, resulting in the emergence of *hubs*, that is points that appear more often in the nearest neighbors lists of the points in the data set. Figure 1.7 shows plots of the distribution of 5-occurrences for Euclidean, $l_{0.5}$, and cosine distances. The data sets were generated having attributes independent and identically distributed (i.i.d) according to the uniform $[0, 1]$ distribution and dimensions three, twenty and one hundred. One can see that while in three dimensions the distribution is consistent with the binomial distribution, in higher dimensions the distributions become skewed to the right, which means that hubs start to appear.

They based the explanation of the appearance of hubs on earlier theoretical results, according to which high dimensional points are approximately lying on a hypersphere centered at the data set mean. Other earlier works suggested that the distribution of distances to the data set mean has a non negligible variance for any finite dimension. Hence, the existence of a non-negligible number of points closer to the data set mean is expected in high dimensions. These points, by being closer to the mean, tend to be closer to all other points – a tendency which is amplified (in relative terms) by high dimensionality, making points closer to the data set mean have increased inclusion probability into k-NN lists, even for small values of k.

The hubness phenomenon has been studied in real life data sets too. They (Radovanovic et al, 2009) found that hubness appears in real life data sets too, and that it is strongly correlated with two factors:

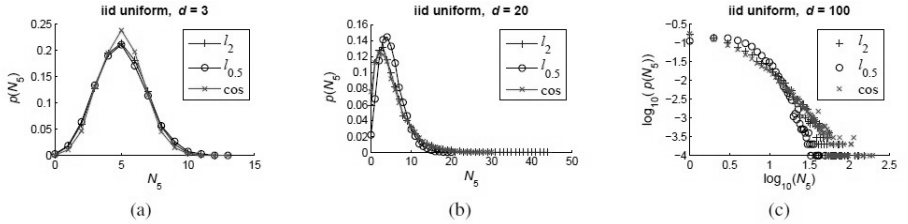


Figure 1.7: Distribution of 5-occurrences for Euclidean, $l_{0.5}$, and cosine distances for iid uniform random data sets with dimensionality (a) $d = 3$, (b) $d = 20$ and (c) $d = 100$ (log-log plot)

- Intrinsic dimensionality[1]. Skewness of the k-occurrences distribution depends on the intrinsic and not the embedding dimensionality of the data set.
- Clustering. Real life hubs are closer than other points to their respective cluster centers.

The whole discussion about hubs so far did not take into account the classes the samples belong to. Taking into account samples class one can define *bad* and *good* hubs. First the number of *bad k-occurrences,* or *bad hubness score*, of a sample $\vec{x}$ is the number of points from the data set for which $\vec{x}$ is among the k nearest neighbors and the classes of $\vec{x}$ and these points do not mach. Respectively the number of *good k-occurrences,* or *good hubness score*, of a sample $\vec{x}$ is the number of points from the data set for which $\vec{x}$ is among the k nearest neighbors and the classes of $\vec{x}$ and these points do mach. Naturally for every sample $\vec{x}$: $N_k(\vec{x}) = BN_k(\vec{x}) + GN_k(\vec{x})$, where $N_k(\vec{x})$ is the sample's k-occurrences value and $BN_k(\vec{x})$, $GN_k(\vec{x})$ are its bad and good k-occurrences values.

Bad hubs are samples with high bad k-occurrences value and *good hubs* are samples with high good k-occurrences value.

Radovanovic et al. studied bad hubs in real life data sets and in several machine learning contexts. They showed that bad hubs influence classification (k-NN classifier, AdaBoost) and clustering algorithms.

In this work we take a deeper look at the influence bad hubs can have in SVMs.

1.3 Goals and contributions

The main aim of this work is to explore how hubs influence support vector machines. Do bad hubs affect the separating hyperplane of a SVM? If this is

[1]Embedding dimensionality of a data set is the number of attributes the samples have, while intrinsic dimensionality is the dimensionality of the manifold defined by the data set. In other words intrinsic dimensionality is the number of attributes that suffice to describe the data set.

the case then bad hubs should be support vectors. And this in turn means that they are located inside the margin defined by the hyperplane and the data set. Questions regarding bad hubs and support vectors are discussed in chapter 2.

A main goal of this work is to determine the differences between low and high dimensionality. This is natural to consider since hubness is a side effect of high dimensionality. Throughout this work we compare low to high dimensionality.

Bad hubs, as will be shown, are support vectors, but do they suffice in order to generate a good decision boundary generated by a SVM trained on them or do they behave more like noise? And if the latter is the case what hubness score samples are necessary to determine a good decision boundary? These questions will lead to the definition of hubness ratio, which is discussed in chapter 3.

Finally we would like to verify our theoretical results. We do this by introducing instance selection methods that are the results of the conclusions established that far. The accuracy of these methods, if it is above random, would give base ground to our theory about hubs and support vector machines. Instance selection methods and the experiments comparing them to the random method are discussed in chapters 4 and 5.

For a summary of our main contribution see chapter 6.

1.4 Data sets

In this section we describe the data sets used throughout this work. Two kind of data sets are being used. Synthetic, that is artificially generated, and real life.

Synthetic data are generated having attributes independent and identically distributed (i.i.d) according to the standard normal distribution. Unless otherwise stated two hundred samples for each class are generated. A constant value is added to the attributes of the second class, defining the *separation* of the two classes. *Dimensionality* is defined as the number of attributes the samples have.

That is, if $s_0 = (s_{01}, s_{02}, \ldots, s_{0d})$ is a sample from the first class then

$$s_{0i} \sim \mathcal{N}(0,1)\,,\ i = 1, \ldots, d \tag{1.3}$$

d is the dimensionality of the space the samples lie in and $\mathcal{N}(0,1)$ is the standard normal distribution.

Respectively if $s_1 = (s_{11}, s_{12}, \ldots, s_{id})$ is a sample from the second class then

$$s_{1i} \sim \mathcal{N}(0,1) + sep,\ i = 1, \ldots, d \tag{1.4}$$

where sep is the separation and d is the dimensionality.

Samples generated according to the procedure described are distributed according to a multivariate normal distribution and the separation added to the attributes of the second class samples affects only the mean of the distribution.

We consider dimensions two, ten, twenty, fifty and one hundred.

The separation varies from *zero*, where the distance between the two class means is zero, to *large* where the two classes are linearly separable. We divided

the distance between large and zero separation to four equal parts, producing separations *small, medium* and *bigger*. The separation value at which the classes become linearly separable is dimensional dependent and because of this fact so are the other three values. This is why we use ordinal instead of arithmetic values for the separation. Table 1.1 shows the large separation values and the corresponding distances between the class means.

dimension	min required separation	distance between class means
2	4	5.6569
10	1.8	5.6921
20	1.1	4.9193
50	0.6	4.2426
100	0.35	3.5000

Table 1.1: Min required separation is the minimum separation that needs to be added to the attributes of the second class, so that if samples for two classes are generated with attributes following equations 1.3 and 1.4 the classes become linearly separable. Distance between class means is the Euclidean distance of the means of the distributions of the two classes that corresponds to the minimum required separation.

For real life data sets we used the ones that are described in table 1.2 (they are from the UCI Machine Learning Repository http://http://archive.ics.uci.edu/ml/) . The first column displays the name of the data set. The second column shows the total number of samples that are available, while columns three and four show the number of samples belonging to each class. The next column (fifth) shows the number of attributes each sample has, in other words the dimensionality. The last column shows the intrinsic dimensionality of the data set as calculated by the maximum likelihood estimator ([9]).

For each real life data set we found what is the kernel of a SVM that achieves the best accuracy over a ten-fold cross validation. Linear, polynomial and RBF kernels were used in this query. We tested discrete values for the C parameter, the RBF sigma parameter of the RBF kernel and the polynomial order of the polynomial kernel. The C parameter ranged through the values {0.01, 0.1, 1, 10, 100}, the RBF sigma parameter through the values {0.01, 0.1, 0.5, 1, 2, 5, 10} and the polynomial order through the values {2, 3, 5, 7, 10}. Table 1.3 shows what is the kernel, and its parameters, that achieves the best accuracy for each data set.

Data set	Samples	Class 0	Class 1	Dim	Intrinsic Dim
diabetes	768	500	268	8	6
ionosphere	351	126	225	34	13.57
musk1	476	269	207	166	6.74
parkinsons	195	48	147	22	4.36
sonar	208	97	111	60	9.67
spectf	266	55	211	44	13.83
wdbc	569	357	212	30	8.26
wpbc	198	151	47	33	8.69
australian	690	383	307	14	5.28
breast-cancer	683	444	239	10	5.63
fourclass	862	555	307	2	1.88
german.numer	1000	700	300	24	7.92
heart	270	150	120	13	6.56
liver-disorders	345	200	145	6	5.45
splice	1000	483	517	60	27.34
svmguide1	3089	1089	2000	4	3.60
spambase	4601	2788	1813	57	11.45

Table 1.2: Real life data sets

Data set	C	Kernel
diabetes	1	polynomial $n = 3$
ionosphere	10	rbf $\sigma = 2$
musk1	10	rbf $\sigma = 10$
parkinsons	10	rbf $\sigma = 2$
sonar	10	rbf $\sigma = 5$
spectf	10	rbf $\sigma = 5$
wdbc	0.1	linear
wpbc	10	rbf $\sigma = 5$
australian	10	polynomial $n = 2$

(a)

Dataset	C	Kernel
breast-cancer	0.1	linear
fourclass	1	rbf $\sigma = 0.1$
german.numer	1	polynomial $n = 2$
heart	0.01	linear
liver-disorders	10	polynomial $n = 2$
splice	10	rbf $\sigma = 5$
svmguide1	1	polynomial $n = 5$
spambase	100	rbf $\sigma = 10$

(b)

Table 1.3: The SVM kernel, and its parameters, that achieves the best accuracy over ten-fold cross validation for each real life data set.

Chapter 2

Bad Hubs

In this chapter we define bad hubs, identify the factors that affect their score and clarify their relationship to SVMs.

2.1 Defining bad hubs

Using synthetic data we explore when do bad hubs start to exist and how strong is their hubness, i.e. arithmetic value of bad hubness score (Bad and good hubness scores are defined in section 1.2).

The two factors, dimensionality and separation as defined in the previous chapter, affect the bad hubness phenomenon.

Let us consider the case where separation is kept constant and low, so that in effect there is only one distribution generating samples for both classes. We know from previous works ([13]) that in low dimensions there are no hubs, so we expect that there will also be no bad hubs. As dimensionality increases hubs start to exist and bad hubs will also appear.

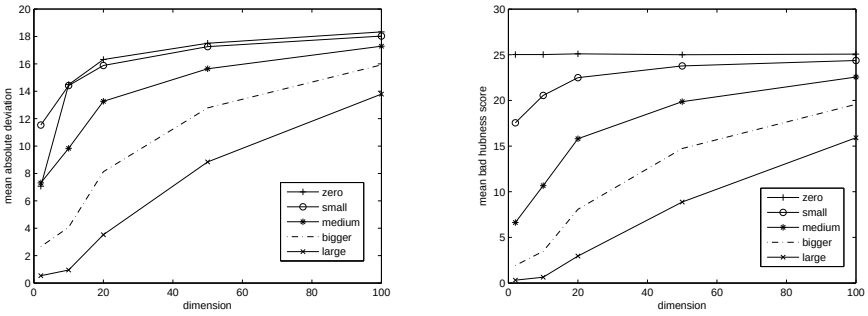
On the contrary if dimension is kept constant bad hubs will be quite strong (have big bad hubness score) as long as the two classes are near enough, separation is small, and weaken as separation increases. On the extreme case where separation is too big there will only only be samples with zero bad hubness score.

In order to visualize the previous thoughts we calculated the mean absolute deviation[1] (MAD) and mean of bad hubness distribution for the five different dimensionality and separation values. The results can be seen in figure 2.1. The values used for drawing the lines are the means of thirty repeats of the experiment. Fifty neighbors were considered for calculating bad hubness scores.

