Decision Boundary Approximation:

A new method for locally explaining predictions of complex classification models.

Author: Georgios Vlassopoulos (s1950193)

Thesis advisor: Dr. Tim van Erven

MASTER THESIS

Defended on 11 July 2019

Specialization: Data Science

STATISTICAL SCIENCE FOR THE LIFE AND BEHAVIOURAL SCIENCES

Abstract

Machine Learning classifiers are naturally black boxes when it comes to interpretation. In this thesis, Decision Boundary Approximation (DBA), a new algorithm for locally explaining complex binary classifiers is developed, tested experimentally and discussed. The algorithm explains predictions of individual instances, by approximating their most relevant region of the decision boundary with a linear model. We overview and discuss limitations of existing methods when applied to classification, with specific focus on LIME due to the similarity with DBA concepts. Experiments with DBA, cover both low dimensions and sparse high dimensional data. In Experiment 1 we show that DBA can provide stable explanations for various decision boundary structures in a 2D simulated case. Experiment 2 demonstrates that DBA outperforms LIME for low dimensionalities, while in Experiment 3 (MNIST data) we show that when data are sparse, DBA explanations can include features that are absent from the explained example, making the explanation more complete compared to LIME. In Experiment 4 we explain a Naive Bayes trained on SMS ham/spam messages and show that the DBA solution is in agreement with the Naive Bayes posterior. Finally, the benefits and drawbacks of DBA are discussed elaborately and future recommendations for modifications are given.

Dedication

To those who inspired me.

Acknowledgements

I would like to express my gratitude to Leiden University for giving me the opportunity to learn. I want to thank all of my teachers and fellow students for sharing their knowledge with me during my studies.

A special thanks goes to my supervisor Dr. Tim van Erven, for choosing me for this project and more importantly, for caring for my future. His support, feedback and trust were above expectations. I also thank Prof. Dr. Eric Postma (Tilburg University), Prof. Dr. Mykola Pechenizkiy (Eindhoven Univerisity), Dr. Henry Brighton (Tilburg), Dr. Vlado Menkovski (Eindhoven), Geoffrey Stoel and his collegues (KPN telecommunications) for offering me a PhD vacancy to continue my research in Explainability. I would also like to thank my friend Dr. Harald van Mil (Leiden University), for his personal interest during my studies, his valuable help when I needed it and the projects he shared with me.

Of course another special thanks goes to my friends in Greece and the Netherlands (you know who you are) who believed in me and unconditionally supported me. You all have a beautiful mind and I'm proud of you.

More importantly, I would like to thank the person behind everything, my breath, partner and love Maria Banagou. If it weren't for you, these words would have never been written. However, I do not love you for that, but for what you really are.

Last but not least, I would like to thank my mother for her fairytales and my father for teaching me that life is not a fairytale.

Contents

Chapter 1

Introduction

Explainability is a new emerging area which attempts to provide approximate interpretations for the predictions made by Machine Learning models. We need *transparency* in complex machine learning to be able to shed light on the black box process of the machine's decisions. However sometimes complete transparency is impossible, either due to the high complexity of the classifier which may use relationships that do not match human intuition at all, or because the size of the information is vast compared to what the decision maker can handle. Thus when deriving an explanation, one should consider options for simplifying the feature space or the relations used by the classifier. This is a trade-off between *fidelity* and *interpretability.*

Methods towards this direction are divided into two main categories: *model agnostic* and *model specific* methods. In model agnostic methods the explainability algorithm ignores the structure and the training process of the classifier and only has access to the prediction function and (possibly) the training set. This category contrasts *model specific* methods that are based on the properties and structure of the trained model, such as propagating effects through a network's layers (Kuo et al. 2018). In general, model specific methods have more resources to exploit in order to bring transparency, although in some cases they simplify the training process to derive an interpretable result (O. Bastani, Kim, and H. Bastani 2017). Hence, a clear advantage of model agnosticism is that no accuracy is sacrificed to increase interpretability, due to the independence from the trained structure. Most of model agnostic methods involve fitting an *explainer* (surrogate approaches), i.e. an additive (sparse) linear model to *learn* the predictions of the complex model (Ribeiro, Singh, and Guestrin 2016) or study cases to understand patterns (Laugel et al. 2017).

Another important distinction that should be considered, is between *local* and *global* explanations. Global explanations attempt to provide transparency on the overall decision rule by fitting the explainer to a labeled sample of the whole population (e.g. the training set). On the contrary, *local* explanations aim to explain the behaviour of the decision rule in the *neighborhood* of an instance x_0. When the model is highly non-linear, a local explanation may highly deviate from the global (pooled) picture: each case is subject to a rule of different nature. Thus, global explanations can be over-simplistic (or even worse impossible) when the model is complex, because locally important features will not necessarily be globally important.

One of the most popular, due to its simplicity, methods for providing locally faithful model agnostic explanations is LIME (Local Interpretable Model agnostic Explanations) developed by Ribeiro, Singh, and Guestrin 2016. The algorithm attempts to learn the important features in the neighborhood of an example case x_0, by maximizing the agreement of the explainer with the complex model in the locality of x_0. However we will argue that when it comes to classification such an approach does not always result to a linearization of the actual decision rule, which is the decision boundary. It will be shown that surrogate explanations might be misleading, unless a direct decision boundary linearization is employed.

In this thesis, we propose a novel model agnostic approach in Explainability: *Decision Boundary Approximation (DBA)*, which aims to extract faithful case-wise explanations by *linearizing* relevant decision boundary regions (i.e. parts of the classifier's decision rule that are locally relevant to x_0). The method consists of three main steps:

- **Detect** the closest decision boundary point for the case x_0 to be explained.
- **Simulate** data around this point such that both classes are sampled equally. Obtain the simulation predictions from the complex model.
- **Explain** this part of the decision boundary by fitting a penalized logistic regression in the labeled simulation set.

If the simulation design is sufficient, the explainer will fit a linear hyperplane to approximate a specific region of the decision boundary - the region which is *closest* to the explained instance, according to a similarity measure. The method results in approximating the behaviour of discrimination between two subsets of cases: similar cases to the instance to be explained and a certain subset of opposing points that lie from the other side of the boundary region that we approximate. By taking in account only these two subsets, the non-linear behaviour will be much easier approximated with a linear model, since the subsets will tend to be separable. An explanation of this kind could provide insight to questions such as: "Why does the classifier, classify this image as cat, while that as dog?" or "Why is this sales plan predicted as failure when others alike it as success?". Another important benefit which arises by explaining the closest decision boundary region is a trustworthy direction to change the classifier's prediction. Having a case x_0 classified as "A" one could wonder which feature values of x_0 should change (and to which extent) in order for the classifier to activate "B": "Which is the minimal variation we should apply in the example sales' plan for the classifier to classify as success?". We will show that detecting and approximating the closest decision boundary region can in some cases provide an answer that might face practical application.

Outline The thesis will begin with an overview of existing methods (Chapter 2) with special attention to LIME due to its similar concepts with DBA. In Chapter 3, we will introduce DBA and elaborate on its theoretical assumptions. We will provide illustration of the algorithm's functions through toy examples that motivate its components. In Chapter 4, the algorithm will be tested in a series of 4 experiments of increasing dimensionality. Experiment 1 will test the stability of the algorithm and the effect of complexity of the decision boundary on the final explanation in a 2-dimensional simulated case, for various non-linear classifiers. In Experiment 2, we will perform a low dimensionality comparison with LIME in 3 datasets and 4 different boundary structures. Experiment 3 is an application to MNIST data in which the explainer employs dimensionality reduction to assess the impact of dimensionality on the simulation process. Two modifications of the default simulation process will be compared. Furthermore, DBA will be employed to produce explanations for the missclassified cases to reveal the classifier's biases. In Experiment 4 we explain a Naive Bayes classifier which is trained on SMS spam/ham messages to compare the DBA explanations with the Naive Bayes posterior probabilities. We conclude this work with a discussion of the topics presented, along with proposals for modification and future work (Chapter 5).

Chapter 2

Existing Methods

In this chapter we overview, discuss and show the limitations of some of the most similar concepts with the method developed in this research, to motivate our selections. In section 2.1, we provide a categorization of Explainability methods along with references on the literature of the most representative examples of each approach. Section 2.2 lists several approaches that belong on *model agnostic* category (in which DBA belongs), i.e. general methods that can explain any classifier. In section 2.3, the technical background of the most similar algorithm to DBA (LIME) are described. Moreover, the limitations of LIME are discussed in 6 main points. Finally in section 2.4, the motivation for developing DBA is summarized.