The following observations can be made by looking at the plots:

1. Both the mean and MAD lines are monotonically increasing. This means

[1] MAD is a robust measure of the variability of univariate samples. For a univariate data set $x_1, x_2, \ldots, x_n$ MAD is equal to $mean_i\left(|x_i - mean_j(x_j)|\right)$



(a) Mean absolute deviation of bad hubs distribution.

(b) Mean of bad hubs distribution

Figure 2.1: Mean absolute deviation and mean of bad hubness distribution versus dimension for five different separation values.

that keeping the separation constant, not in absolute value but in a relative way, and increasing dimensionality produces stronger bad hubs.

2. The values of large separation lines for low dimensions (two, ten) is very small, indicating that in order for the two classes to be linearly separable their means must be taken away apart so that only very few samples with non zero bad hubness score exist. This does not seem to be the case for higher dimensions, where a small distance of the classes means is sufficient to linearly separate the two classes. At these distances strong bad hubs exists.

3. For some constant dimension as separation increases bad hubs get fewer and weaker.

4. In dimension two, even though the mean bad hubness score for zero separation is bigger than that of other separation values, MAD is smaller. This means that at zero separation all samples have equally big bad hubness scores, so in effect there are no bad hubs.

5. The difference between the maximum and minimum MAD (mean) value for some constant dimension is getting smaller as dimension increases. This is another justification for the fact pointed at 2, that since the distance needed to separate two classes is getting smaller as dimension increases bad hubs are more persistent, meaning that they keep on existing even when the two classes are linearly separable.

Another defining property of bad hubs is their distance to the opposite class mean. It seems natural to think that the greater the bad hubness score the closest the sample is to the mean of the opposite class. To quantify this idea we

	10 neigh			50 neigh		
	zero	medium	large	zero	medium	large
dim 2	-0.15	-0.77	-0.14	-0.75	-0.92	-0.55
dim 100	-0.90	-0.89	-0.84	-0.98	-0.98	-0.97

Table 2.1: Correlation between bad hubness score and distance to opposite class mean for dimensionalities two and one hundred. Ten and fifty neighbors were considered for calculating hubness scores and for each number of neighbors zero, medium and large separations were considered.

calculated the Spearman [2]correlation between bad hubness score and distance to opposite class mean. Table 2.1 shows the results. From the figure we can conclude that in high dimensionality there is always a strong correlation between bad hubness and distance to opposite class mean. This does not seem to be the case in low dimensionality, where the correlation strongly depends on separation and the number of neighbors used for calculating hubness scores. So **in high dimensionality stronger bad hubs are always closer to the opposite class mean.**

2.2 Location of bad hubs relatively to support vectors

In this section we will look into how the property of a sample being a bad hub relates to the property of being a support vector. We will do so for both synthetic and real life data sets.

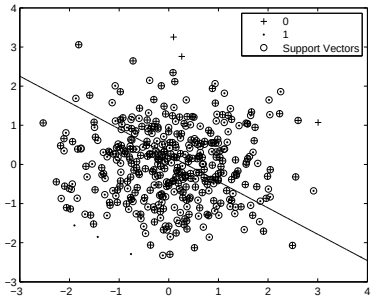
2.2.1 Synthetic data sets

Synthetic data were generated and a linear SVM was trained on these data. Several dimensionalities and for each dimensionality several separations were examined. We will start with low dimensions where visualizations and intuition are possible and proceed to higher dimensions. We will examine how samples' distance to separating hyperplane develops as separation grows. To that end we plot bad hubness score versus distance to hyperplane for all samples.

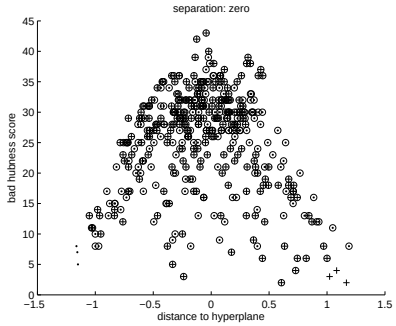
Figures 2.2, 2.3 show the results of the experiment for dimension two. Samples of one class are marked with a plus (+) sign while the samples of the other class are marked with a point (.). Support vectors are marked with a circle.

At zero separation almost all samples are support vectors with the majority of them having big bad hubness scores. Bad hubs of any score can be found

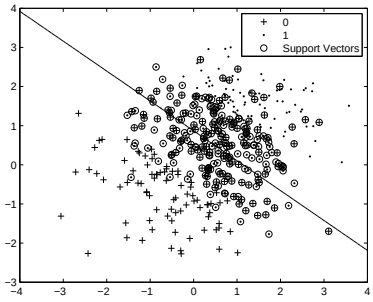
[2]Spearman correlation is a non-parametric measure of statistical dependence between two variables. . It assesses how well the relationship between two variables can be described using a monotonic function. It varies from -1 to $+1$. Values close to -1 indicate that large values of one variable are associated with small values of the other variable, values close to zero indicate no dependence between the values of the two variables, while values close to $+1$ indicate that large values of one variable are associated with large values of the other variable.



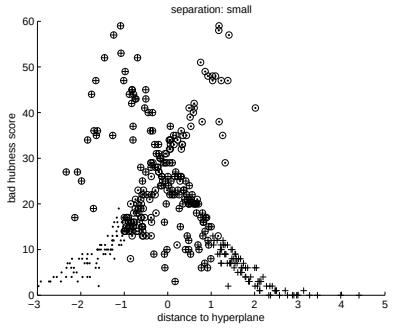
(a) SVM separation zero



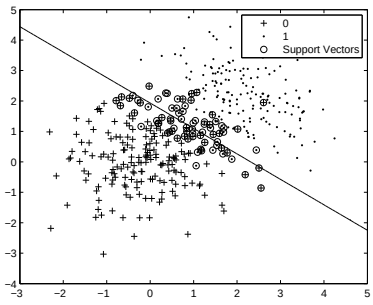
(b) Bad hubs vs distance to plane, separation zero



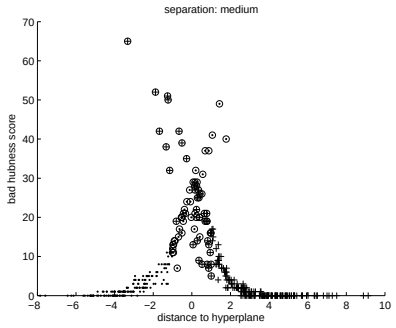
(c) SVM separation small



(d) Bad hubs vs distance to plane, separation small

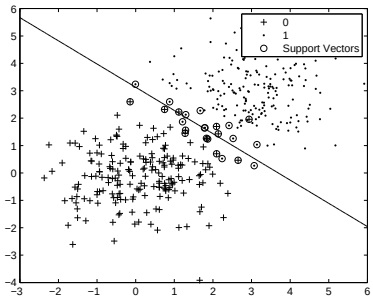


(e) SVM separation medium

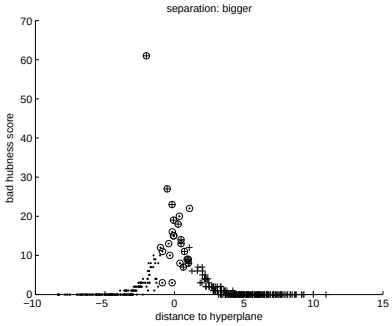


(f) Bad hubs vs distance to plane, separation medium

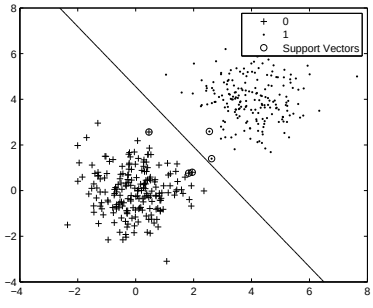
Figure 2.2: Dimension two. SVMs trained on classes with different separations. Besides each SVM there is the corresponding bad hubness to distance to hyperplane plot. Plus (+) and dot (.) are the signs of the two classes. Support vector are marked with a circle.



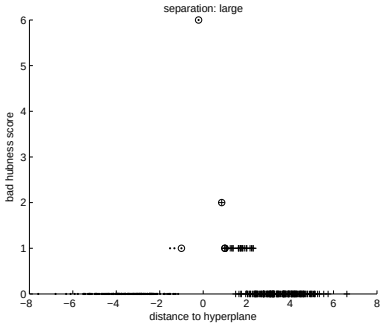
(a) SVM separation bigger



(b) Bad hubs vs distance to plane, separation bigger



(c) SVM separation large



(d) Bad hubs vs distance to plane, separation large

Figure 2.3: Dimension two. SVMs trained on classes with different separations. Besides each SVM there is the corresponding bad hubness to distance to hyperplane plot. Plus (+) and dot (.) are the signs of the two classes. Support vector are marked with a circle.

to be support vectors and at any distance from the hyperplane. As separation increases support vectors with small bad hubness score cease to exist and top bad hubs are located at the opposite side of their class near the mean of the opposite class (hence the big bad hubness score). This can be easily seen at medium separation (figures 2.2e, 2.2f) where the plus class sample at the upper right corner of the plot behaves like an outlier for that class. At larger separations top bad hubs define the boundary of the two classes and are located at (-1,1) distance range from the hyperplane. In all of these cases the ± 1 margin around the hyperplane is determined by medium bad hubness score samples. Only when the separation is large enough so that the two classes are well separated top bad hubs seem to be located at the ± 1 margin.

When dimension increases (figures 2.4, 2.5) illustrate how bad hubs behave as support vectors for dimension twenty) the first interesting thing one can note is the existence of bad hubs from zero separation. While in dimension two (zero separation) all samples had equally big bad hubness scores, in higher dimensions bad hubs get produced, i.e. samples with bigger bad hubness score than the rest of the samples. As in lower dimensions top bad hubs are at (-1,1) distance range from the hyperplane, while weaker bad hubs can be found at any distance from the hyperplane. Samples that are not support vectors have small bad hubness scores and are located far away the hyperplane.

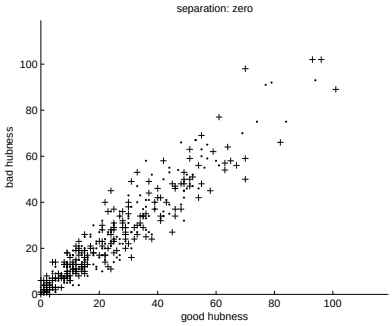
Another interesting thing that can be noticed is that a strong correlation between bad hubness score and good hubness score starts to exist. In lower dimension this was true only for zero separation but for higher dimensions the correlation holds even for medium separations. An explanation for this can be found in section 2.4.

As separation increases top bad hubness as well as low bad hubness support vectors are located at (-1,1) distance range from the hyperplane. Due to the quick separation of the two classes, even for small absolute values of separation, it is difficult to find top bad hubs that behave like outliers being near the opposite class mean. This is also the reason why there are no support vectors with low bad hubness score far away the hyperplane. In order for that samples to exist they should be located at the edges (if they should have small bad hubness score) of the opposite (if they should be support vector) class distribution. But this becomes too difficult to happen as the two classes move quickly far apart.

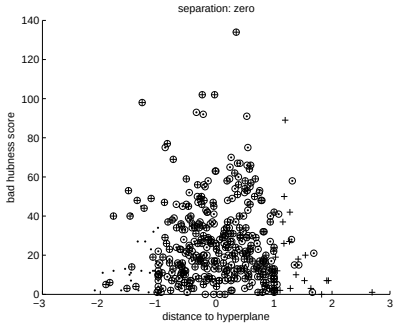
Again the medium sized bad hubs determine the ± 1 margin of the hyperplane, except for the case where the two classes are well separated. In the last case top bad hubs are located at distances ± 1 from the hyperplane.

A few remarks can be noted when dimension increases even more (see figures 2.6, 2.7, for dimension 100). First of all the correlation between bad and good hubness scores is more stronger and persists even at large separations. This means that bad and good hubs become the same.