2.1 Taxonomy of methods

One can categorize the different approaches in Explainability in various categories. In this section we provide an overview of these categories and mention some of the most important explainability methods.

Distinction 1 - Model-specific vs Model-agnostic

Model agnostic methods treat the classifier as a *black box*, ignoring its internal mechanisms. LIME (Ribeiro, Singh, and Guestrin 2016) is an example of a model agnostic method since it only requires access to the predicted probabilities. A *model specific* method takes into account the model parameters and exploits the model's structure to extract the explanation. This means that a model-specific method is designed specifically for a certain type of models (e.g Neural Networks). For networks, there has been a lot of work to design model specific frameworks for transparency. A few examples are Deeplift (Shrikumar, Greenside, and Kundaje 2017), Layer-Wise Relevance Propagation (Bach et al. 2015), CAM (Zhou et al. 2015), Integrated Gradients (Sundararajan, Taly, and Yan 2017) and tree-based DeepRed (J.R., E., and F. 2016).

Distinction 2 - Intrinsic vs post hoc

Intrinsic explanations aim to incorporate the explanation process into the training procedure of the model (Du, Liu, and Hu 2018). For instance Zhang, Wu, and S. Zhu 2017 have proposed a method for training interpretable Convolutional Neural Networks (CNN), by introducing a generic loss to regularize the representation of filters and improve the network's interpretability. In some cases intrinsic methods sacrifice accuracy for interpretability in order to derive the explanations. On the other hand, *post hoc* explanations are derived after the classifier has been trained. A model-agnostic explanation is by necessity post hoc, however post hoc explanations can also be model specific.

Distinction 3 - Global vs local

Explanations can either refer to the overall behaviour of the classifier (global) or the locality of a single instance (local). The term *local* means that explanations are subject specific: for each data instance to be explained, we obtain a different explanation which is only *faithful* to the model for a certain subset of the feature space. Explanations of this kind are valuable for decision makers, that are looking for answers regarding single data points. In this thesis when a local method is employed we will refer to the explained data instance as x_0.

The *global* explanation corresponds to the overall behaviour of the classifier (marginal effect of features) and provides a picture for the average importance. If the classifier is highly non-linear, local explanations will differ from the global explanation. Global explanations are useful for putting trust on the model as a whole, however they cannot be representative for all data instances, when the classifier is not linear.

Distinction 4 - Example based vs surrogate methods

Example based methods focus on example cases that are used as representatives to qualitatively interpret the model. They aim to make the interpretation *human intuitive*. This is more useful in general for image or text data, where we can visualize different predicted instances for an intuitive explanation. For example, for image data one can visualize the pictures of similar data cases (*prototypes*) to the explained instance x_0 for both classes. An important sub-class of this category are *counterfactual* explanations (Wachter, Mittelstadt, and Russell 2017). Counterfactual explanations simulate opposite cases of x_0 that represent variations that force the class to *change*. They aim to provide an answer to the question: "Which variation should we apply to x_0 in order to force the classifier to change its prediction?". Counterfactual explanations can be simulated cases rather than training data but still represent an instance that can be compared to x_0 by obtaining feature differences. To derive a counterfactual explanation in binary classification, we would like to locate the closest data point classified in the opposite class of x_0. Multiple approaches have been proposed such as a greedy *growing spheres* algorithm (Laugel et al. 2017) to optimize a loss and locate with simulation the closest point from the opposite class. We will illustrate how DBA shares rather similar concepts with this type of explanations, since in its first step it locates the most similar decision boundary point to x_0, thus it derives a counterfactual explanation.

In contrast to example based explanations, *surrogate* methods aim to employ an *explainer*, i.e. a linear interpretable model, to learn the predictions of the complex model, either locally or globally (Ribeiro, Singh, and Guestrin 2016 , Bucila, Caruana, and Niculescu-Mizil 2006). The explanation is a coefficient vector β which reflects the effects of the features on the prediction(s) of the classifier. As it will be illustrated in Chapter 3, DBA practically shares both counterfactual and surrogate nature.