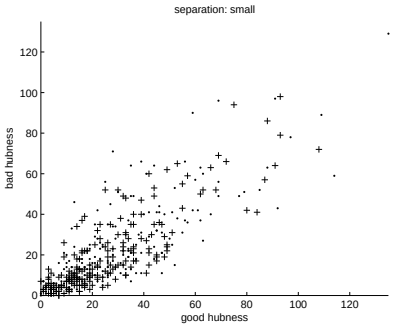
A side effect of this fact is that for the first time we start to notice top bad hubs that are not support vectors, that is they are located in their class side of the hyperplane and yet they are among the most favor neighbors of the samples of the other class. These bad hubs should be good hubs but because bad and good hubness are the same they behave like bad hubs as well. As is logical to



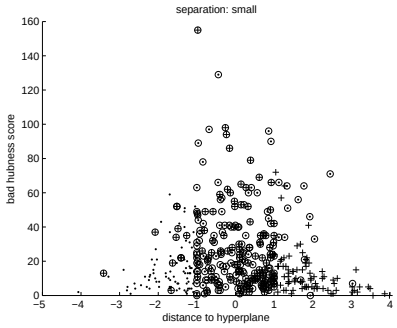
(a) Bad vs good hubness, separation zero



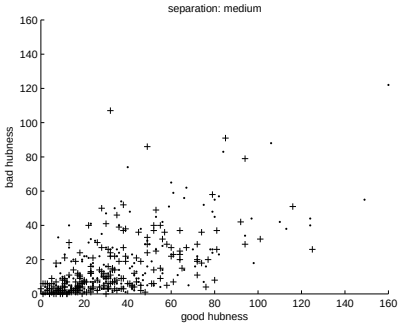
(b) Bad hubness vs dist to plane, separation zero



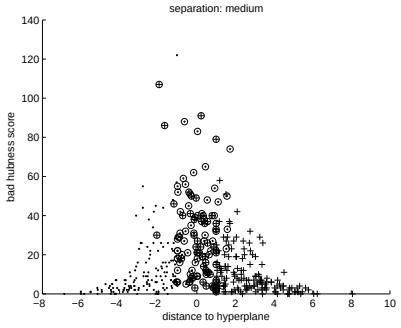
(c) Bad vs good hubness, separation small



(d) Bad hubness vs dist to plane, separation small

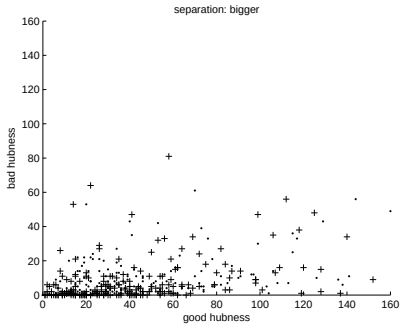


(e) Bad vs good hubness, separation medium

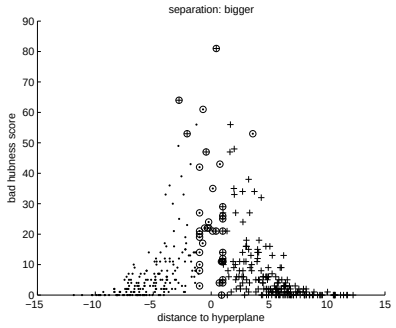


(f) Bad hubness vs dist to plane, separation medium

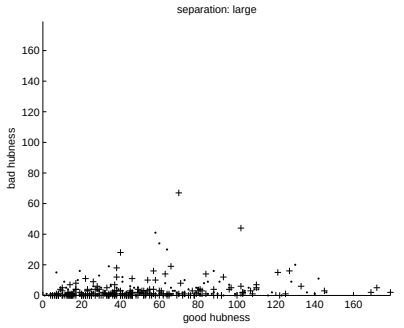
Figure 2.4: Dimension 20. Evolution of bad hubness vs good hubness and distance to hyperplane as separation increases



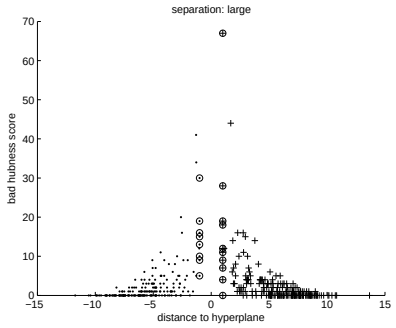
(a) Bad vs good hubness, separation bigger



(b) Bad hubness vs dist to plane, separation bigger

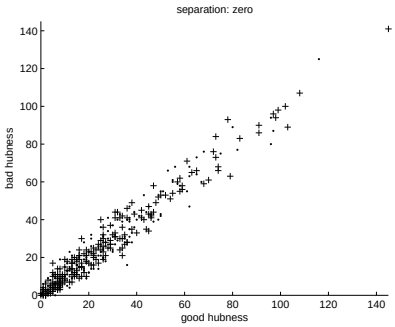


(c) Bad vs good hubness, separation large

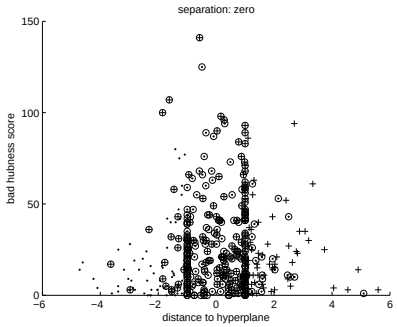


(d) Bad hubness vs dist to plane, separation large

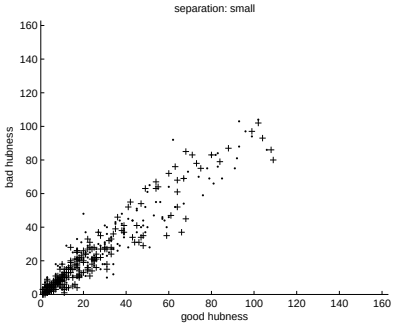
Figure 2.5: Dimension 20. Evolution of bad hubness vs good hubness and distance to hyperplane as separation increases.



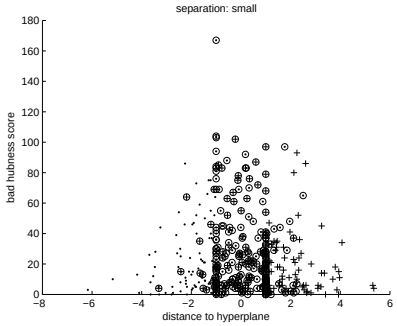
(a) Bad vs good hubness, separation zero



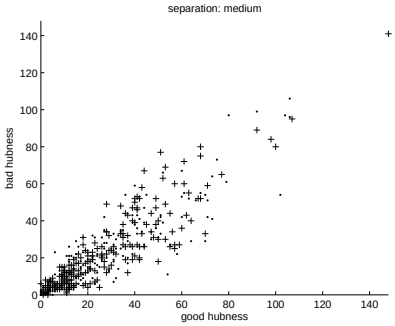
(b) Bad hubness vs dist to plane, separation zero



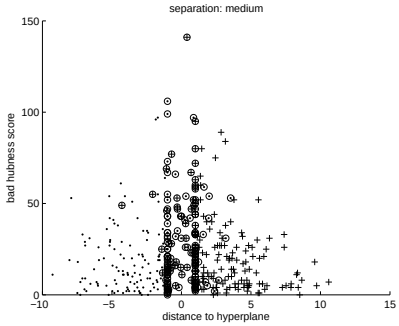
(c) Bad vs good hubness, separation small



(d) Bad hubness vs dist to plane, separation small

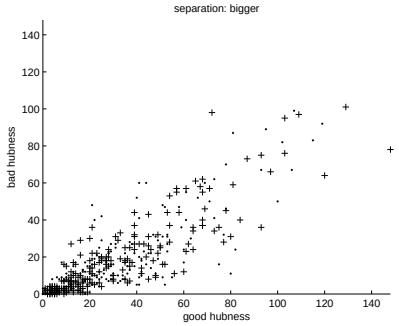


(e) Bad vs good hubness, separation medium

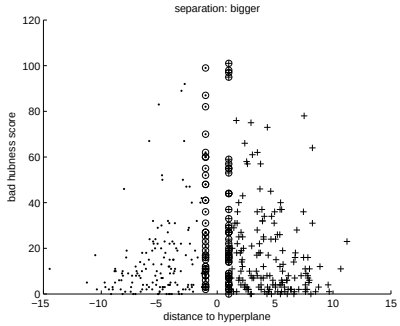


(f) Bad hubness vs dist to plane, separation medium

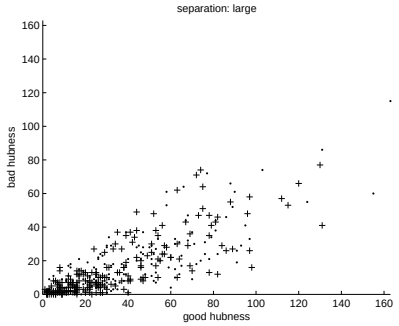
Figure 2.6: Dimension 100. Evolution of bad hubs as support vectors



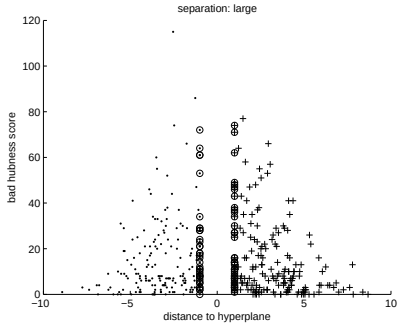
(a) Bad vs good hubness, separation bigger



(b) Bad hubness vs dist to plane, separation bigger



(c) Bad vs good hubness, separation large



(d) Bad hubness vs dist to plane, separation large

Figure 2.7: Dimension 100. Evolution of bad hubs as support vectors

expect this phenomenon lowers the probability of a bad hub being a support vector. Probabilities of bad hubs being support vectors will be discussed in more detail in section 2.3. For an explanation of the strong correlation between bad and good hubness in high dimensionality see section 2.4.

Another consequence of the bad and good hubness correlation is the fact that top bad hubs do not determine the ± 1 margins of the hyperplane even for large separations.

The remarks done for the twenty dimensions retain as dimensions grow with the notice that the separation of the class is even more quicker. This can be seen from figure 2.7b (bigger separation for one hundred dimension) where the two classes are already almost linearly separable.

2.2.2 Real life data sets

Here we verify the above results in real life data sets.

Figures 2.8 and 2.9 show plots of bad hubness versus good hubness and distance to hyperplane for several real life data sets.

German numer (figures 2.8a, 2.8b) and spambase (figures 2.8c,2.8d) data sets exhibit typical behavior. Most of the top bad hubs are in distance range (-1,1) from the hyperplane, while there are bad hubs that are located well inside the other class's region.

This is most apparent in heart (figures 2.8e, 2.8f) and especially in breast cancer (figures 2.9a, 2.9b) data set. In the last one, one can see well defined bad hubs of the plus marked class. This bad hubs behave as outliers being well inside the opposite class.

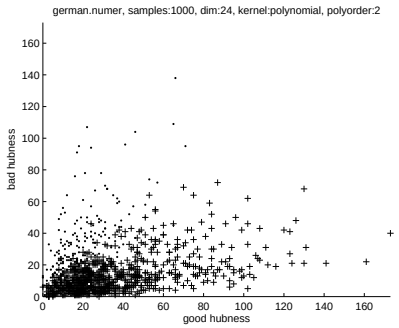
Finally splice (figures 2.9c, 2.9d) data set shows a situation where due to the large dimensionality bad hubness and good hubness is almost the same which produces bad hubs that are not support vectors, that is they are located inside their class region.