In the next section we provide an overview of some of the most commonly used model agnostic methods, since Decision Boundary Approximation belongs in this category.

2.2 Model agnostic methods

Although within the last two years, model specific methods have flourished, model-agnostic methods are still of limited number. This research aims to contribute towards the direction of model-agnosticism. Due to the focus on this type of methods, in this section we overview the most frequently used model agnostic methods (apart from LIME which is discussed separately in section 2.3).

2.2.1 Visual Methods

Visual methods attempt to shed light on predictions of complex models by plotting the effects of a small number of features (≤ 2) on the predicted response (in classification the predicted probability). This is mainly achieved through decomposition of the partial dependence function. The partial dependence function in binary classification can be expressed via the log-odds analogue (Hastie et al. 2009),

$$f_k(X_1, ..., X_p) = \log\left[P_1(X_1, ..., X_p)\right] - \log\left[P_0(X_1, ..., X_p)\right], \tag{2.1}$$

where X_i is a feature employed by the model, p the dimensionality, $P_1(X_1, ..., X_p)$ is the predicted probability (given the values of the p features) for class *1*, while P_0 for class *0*. This expresses the log-odds as a function of p features. If we marginalize f_k by assuming that only k of these features vary, we can visualize the partial dependence of the average log-odds on the k features. Due to visualization restrictions, in practice $k \leq 2$. This yields a global visual interpretation method: PDP (partial dependence plot), (Friedman 2001, section 8.2). The method assumes independency of features, which is a clear disadvantage when it comes to correlated data structures. Another inherent drawback is the restriction that we can only examine the effect of at most 2 features, due to visualization limitations. However, this approach is frequently preferred due to its simplicity and intuitiveness.

A local variant of PDP methods for individual instances is mentioned as Individual conditional expectation plot (Goldstein et al. 2015). To plot an individual instance prediction path, only one feature is varied in a grid and the rest are kept constant. Then, the predicted probabilities of the variants of the instance are obtained from the complex model and eventually plotted in a simple line plot. All instances can be plotted on the same plot to study the global behaviour for this feature. An even more careful local method, ALE (Apley 2016), attempts to approximate the gradient of the partial dependence function to extract the feature importance. This may take into account the correlations between features, but still bears the limit of dimensions explained due to visualization dimensionality restrictions.

Overall, visualization methods are attractive due to human intuitiveness; yet the complete picture for the global or local behaviour cannot be properly captured, unless the effect of all features on the predictions is explored simultaneously. In a model agnostic framework this can only be managed through surrogate models. In the next section we will discuss the most simple surrogate approach, a method which attempts to explain by fitting an additive linear model as an explainer.

2.2.2 Global surrogate

The global surrogate is a naive post-hoc model agnostic surrogate approach which aims to globally learn the predictions of the complex model. Predictions on the training set T (or any sample of the population) are acquired through the model's prediction function. A linear interpretable model g fits in $(T, f(T))$ to globally maximize its agreement with the classifier f. The only measure available to evaluate the explanations is the coefficient of determination,

$$R^2 = 1 - \frac{\sum_i (f(x_i) - g(x_i))^2}{\sum_i (f(x_i) - \frac{1}{n}\sum_i f(x_i))^2} \tag{2.2}$$

which reflects the percentage of variability of the predictions $f(x)$ that the linear model g can capture. Misleading results can arise when the decision boundary is highly non-linear. The explainer might face instability and effects that are locally important might be cancelling out. That is the motivation that led Ribeiro, Singh and Guestrin to introduce LIME, a local surrogate approach which is discussed in the following section.

2.3 LIME

Local Interpretable Model-agnostic Explanations,(Ribeiro, Singh, and Guestrin 2016) is a surrogate model agnostic method which attempts to provide local explanations for single instances. The approach aims to bring a balance between *fidelity* (how accurately is the complex model approximated) and *interpretability* (how user-friendly are the explanations). The most fundamental assumption of the framework is that any non-linear model is locally linear, thus the authors propose to maximize explainer faithfulness for single instances. The current section will illustrate some fundamental aspects of LIME algorithm, along with its limitations (section 2.3.5). The main concepts of the algorithm are distributed in four main paragraphs (2.3.1-2.3.4).

2.3.1 The latent space

LIME authors begin by introducing *interpretable data representations*. They argue that although the classifier might use complex data structures such as sequences of words, the explanation method should assume representations compatible with human intuition (e.g a bag of words). For that reason, they map each instance $x \in \mathbb{R}^p$ to a representation $x^{'}$ in a latent feature space $\{0,1\}^{p^{'}}$ where $p^{'} \leq p$. Thus, in LIME framework each instance is represented by a binary vector $x^{'} \in \{0,1\}^{p^{'}}$ (presence or absence of features). For image data an image segmentation process is included to create *superpixels* i.e. meaningful batches of pixels (for example the ear of a cat) and use them as interpretable data representations. In this case, each entry of the binary vector will correspond to presence or absence of a certain superpixel, efficiently decreasing p. For simpler problems with numerical features it is also possible to ignore binary vector representations and work in the full feature space.

2.3.2 The general framework

The main idea of LIME lies on the maximization of the **local** faithfulness of an explainer $g : \{0,1\}^{p^{'}} \to [0,1]$ to the complex classifier $f : \mathbb{R}^p \to [0,1]$ [1] under the constraint of a threshold for complexity (set by the user). This constraint can be defined by a measure of complexity $\Omega(g)$ (e.g the number of non-zero weights of the explainer). The model g may be any model in a set of interpretable models G.

To maximize faithfulness in the locality, the authors propose a loss which is weighted by proximity with the explained example x_0. If we denote this proximity measure by $\pi_{x_0}(x)$ to express the similarity between x_0 and an instance x, the explanation can be expressed as:

$$\xi(x_0) = \underset{g}{\operatorname{argmin}}\{\mathscr{L}(f, g, \pi_{x_0}) + \Omega(g)\}, \tag{2.3}$$

where $\mathscr{L}(f, g, \pi_{x_0})$ is a loss reflecting the agreement in predictions of f and g in the locality of x_0. The authors propose a quadratic weighted loss for this purpose:

$$\mathscr{L}(f, g, \pi_{x_0}) = \sum_{x} \pi_{x_0}(x)(f(x) - g(x^{'}))^2 \tag{2.4}$$

where summation here is over all simulated instances and $\pi_{x_0}(x)$ a proximity measure representing the similarity of x_0 with x. Since the loss is weighted by proximity, the explainer will tend to favor the similar points with x_0 leading to a natural bias towards the example x_0.

[1] In LIME default framework the set of values of g and f is $[0,1]$ rather than $\{0,1\}$ because they learn the predicted probabilities and not the labels.

As a complexity measure, the authors propose $\Omega(g) = \infty\mathbf{1}[||\beta||_0 > K]$, i.e. setting a limit in the number of features used. Here, K denotes the user threshold for the number of features used by the explainer in the sparse model, $||\beta||_0$ the L_0 norm [2] of the estimated coefficient vector and $\infty\mathbf{1}$ the characteristic function (participation or not in the feature set).

LIME attempts to approximate the behaviour of f in the neighborhood of x_0, by solving the minimization problem 2.3. The above general expression can be written specifically for any choices of g, π_{x_0} and Ω. In LIME's default setting this loss is minimized by fitting an interpretable regression model (e.g L_2 - linear model or decision tree) to learn the class probabilities predicted by f, on a set of simulated data points. The authors note that with this $\Omega(g)$ choice, (2.3) cannot be solved directly, so first they perform feature selection with L_1 regularization (LASSO) to select the K-most important features and then learn a sparse explanation through least squares. In the next paragraph LIME's simulation procedure is described.

2.3.3 Sampling procedure

Depending on the nature of the data, LIME's simulation procedure varies. We summarize the main alternatives that LIME employs for simulation.

Numerical features When features are numerical, LIME's implementation involves computing statistics for each feature (mean and standard deviation), thus the process depends on the training set. They then sample each feature from $N(0, 1)$, multiply by its standard deviation and add its mean to obtain a sample. This is equivalent with sampling from the center of mass of the training set. In this case the explained instance x_0 is not involved in the process of simulation and the idea of the latent space, described in section 2.3.1, is dropped. Finally, we note that here the user has the option of discretizing the features into quantiles.