2.3 Bad hubs tend to be support vectors

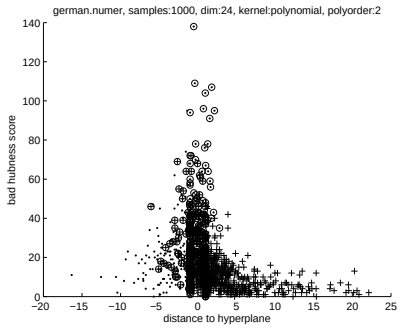
In this section we verify, in a statistical way, that bad hubs tend to be support vectors. We use the Wilcoxon[3] rank sum test, which is equivalent to Mann-Whitney U test, in order to test whether there is a difference in the bad hubness scores (calculated for fifty neighbors) of support vectors and non support vectors. We calculated the p-value of the statistic for the two groups: bad hubness scores of support vectors and bad hubness scores of non support vectors. If the p-value is less than 0.05 then we can reject the null hypothesis at 95% significance level that the bad hubness scores of the two groups are equally big for the alternative hypothesis that there is difference in the bad hubness scores.

We run the test for both synthetic and real life data sets.

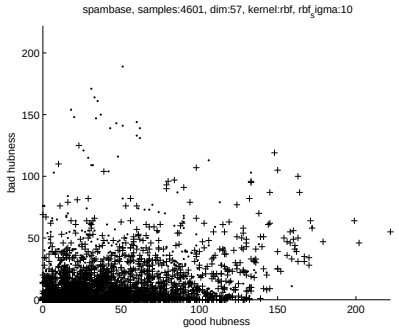
[3]The Wilcoxon rank sum test or equivalently the Mann-Whitney U test is a non parametric statistical test for assessing whether two independent samples of observations have equally large values.



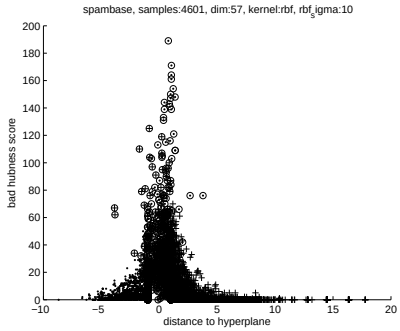
(a) german-numer



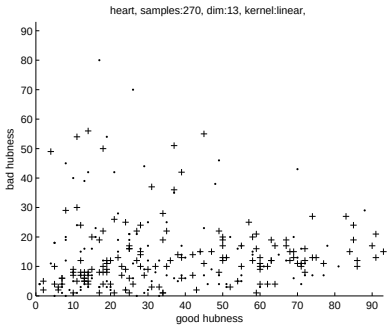
(b) german-numer



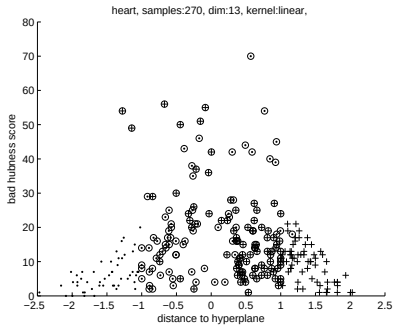
(c) spambase



(d) spambase

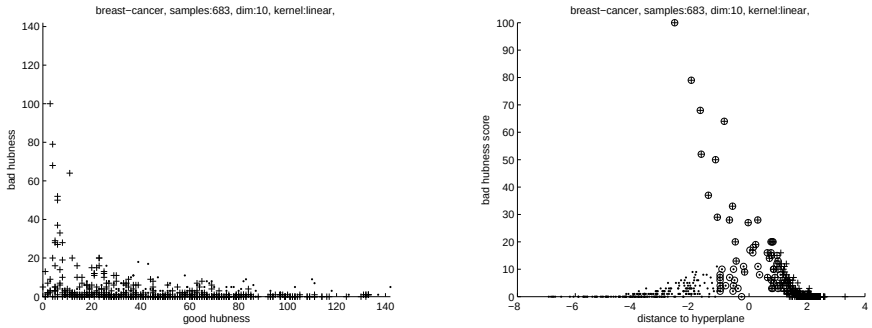


(e) heart



(f) heart

Figure 2.8: Bad hubness vs good hubness and bad hubness vs distance to hyperplane for several real life data sets



(a) breast cancer

(b) breast cancer

(c) splice

(d) splice

Figure 2.9: Bad hubness vs good hubness and bad hubness vs distance to hyperplane for several real life data sets

2.3.1 Synthetic data sets

Figure 2.10 shows the results of the experiment for synthetic data sets. As expected at high separations the very small p-values indicate that there are a few support vectors with high bad hubness scores, while the majority of samples are not support vectors having low bad hubness scores. Nevertheless we observe that the p-value for high dimensionalities is bigger than the p-value for low dimensionalities. This happens because as already pointed out in section 2.5 in high dimensionality bad hubs exist that are not support vectors due to the strong correlation between bad and good hubness scores. So in low dimensionality all top bad hubs are support vectors while in high dimensionality not all top bad hubs are support vectors. The latter fact is the reason for the increased p-value in high dimensionality.

At low dimensionality and small separation values the p-value remains very small. In these cases the majority of the samples are support vectors and the few samples that are not have small bad hubness scores. It is the inverse of what is happening at high separation values.

At high dimensionality and small separation values the p-value is big. The explanation again is that in high dimensionality there are bad hubs that are not support vectors. So at small separation values almost all samples are support vectors but while at low dimensionality the few samples that are not support vectors have small bad hubness scores, at high dimensionality the few samples that are not support vector can have any large value. This results in big p-values.

For an explanation of the strong correlation between bad and good hubness in high dimensionality see section 2.4. Briefly this happens because in such high dimensionalities the distance from the samples to their class mean is a lot bigger than the distance between the two class means. So a sample being closer to its class mean, having thus big good hubness score, is also closer to the opposite class mean, having thus big bad hubness score.

Since the statistical test provides confidence at 95% significance level that there is difference in the bad hubness scores of support vectors and non support vectors in almost all the cases, we can come to the conclusion that bad hubs tend to be support vectors.

2.3.2 Real life data sets

We run the same statistical test, which assesses that there is a significant difference in the bad hubness scores of support vectors and non support vectors, to the real life data sets. Again fifty neighbors were used to calculating hubness scores, except for the parkinsons data set because the number of samples for one class is only forty eight. In that data set we used twenty neighbors in order to calculate hubness scores.

Table 2.2 shows the results of the experiment. All the values are as expected except for two data sets: fourclass and sonar. We explain why this is happening.

The fourclass data set is a two dimensional problem and since there are no hubs in low dimensions there should be no pattern in the places of support

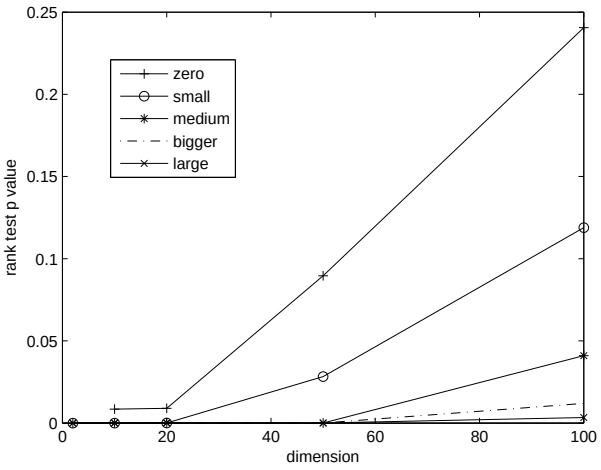


Figure 2.10: P-values of the Wilcoxon rank sum test involving the hypotheses H_0 : bad hubness scores of support vectors are equally big to bad hubness scores of non support vectors against the alternative H_1 : there is significant difference in the bad hubness scores of support vectors and non support vectors.

vectors with respect to bad hubness score.

For the sonar data set we note that there is a strong correlation between bad and good hubness scores. This results in many bad hubs not being support vector. We remind that the explanation for this phenomenon will be given in section 2.4.

2.4 Bad and good hubs in high dimensionality

In this section we understand why there is a strong correlation between bad and good hubness in high dimensions.

For a given dimensionality and arithmetic value of separation we calculate the *relative separation* as

$$\text{relative separation} = \frac{mean(\text{distance of each sample from its class mean})}{\text{distance between class means}}$$

As figure 2.11 illustrates this is a stable measure of separation since it is independent of dimensionality.

The same figure also illustrates the relative separation for four distinct cases: separation small and large for dimensions two and one hundred. In one hundred dimensions even for the large separation, where the two classes are linearly separable, the distance of each sample from its class mean is twice as bis as the distance between the two class means. So we can imagine that in effect there is only one class mean generating samples for both class, therefore samples from

Data set	p-value
diabetes	0.0000
ionosphere	0.0000
musk1	0.0000
parkinsons*	0.0003
sonar	0.4407
spectf	0.0000
wdbc	0.0000
wpbc	0.0036
australian	0.0000

(a)

Data set	p-value
breast-cancer	0.0000
fourclass	0.6824
german.numer	0.0000
heart	0.0000
liver-disorders	0.0000
splice	0.0014
svmguide1	0.0000
spambase	0.0000

(b)

Table 2.2: P-values of Wilcoxon rank sum test for real life data. The test assesses that there is a significant difference in the bad hubness scores of support vectors and non support vectors. (*) 20 neighbors were used for calculating hubness scores instead of fifty. This has been done because the number of samples for one class is only forty eight.

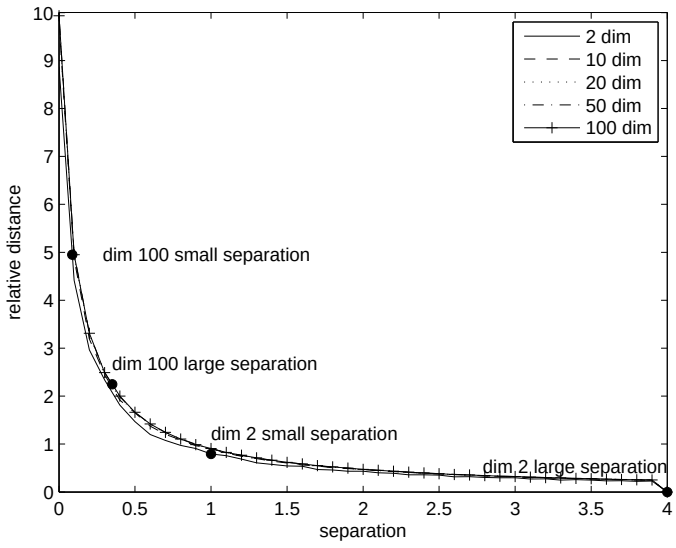


Figure 2.11: Relative distance vs separation. Figure shows that in high dimensions the mean distance of the samples from their class mean is a lot bigger than the distance between the class means. This results in strong correlation between bad and good hubness. This correlation does not exist in low dimensions.

data set	dim	intrinsic dim	corr
splice	60	27.34	0.87
sonar	60	9.67	0.73
ionosphere	34	13.57	0.74

(a)

data set	dim	intrinsic dim	corr
fourclass	2	1.88	-0.41
svmguide1	4	3.60	-0.31
australian	14	5.28	0.18

(b)

Table 2.3: Spearman correlations between bad and good hubness scores for real life data sets. For each data set the dimensionality and intrinsic dimensionality is also shown.

any class will be close to samples of both classes and have big bad and good hubness scores.

On the other hand in two dimensions the distance of the samples from their class mean is smaller than the distance between the two class means, so there could be no correlation between bad and good hubness.

2.4.1 Real life data sets

We calculated the Spearman correlations between good and bad hubness in real life data sets, in order to verify that there exists strong correlation in high dimensions and not in low.

Table 2.3 shows the results. Except the data set name and the the calculated Spearman correlation, the dimensionality and the intrinsic dimensionality of each data set is shown. We can conclude from the table, that indeed in high dimensions there is strong correlation between bad and good hubness. This correlation is not apparent in low dimensions.

2.5 Conclusions

So far we have established quite a few remarkable facts that we summarize here:

- Bad hubs tend to be support vectors.
- Not all support vectors are bad hubs. There are support vectors whose bad hubness score is not among the largest values.
- In high dimensionality due to strong good and bad hubness correlation, bad hubs may not be support vectors.