Image and text For text and image, the authors of LIME propose a sampling procedure which only depends on x_0. They simulate permutations of the instance x_0, by binarizing it and drawing from its **non-zero** elements (elements that correspond to 1 in the binary vector). If the total number of features is p', they uniformly draw a random number k from the set $\{1, ..., p' - 1\}$ and then uniformly draw k non-zero elements from the binary vector, to create a permuted binary vector. The permuted binary vectors can be used as *filters* on x_0 to produce a sample in the full feature space (for example the original image with some random parts completely missing).

After a sample is available, its predictions are acquired by the complex model. The authors claim that these predictions can be either the labels or the predicted probabilities, however we will show that properly learning the labels under this framework is not possible (section 2.3.5).

[2]Not actually a norm, this symbolism denotes the number of non-zero elements of the vector.

In section 2.3.5 it will be illustrated that both procedures have problems when it comes to classification, especially for numerical features and sparse text/image data, by means of stability and nature of the estimates.

2.3.4 Explanation procedure

The explanation in the neighborhood strongly depends on the definition of the proximity measure. The authors propose for such a measure

$$\pi_{x_0}(x) = \exp\{-d(x, x_0)^2/\sigma^2\} \tag{2.5}$$

i.e. an exponential kernel with d a metric and σ the width of the kernel. A large kernel width means that by keeping $d(x, x_0)$ constant, instances receive a larger weight compared to small kernel widths. Thus the contribution in the loss of points with large $d(x, x_0)$ will be significantly larger than cases where σ is small. The choice of the metric d is left open, although Euclidean metric is proposed for numerical features.

Putting the above together, LIME simulates instances and then weights them by computing $\pi_{x_0}(x)$ for each simulated instance x. It then performs feature selection to select the K-most important features according to Ω. It then solves a weighted (by π_{x_0}) least squares problem to learn the predictions of the complex model. Eventually the explanation produced, is a vector of coefficients β which represent the effect for each feature (or interpretable representation) on the predictions of the complex model (here probabilities). That is a local explanation and is faithful to f only in the neighborhood of x_0, i.e. it attempts to capture the important features for classification in the locality of the explained instance. Ribeiro et al claim that the value of the loss (2.3) can be reported as an evaluation measure for the explainer's local fit.

Although LIME's concepts maintain a user-friendly balance between simplicity and formality, there are inherent problems for explaining classifiers with numerical features and explanations in many cases can be misleading. We argue that in the general case, it is impossible to linearize the neighborhood in a meaningful way if we do not consider only a certain part of the decision rule, i.e. we should become specific of which decision boundary region we attempt to explain. In the next section we will discuss the inherent problems of LIME in detail and provide illustrative examples to support our argumentation.

2.3.5 Limitations

It is argued that in classification tasks and some specific data types, the way that LIME approximates is not sufficient and its explanations might be misleading. We summarize its drawbacks in 6 points which are discussed below.

Point 1: When data are sparse, sampling from x_0 is not sufficient.

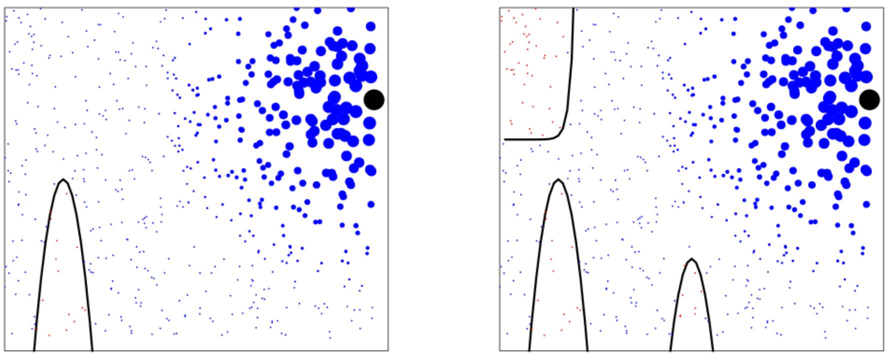
Consider a text message x_0 with only 10 words while the classifier might have been trained in a vast vocabulary. Thus, the representation of the message in the feature space would be a sparse vector with 10 non-zero elements. Under LIME framework, permuted instances would be created by sampling a number $k \in \{1, ..., 9\}$ and then sample k from the 10 words of x_0. The algorithm will then fit a linear model to learn which is the effect of each one of these words to the classification result. However, the fact that some words are missing from the message might also contribute to the classification result. For instance, explaining the prediction of a "spam" message should also include missing words that are important for the "ham" class. LIME framework completely ignores the effects of features that have zero value in the explained example, which results to **overestimating** the non-zero elements. In Experiment 4, we will show that there are sparse cases that missing features are more important than non-zero elements of x_0 (see section 4.4.4, Table 4.3). In such cases, LIME explanation will be incomplete and possibly misleading. Hence we urge the need of also exploring features that have zero value in the explained data example.