- In low dimensionality bad hubs behave like an outlier for their class, being located well inside the opposite class.
- Top bad hubs do not determine the ± 1 margins of the hyperplane.

2.6 Epilogue and intro to next chapter

We have seen that bad hubs tend to be support vectors and that in high dimensions bad hubs are also good hubs. Good hubs, being amongst the nearest neighbors of many samples of their class, should do well in describing their class. So samples with equally big bad and good hubness should combine the benefits of both.

This is the starting point of defining *hubness ratio*.

Chapter 3

Hubness Ratio

So far we have established that bad hubs are important, one of the reasons being is the fact that they tend to be support vectors. Good hubs are also important because they are among the most popular neighbors of their class samples. So it is worth examining samples with equally big bad and good hubness scores.

In order to do so we define for every sample its *hubness ratio* as

$$\text{hubness ratio} = \frac{\text{bad hubness}}{\text{good hubness}}$$

and look for samples with hubness ratio close to one.

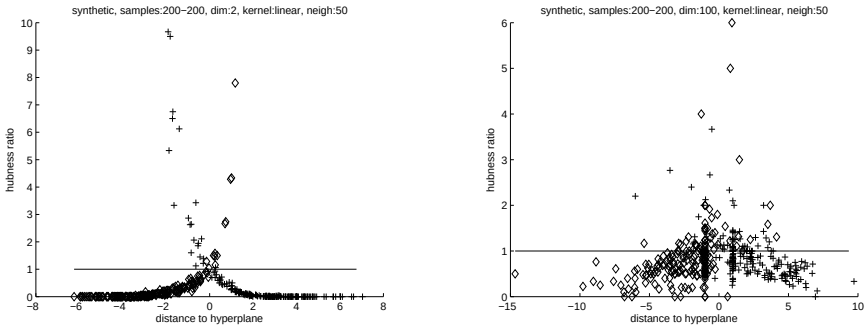
3.1 Samples with hubness ratio one

In this section we will show that (i) as dimension increases the percent of support vectors that are close to hubness ratio one increases too and (ii) that samples with hubness ratio close to one are closer to the hyperplane than the rest of the samples .

Figure 3.1 provides a starting point. It shows plots of hubness ratio versus distance to hyperplane for synthetic generated data with medium separation between the classes. A SVM with linear kernel was trained on these data. Fifty neighbors were considered for calculating hubness scores. From these plots we can come up with the ideas stated earlier.

We follow a more quantitative approach now. We consider six different cases for the notion of hubness ratio close to one. Samples with hubness ratio in the interval $[1 - n, 1 + n]$ $n = 0, 0.1, 0.2, 0.3, 0.4, 0.5$ are thought of as close and the rest as far.

In the first experiment we measure the percent of support vectors that are close to hubness ratio one. We run the experiment ten times and plotted the mean percent of support vectors that have hubness ratio close to one versus dimension for the six different close cases. Figure 3.2 shows the results. Monotonically increasing lines suggest that the percent of support vectors that are close to hubness ratio one increases as dimension increases.



(a) (b)

Figure 3.1: Hubness ratio vs distance to hyperplane for dimensions two and one hundred and medium separation between the classes

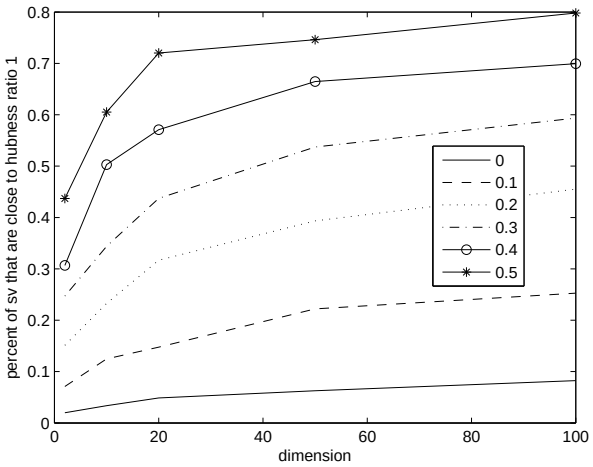
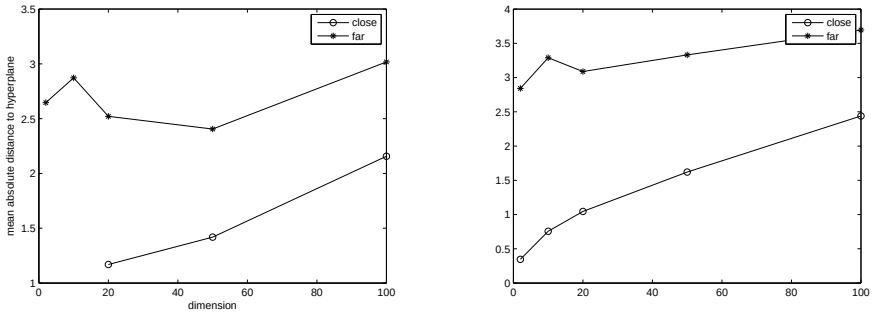


Figure 3.2: Fraction of support vectors that have hubness ratio close to one versus dimension for six different cases of close to hubness ratio one. Close are samples whose hubness ratio falls in the interval $[1 - n, 1 + n]$ $n = 0, 0.1, 0.2, 0.3, 0.4, 0.5$.



(a) $n = 0$ (b) $n = 0.5$

Figure 3.3: Mean absolute distance versus dimension for samples with hubness ratio close to one against samples with hubness ratio far from one. Close are samples with hubness ratio in the interval $[1 - n, 1 + n]$, far are the rest of the samples. Sub-figures show plots for two different values of n, 0 and 0.5.

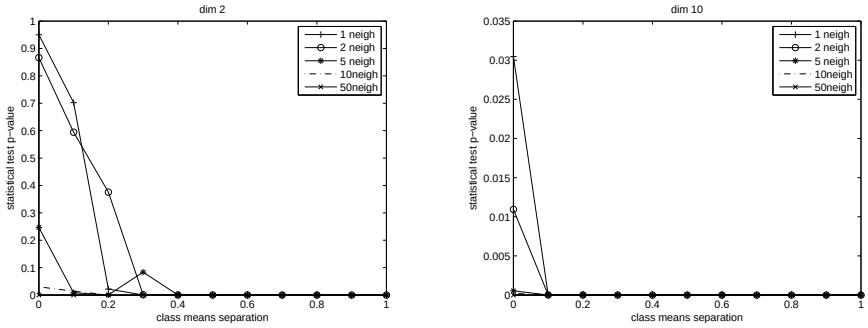
In the second experiment we calculated the mean distance to hyperplane of samples with hubness ratio close to one and compared it to the mean distance to hyperplane of samples with hubness ratio far from one. We conducted the experiment ten different times and plotted the means versus dimension for the six different close to hubness ratio one cases. Figure 3.3 shows the results. From it we can see that for all dimensionality values the mean distance to the hyperplane of samples that are close to one is smaller than the mean distance to hyperplane of samples that are far from one.

3.2 Hubness ratio of support vectors and non support vectors

In this section we will show that support vectors' hubness ratio is closer to one than non support vectors' hubness ratio and that this relation is stronger in high dimensions.

To that end we run the following statistical test: For constant dimension, separation between class centers and number of neighbors used to calculating hubness scores, we created ten different synthetic data sets with 200 samples per class. A SVM with linear kernel was trained on each data set and two lists were generated: one containing the support vectors of all the data sets and the other list contained the rest of the samples. A two sample Kolmogorov-Smirnov test [1]was run on the absolute value of the difference between the hubness ratio of the samples and the value of one. The null hypothesis was that the cdfs of

[1]The two sample Kolmogorov-Smirnov is a non parametric test that is used to assess whether the two samples come from the same distribution



(a) dimension 2 (b) dimension 10

Figure 3.4: Kolmogorov-Smirnov p-value test of the populations 1) absolute value of the difference of support vectors' hubness ratio to one and 2) absolute value of the difference of non support vectors' hubness ratio to one, versus separation for dimensions 2 and 10

the two populations are equal and was tested against the alternative that the cdf of the first population, corresponding to the absolute difference of support vectors' hubness score to one, is larger than the cdf of the second population (absolute difference of non support vectors' hubness score to one). This means that the values of the first population tend to be smaller than the values of the second population and this in turn means that support vectors' hubness score is closer to one than non support vectors' hubness score.

The above procedure was repeated for separation varying from 0 to 1 with step 0.1 and the number of neighbors taking the values 1, 2, 5, 10, 50. We plotted the statistical test p-value versus separation for the different values of the number of neighbors and dimensions 2 and 10. Results can be seen in figure 3.4. From the figure it can be seen that in high dimensions no matter what the separation between the class means is, the small p-value indicates that we can reject the null hypothesis and conclude that support vectors' hubness ratio is closer to one than non support vectors' hubness ratio. On the contrary in low dimensions the high p-values at small separations between class means reveal that we can not reject the null hypothesis, which states that the cdf of the absolute value of one minus support vectors' hubness ratio is the same as the cdf of the absolute value of one minus non support vectors' hubness ratio.

Having shown that support vectors' hubness ratio is closer to one than non support vectors' we look more closely at the distribution of hubness ratio of support vectors and non support vectors.

Figure 3.5 shows the median and mean absolute deviation (MAD) of the hubness ratio distribution for support vectors and non support vectors as a function of dimensionality, for two different values of the number of neighbors used for calculating hubness scores, fifty and one hundred and fifty. Two things

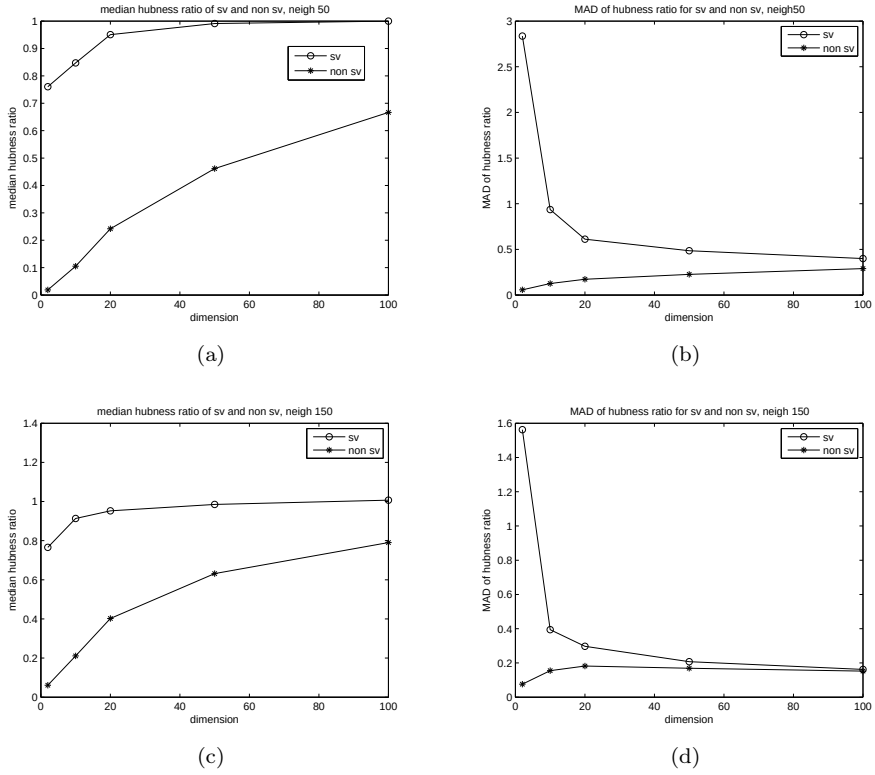


Figure 3.5: Median and MAD of hubness ratio distribution for support vectors and non support vectors.

can be noticed from the plots of the median: (i) it is always smaller than one and (ii) as dimensionality increases it is getting closer to one..

In the same figure and from the plots of MAD we can see that it is getting smaller as dimensionality increases. Especially there is a big drop in the MAD value when changing from dimension two to dimension ten. The smaller MAD value in high dimensions indicates that support vectors' hubness ratio is more concentrated around the value one.

3.2.1 Real life data sets

We repeat here the above statistical test to real life data sets in order to show that support vectors' hubness ratio is closer to one than non support vectors' hubness ratio.

In order to avoid displaying the p-value of the statistical test for distinct values of the number of neighbors we calculated the mean hubness ratio of each sample as the number of neighbors varied from one to the total number of

data set	p-value
diabetes	3.6e-007
ionosphere	2.1e-005
musk1	0.0012
parkinsons	0.4889
sonar	1.5e-004
spectf	6e-014
wdbc	4.6e-022
wpbc	0.0998

(a)

data set	p-value
australian	1.6e-007
breast cancer	4.3e-012
german numer	2.2e-012
heart	4.2e-019
liver disorders	0.2316
splice	1.4e-023
svmguide1	1.7e-018
spambase	3.5e-023

(b)

Table 3.1: Kolmogorov-Smirnov statistical test p-values for real life data sets.

data set	neighbors no	p-value
parkinsons	11	3.4e-004
wpbc	21	0.0080
liver disorders	31	0.0204

Table 3.2: Distinct values of the number of neighbors and p-values of the Kolmogorov-Smirnov statistical test for the 'bad' data sets of table 3.1.

samples of each data set minus one with step five. Then we run the statistical test over the mean hubness ratios.

Table 3.1 shows the results of the experiment. We can see that in all data sets except three the p-value is extremely small. For those bad data sets we can find distinct values of the number of neighbors where the p-value is small. Table 3.2 shows one such value for each one of these data sets.

3.3 Imbalanced classes

When in a classification problem there are many more instances of one class than the others then it is said that the classes are imbalanced. Imbalanced classes can exhibit a special behavior when calculating hubness ratios. By using a suitable number of neighbors for calculating hubness scores one can achieve to separate the samples of the two classes: samples of one class will have hubness ratio smaller than one and samples of the other class will have hubness ratio bigger than one.

Hubness ratio separation happens because classes are imbalanced and by considering larger and larger number of neighbors every sample of the imbalanced class will eventually get into a neighbor where the samples of the other class are more than the samples of its own class. In the extreme case where the total number of samples of both classes is used for calculating hubness scores, hubness ratios will be equal for the samples of the same class and it will be

dimension	separation small	separation medium	separation bigger
2	230	-	-
10	120	240	-
20	80	200	240
50	70	130	210
100	60	80	140

Table 3.3: Number of neighbors required to separate the hubness ratios of two classes.

equal to the ratio of the number of samples of the two classes.

Table 3.3 shows the number of neighbors 'usually' required in order to separate the hubness ratios of the two classes for separations small, medium and bigger. 'Usually' means that in at least three out of five calculations for the particular dimension and class separation the hubness ratios were separated. Multiples of ten were tried for the number of neighbors. The first class had 200 samples and the second 50 samples. A '-' in the table means that no hubness ratio separation can be achieved unless 249 neighbors (the total number of samples minus one) were used for calculating ratios.

From the table we can see that (i) the higher the dimension the less number of neighbors is required to separate the ratios and (ii) for constant dimension, the closer the two classes are, the less number of neighbors is required to separate the ratios. This is reasonable because since the class centers are close, in any given region of the space there will be samples from both classes with approximately the same analogy as the class analogy. Hence only a small number of neighbors is sufficient to separate the hubness ratios of the two classes.

Chapter 4

Methods

In this chapter we exploit the knowledge gained in the previous chapters and define two instance selection methods: *RatioOne* and *BelowOne*

Both methods take as input arguments *data*, *groups*, *sampleSize* and $neighborsNo$. *data* is a two dimensional array with rows corresponding to samples and columns corresponding to attributes. *groups* is a one dimensional array that contains the class each sample belongs to. The ith element of the *groups* array corresponds to the ith row of the *data* table. Its values are 0 for samples of the first class and 1 for samples of the second class. *sampleSize* is the number of samples the methods should select from the total number of samples. The $neighborsNo$ argument is used for calculating hubness scores.

Both methods return *sampleSize* rows from the *data* array, corresponding to the samples they have selected.

4.1 RatioOne

This method first sorts the samples in ascending order of their absolute difference of hubness ratio to the value of one. Then it selects the *sampleSize* samples that are closer to hubness ratio one keeping the class ratio in the selected samples equal to the class samples in all samples.

This method gives priority to samples with hubness ratio close to one, either smaller or bigger. The reason for defining this method is that as shown in section 3.1 the percent of support vectors that have hubness ratio close to one increases as dimensionality increases.

Algorithm 1 shows pseudocode of the method. *calcRatios*() function calculates for each sample its hubness ratio. $balancedFirst()$ function returns the first *sampleSize* elements from the $sortedIndx$ array so that class ratio in the selected elements is equal to class ratio in *groups* array.

The algorithm first calculates hubness ratios using function *calcRatios*() (line 1) and then the absolute distance of each sample's hubness ratio to the value of one (line 2). Then it uses the *sort* function which returns the permu-

tation of the indexes of the array passed as a first arguments that succeeds in sorting the array in order specified by the second argument (line 3). After wards using function $balancedFirst()$ it gets the first $sampleSize$ elements from the $sortedIndx$ array so that class ratio in the selected elements is equal to class ratio in *groups* array (line 4). Finally it uses the selected indexes to return the corresponding rows from the *data* array (line 5).

Algorithm 1: ratioOne

```
/* selects sampleSize samples from data that are closer to
   hubness ratio 1 keeping class ratio of selected data equal
   to class ratio in all data                                  */
```

Input: data, groups, neighborsNo, sampleSize
Output: selected data

```
1 ratios ← calcRatios(data, groups, neighborsNo);
2 distToOne ← absoluteValue(1-ratios);
3 sortedIndx ← sort(distToOne, 'ascend');
4 selectedIndx ← balancedFirst(groups, sortedIndx, sampleSize);
5 return data [selectedIndx, :]
```

4.2 BelowOne

BelowOne is the second method that we introduce. This method differs from the previous in that it gives priority to the samples whose hubness ratio is closer to one but only smaller than one. Only when there are no more samples with hubness ratio smaller than one, it considers those with hubness ratio bigger than one and in ascending order of their ratio.

There are good reasons for selecting samples with hubness ratio smaller than one. Firstly as shown in figure 3.5 in section 3.2 the median of the hubness ratio of the support vectors is smaller than one and of course closer to one than non support vectors. This means that there are more support vectors with hubness ratio smaller to one than bigger to one and so selecting first those with ratio smaller than one gives us better chances of finding more support vectors.

Secondly, samples with hubness ratio bigger than one have bigger bad hubness than good hubness score and so it is most likely that they are among the strongest bad hubs. Chapter 2 has shown that bad hubs can behave like noise being located at the wrong side of the hyperplane. These noisy samples can result in a misled generated by a SVM hyperplane so it would be best for an instance selection method not to choose these samples. It feels right that choosing samples that are located on their side of the decision boundary would give better results.

Finally, selecting samples with hubness ratio smaller than one, gives the opportunity to the method to select samples that are quite strong good hubs. These good hubs can be a good descriptor of their class shape.

In combination the last two reasons, suggest that samples with hubness ratio

smaller than one should provide good description of both the decision boundary and class shapes.

Algorithm 2 shows pseudocode of the method. First hubness ratios of all samples are calculated (line 1) and then differences of hubness ratios to the value of one (line 2). Then the samples whose ratio is smaller than one are sorted in descending order of their ratio, meaning that those which have ratio closer to one come first (line 3), and samples with ratio bigger than one are sorted in ascending order of their ratio, implying again that those with ratio closer to one come first (line 4). Then the two sorted arrays of indexes are concatenated into one array (line 5). The *concat* function concatenates two arrays without changing the order of the elements in the arrays and with the elements of the array passed as the first argument coming before the elements of the second array. Finally it selects the first *sampleSize* elements of the concatenated array so that the class ratio in samples that correspond to selected indexes is equal to class ratio in all samples (line 6) and returns the corresponding data (line 7).

Algorithm 2: belowOne

```
/* selects the sampleSize samples from data that are closer to
   hubness ratio one but only smaller than one.  Class ratio
   in selected data is equal to class ratio in data         */
```

Input: data, groups, neighborsNo, sampleSize
Output: selected data

```
1 ratios ← calcRatios(data, groups, neighborsNo);
2 distanceToOne ← ratios − 1;
3 belowOneSorted ← sort(distanceToOne ≤1, 'descend');
4 aboveOneSorted ← sort(distanceToOne >1, 'ascend');
5 sortedIndx ← concat(belowOneSorted, aboveOneSorted);
6 selectedIndx ← balancedFirst(groups, sortedIndx, sampleSize);
7 return data [selectedIndx, :]
```

4.3 Auxiliary functions

In this section we define the two help functions $calcRatios()$ and $balancedFirst()$ that we used in our main algorithms *ratioOne* and *belowOne*. In addition we give a brief description of all other functions that we use in the listings of this chapter.

function calcRatios(data, groups, neighborsNo): This function calculates hubness ratios for every sample in *data* array using $neighborsNo$ number of neighbors for calculating hubness scores. We repeat the definitions of *data*, *groups* and $neighborsNo$ arguments. *data* is a two dimensional array, with rows corresponding to samples and columns corresponding to attributes. *groups* is a one dimensional array that contains the class each sample belongs to. The $i$the

element of the *groups* array corresponds to the ith row of the *data* array. Its values are 0 for samples of the first class and 1 for samples of the second class.

This function iterates through all samples (lines 1 - 10) and for each sample it finds its *neighborsNo* nearest neighbors (line 2). Then it iterates through all nearest neighbors (lines 3 - 9) and for each nearest neighbor it either increases its bad hubness score (if nearest neighbor class and sample class do not match) or increases its good hubness score (if nearest neighbor class and sample class do match) (lines 4 - 8).

Function calcRatios(data, groups, neighborsNo)

```
   /* calculates hubness ratios of samples in data using
      neighborsNo number of neighbors.  groups is an array that
      contains the class each sample belongs to                */
   Input: data, groups, neighborsNo
   Output: ratios
 1 foreach sample in data do
 2 |  nnList ← findNN(data, sample, neighborsNo);
 3 |  foreach nn in nnList do
 4 |  |  if groups [nn] == groups [sample] then
 5 |  |  |  goodHubnessScore [nn]++;
 6 |  |  else
 7 |  |  |  badHubnessScore [nn]++;
 8 |  |  end
 9 |  end
10 end
11 ratios ← badHubnessScore / goodHubnessScore;
12 return ratios
```

function balancedFirst(groups, sortedIndx, sampleSize): This function selects *sampleSize* elements from the *sortedIndx* array so that the class ratio in the selected samples is equal to the class ratio in *groups* array. *groups* is a one dimensional array with values 0 and 1 that correspond to class labels. *sortedIndx* is a permutation of the numbers 1 to number of elements of the *groups* array. The ith element of the *sortedIndx* array corresponds to the $sortedIndx(i)$ element of the *groups* array. The function returns the first *sampleSize* elements from the *sortedIndx* array that match the class ratio equality criterion.

At first the function calculates the percentage of the first class in the *groups* array (line 1). Then it calculates the number of elements for each class that it should select (lines 2, 3).

At last the function selects the required number of samples for class one (lines 4-9) and for class two (lines 10-15)

Function balancedFirst(groups, sortedIndx, sampleSize)

```
/* selects sampleSize indices from sortedIndx so that class
   ratio in selected indices is equal to class ratio in groups
                                                              */
```

Input: groups, sortedIndx, sampleSize
Output: selected indices

```
1  groupOnePerc ← length(groups ==0) / length(groups);
2  groupOneNo ← ceil( sampleSize * groupOnePerc);
3  groupTwoNo ← sampleSize- groupOneNo;
4  indxPos ← 1;
5  for i ← 1 to groupOneNo do
6  |  while groups [sortedIndx[indxPos]]!=0 do indxPos ++;
7  |  groupOneIndx [i] ← sortedIndx [indxPos ];
8  |  indxPos ++;
9  end
10 indxPos ← 1;
11 for i ← 1 to groupTwoNo do
12 |  while groups [sortedIndx[indxPos]]!=1 do indxPos ++;
13 |  groupTwoIndx [i] ← sortedIndx [indxPos ];
14 |  indxPos ++;
15 end
16 return concat(groupOneIndx,groupTwoIndx)
```

function absoluteValue(arg): returns the absolute value of its argument.

function sort(array, order): returns the permutation of the indexes of *array* that succeeds in sorting the array in order specified by the *order* argument.

function concat(array1, array1): concatenates the two arrays without changing the order of the elements in each of them and with the elements of *array*1 coming before the elements of *array*2.

function findNN(data, sample, neighborsNo): finds the *neighborsNo* nearest neighbors of *sample* from *data*.

function ceil(arg): rounds *arg* to the nearest integer greater or equal to *arg*.

function length(array): returns the number of elements in the *array*.

Chapter 5

Experiments

In this chapter we compare the two instance selection methods introduced in the previous chapter against the random selection method. The latter just selects randomly, with equal probability, the requested number of samples. On one set of training data the random method was run five times and the mean accuracy was depicted as the accuracy of the random method.

The three methods were asked to select a specified number of samples from the total number and then a SVM was trained on the selected samples. Ten fold cross validation was used to calculate the accuracy of each method. The data sets introduced in chapter 1 were used in the experiments with the kernels that achieve the best accuracy for each data set as described in the same chapter.

5.1 Small sample sizes

In a first experiment the methods were asked to select one to ten (with step one) percent of the total number of samples. Figures 5.1, 5.2 show the results of the experiment.

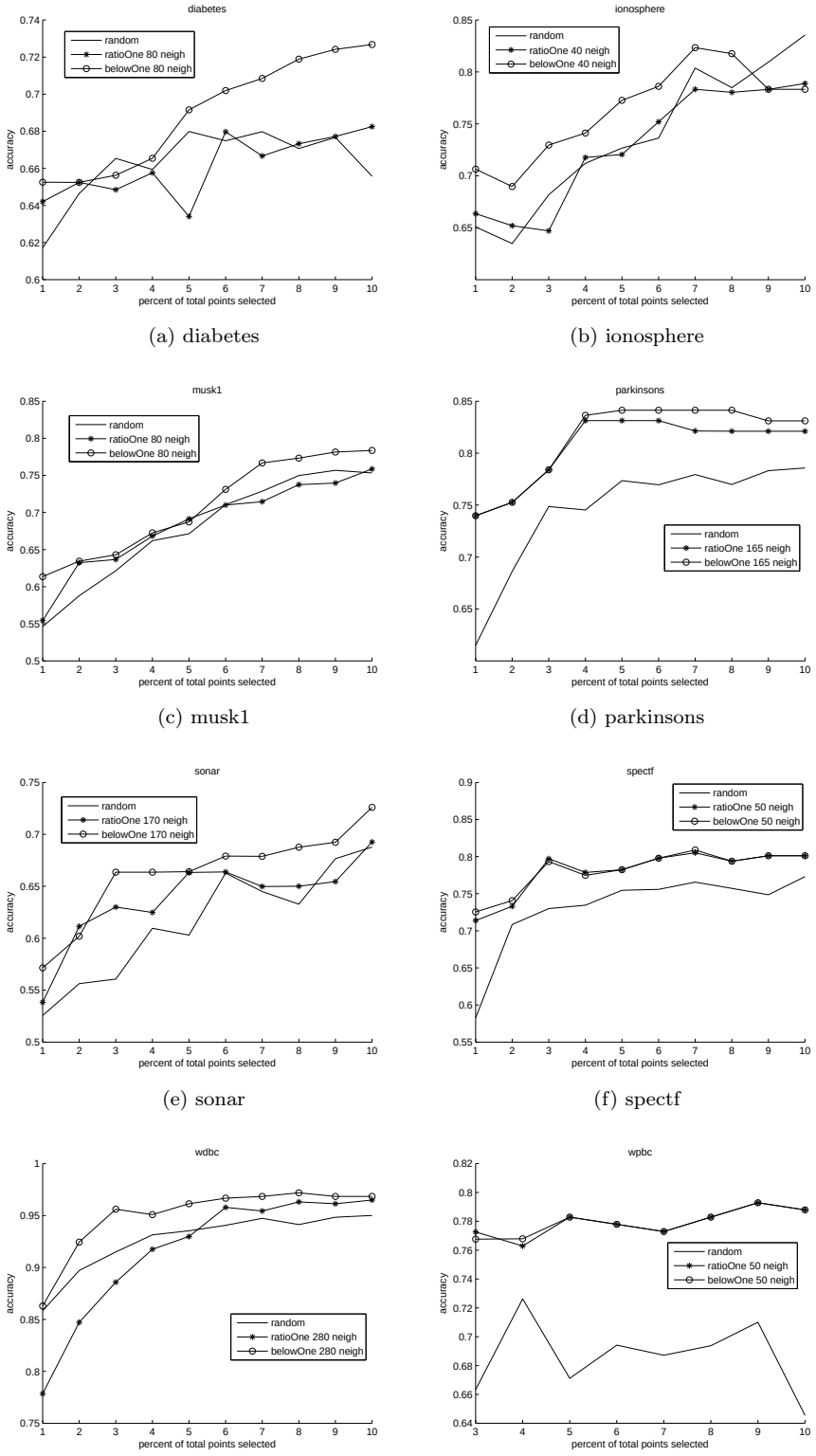
5.2 Large sample size

In a second experiment the methods were asked to select fifteen to fifty five (with step ten) percent of the total number of samples. Figures 5.3, 5.4 show the results of the experiment.

We must notice that the number of neighbors used by our methods for calculating hubness scores is different in the two experiments.

5.3 Conclusions

Table 5.1 shows the results of the experiment. The first sub-table gives the results for the small sample sizes case and the second sub-table for the large



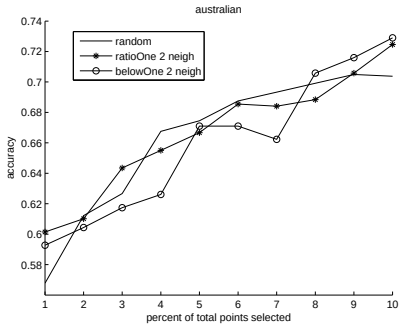
(a) diabetes (b) ionosphere

(c) musk1 (d) parkinsons

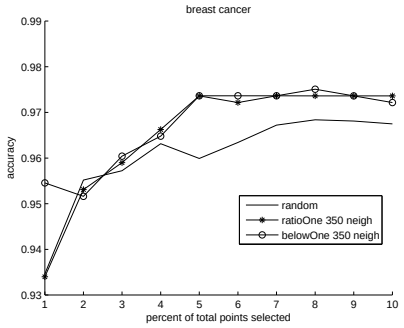
(e) sonar (f) spectf

(g) wdbc (h) wpbc

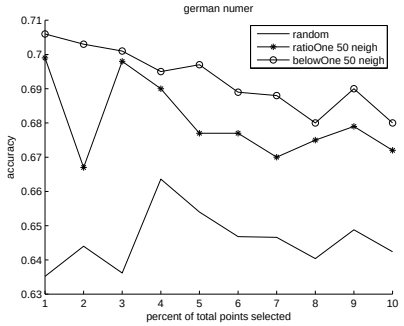
Figure 5.1: Comparison of the accuracy of ratioOne, belowOne and random instance selection methods. One to ten percent of the total number of samples was asked from the methods to return.



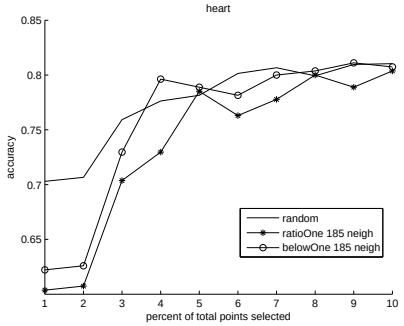
(a) australian



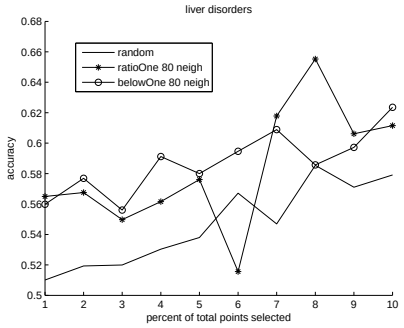
(b) breast cancer



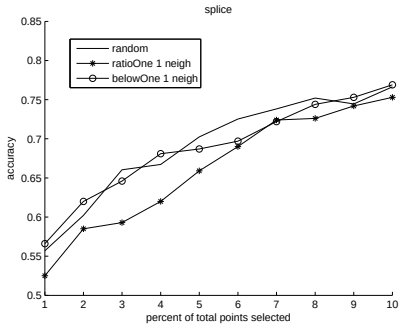
(c) german numer



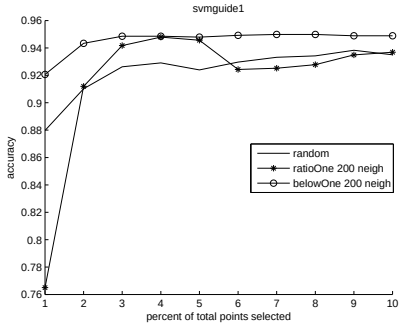
(d) heart



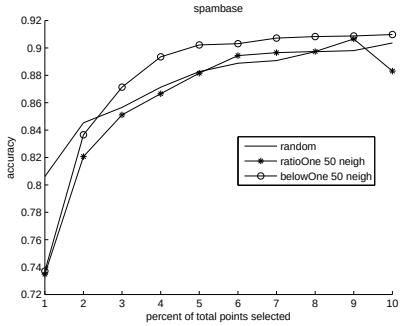
(e) liver disorders



(f) splice

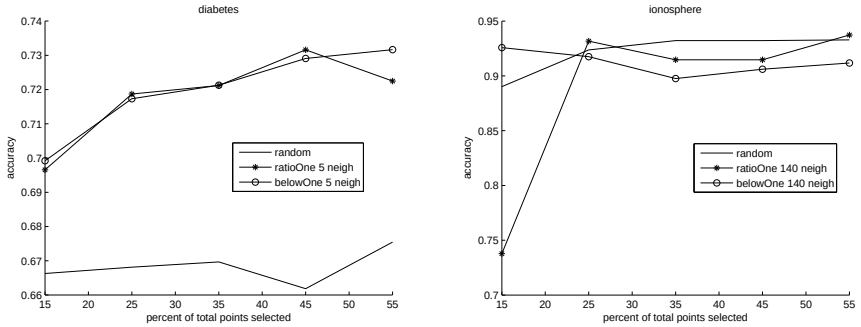


(g) svmguide1



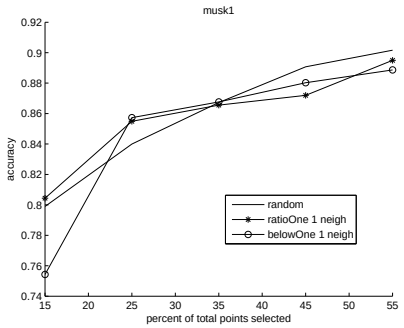
(h) spambase

Figure 5.2: Comparison of the accuracy of ratioOne, belowOne and random instance selection methods. One to ten percent of the total number of samples was asked from the methods to return.

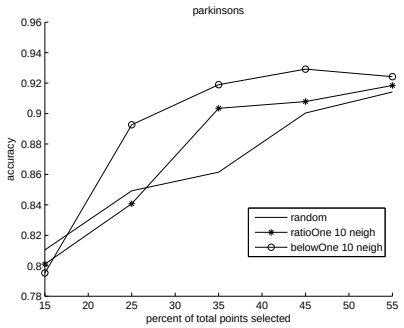


(a) diabetes

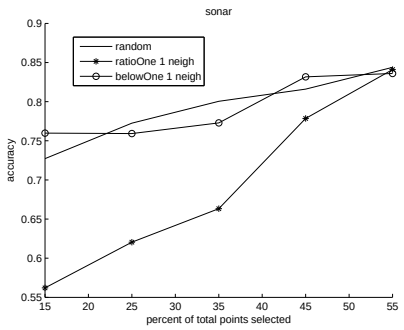
(b) ionosphere



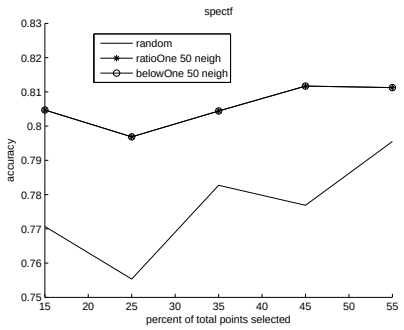
(c) musk1



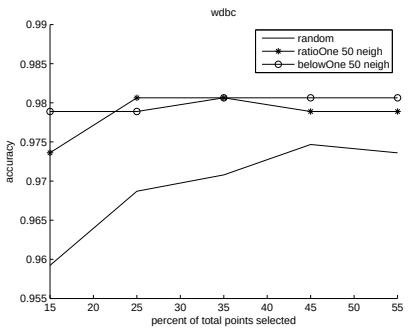
(d) parkinsons



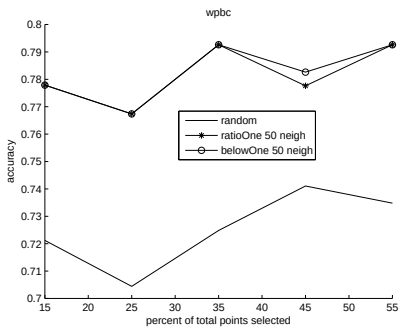
(e) sonar



(f) spectf

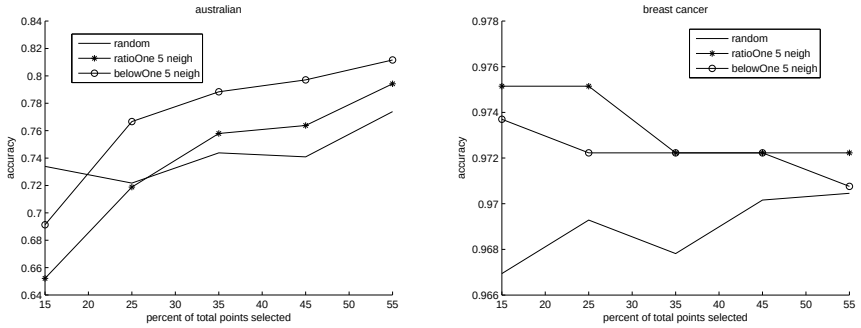


(g) wdbc



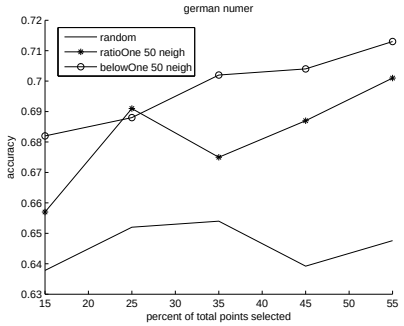
(h) wpbc

Figure 5.3: Comparison of the accuracy of ratioOne, belowOne and random instance selection methods. Fifteen to fifty five percent of the total number of samples was asked from the methods to return.

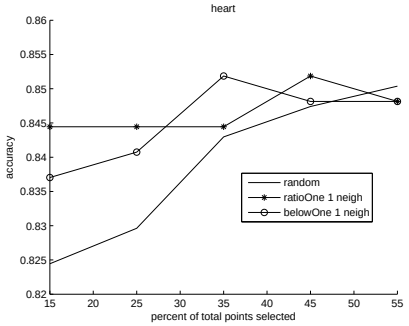


(a) australian

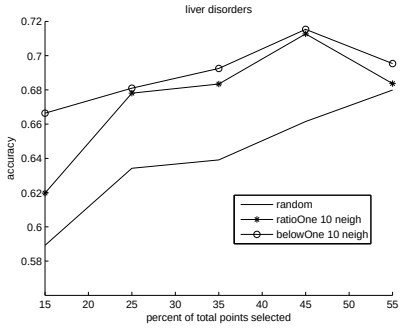
(b) breast cancer



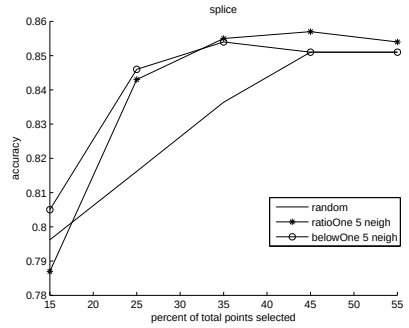
(c) german numer



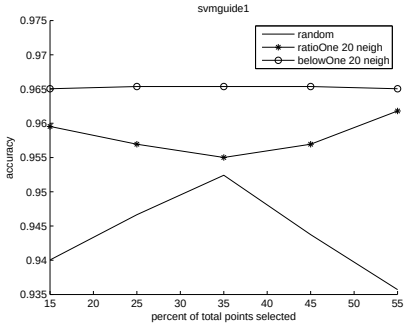
(d) heart



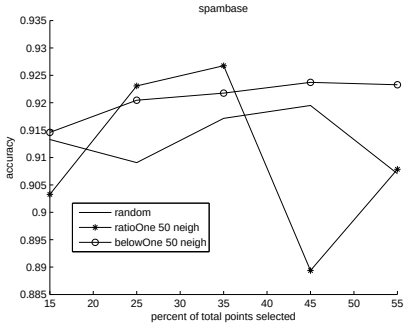
(e) liver disorders



(f) splice



(g) svmguide1



(h) spambase

Figure 5.4: Comparison of the accuracy of ratioOne, belowOne and random instance selection methods. Fifteen to fifty five percent of the total number of samples was asked from the methods to return.

	wins	ties/loses
belowOne	13	3
ratioOne	8	8

(a) small sample sizes

	wins	ties/loses
belowOne	13	3
ratioOne	12	4

(b) large sample sizes

Table 5.1: Summary results of the experiments comparing ratioOne and belowOne instance selection methods to the random instance selection method.

sample sizes case. Each sub-table shows the number of times our methods had better accuracy than the random method and the number of times they had similar or worst accuracy than the random method.

From the table we can conclude that the belowOne method outperforms the random method. The ratioOne method is comparable to the belowOne method in large sample sizes but in small sample sizes it is not as stable as the belowOne method.

The reason for that is that the ratioOne method selects noisy samples with hubness ratio bigger than one. These noisy samples can mislead the SVM especially when the number of samples used for training the SVM is small. When the sample size selected by the ratioOne method is getting bigger the SVM can better figure out the class boundary.

On the contrary the belowOne method avoids the noisy points and concentrates on the class means, succeeding better in describing the classes even for small sample sizes.

Chapter 6

Conclusions and future work

This work has looked into the relation between hubs and support vector machines, especially support vectors. We have concentrated on the differences between high and low dimensionality. Both theoretical an practical results have been established. These results have been verified both on synthetic and real life data sets.

We have explored the location of bad hubs relatively to the separating hyperplane defined by a SVM trained on samples belonging to two different classes and shown that:

- Bad hubs tend to be support vectors but the reverse does not hold. That is there are support vectors that are not among the top bad hubs.
- In low dimensionality top bad hubs behave like outliers.
- In high dimensionality top bad hubs may not be support vectors, because there is a strong correlation between bad and good hubness.

The latter is by itself a strong result:

- In high dimensionality strong bad hubness implies strong good hubness and the reverse, even when the two classes are linearly separable. The reason for this is that in high dimensionality the distance from a sample to its class mean is a lot bigger than the distance between the two class means.

We have defined hubness ratio as the ratio of bad hubness score to good hubness score. We have shown the importance of samples with hubness ratio close to one:

- The percent of support vectors that are close to hubness ratio one increases as dimensionality increases.
- Samples with hubness ratio close to one are closer to the hyperplane than other samples.

- Support vectors' hubness ratio is closer to one than non support vectors' hubness ratio.
- In high dimensionality support vectors' hubness ratio is more concentrated around the value of one than in low dimensionality.

We have used the previous facts to define two instance selection methods:

- RatioOne which picks samples with hubness ratio close to one.
- BelowOne which picks samples with hubness ratio close and smaller than one.

The methods were tested against the random selection method for small and bigger sample sizes and the results were satisfactory. The belowOne method outperformed the random method in small and large sample sizes. The ratioOne method did not have equally good results in small sample sizes because it selects noisy samples with hubness ratio bigger then one. In large sample sizes its performance is comparable to the belowOne method.

Nevertheless there are open questions that need to be answered. We would like to devise an algorithm to figure in advance the right number of neighbors that need to be used by our two instance selection methods. We would like to explore more the conditions under which our methods perform the best and if possible devise an algorithm that based on special features of a data set would suggest our methods as the best choice of instance selection.

Good hubs and possibly a hybrid instance selection method based on good and bad hubs suggest another field of future exploration.

Finally during our work we have seen that there is a big difference between trying to guess support vectors and trying to build a classifier that achieves the best accuracy given that it is trained only on a small portion of the data set. We would like to make this distinction more clear.

Bibliography

[1] C. Aggarwal, Alexander Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space. *Database Theory ICDT 2001*, pages 420–434, 2001.

[2] M. Barros De Almeida, A. de Padua Braga, and J.P. Braga. SVM-KM: speeding SVMs learning with a priori cluster selection and k-means. *Proceedings. Vol.1. Sixth Brazilian Symposium on Neural Networks*, pages 162–167, 2000.

[3] Kevin Beyer, Jonathan Goldstein, R. Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? *Database Theory ICDT 99*, pages 217–235, 1999.

[4] Cristopher Bishop. *Pattern Recognition and Machine Learning*. Springer.

[5] Damien Franc, Vincent Wertz, Michel Verleysen, and Senior Member. The Concentration of Fractional Distances. *Knowledge Creation Diffusion Utilization*, 19(7):873–886, 2007.

[6] Bernd Fritzke. A growing neural gas network learns topologies. *Advances in neural information processing systems*, pages 625–632, 1995.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer.

[8] Ravindra Koggalage and Saman Halgamuge. Reducing the Number of Training Samples for Fast Support Vector Machine Classification. *Processing*, 2(3), 2004.

[9] Elizaveta Levina and PJ Bickel. Maximum likelihood estimation of intrinsic dimension. *Ann Arbor MI*, 2004.

[10] Xun Liang. An effective method of pruning support vector machine classifiers. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 21(1):26–38, January 2010.

[11] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge University Press.

[12] Roland Nilsson, Jose M Pena, Johan Bjorkegren, and Jesper Tegner. Evaluating feature selection for SVMs in high dimensions. *European Journal of Operational Research*, 156(2):483–494, July 2004.

[13] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Nearest neighbors in high-dimensional data. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, 2009.

[14] H Shin and S Cho. Invariance of neighborhood relation under input space to feature space mapping. *Pattern Recognition Letters*, 26(6):707–718, May 2005.

[15] Jigang Wang, Predrag Neskovic, and L.N. Cooper. Training data selection for support vector machines. *Advances in Natural Computation*, pages 554–564, 2005.

[16] Roger Weber, HJ Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proceedings of the*, 1998.

[17] MH Yang and N. Ahuja. A geometric approach to train support vector machines. In *cvpr*, page 1430. Published by the IEEE Computer Society, 2000.

[18] Mario Zechner and Michael Granitzer. A Competitive Learning Approach to Instance Selection for Support Vector Machines. *Knowledge Science, Engineering and Management*, pages 146–157, 2009.