Point 2 - There is no formal rule to define the optimal value for the kernel width σ.

The kernel width is a vital parameter in LIME framework. The explanation strongly depends on this parameter since it adjusts the weights in the loss (2.4). It seems clear that the value of σ should depend on dimensionality, the classifier and the sparsity of the data. We argue though that in classification it should also depend on the distance from the decision boundary. Specifically, the width should be smaller when the point lies very close to the decision boundary, because then the effect of this specific decision boundary region would overcome any other region. If the x_0 lies far away from the decision boundary, we would like to weight instances with a larger kernel width to also consider instances from the opposite class in the loss. However, the authors provide no rule for formally assessing the width for the kernel and in this manner, the concept of neighborhood becomes more abstract: if we are truly to learn the effects in the neighborhood, the kernel width should always be small. Assuming large kernel widths may lead to other problems for some boundary structures, such as low class balance and poor explainer fit. It is also reasonable to assume that by varying σ the explanation will change substantially.

Point 3 - Learning probabilities in insensitive gradient neighborhoods can be problematic.

Let $S = \{x_j\}_{j=1}^{n}$ with $x_j \in \mathbb{R}^p$ be a set of p-dimensional data points that represents a grid with vanishing step which covers a finite region of $\mathbb{R}^p$. For each point x_j, we can obtain the predicted probability $P(C = 1|x_j)$ through the black box model. If we picture these probabilities as a *field* we can estimate the gradient of



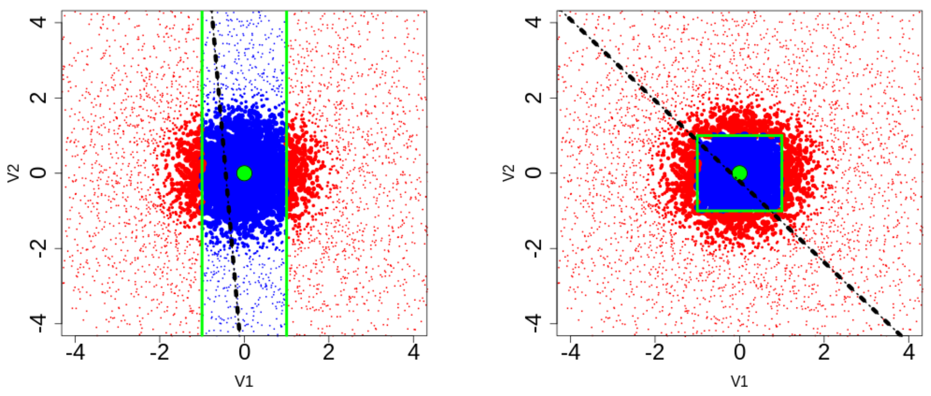
(a) Toy case 1 (b) Toy case 2

Figure 2.1: Toy illustration of Points 3 and 4 (two features). Size of the dots represents the weight assigned on the simulated points by the LIME kernel. Blue points represent class 1 while red points class 2 (all of them small). When learning the probabilities if the gradient in the neighborhood of x_0 (black point) was insensitive of probabilities, predictions of the similar points will not vary much. When learning the labels with LIME and one of the classes dominates the other in the simulation (Fig a), the exponential weighting will not guarantee convergence to a meaningful approximation. When multiple regions lie around x_0 the situation will turn even worse (Fig b).

the true probability field for a certain point of the feature space. It is reasonable to assume, that there exist regions in the feature space that the probability field will be insensitive to changes, i.e. the gradient will be vanishing. This will most commonly happen for data points that lie far away from the decision boundary or when the complex model is highly non-linear. In such cases, when the kernel width σ of equation (2.4) is small, the explainer will assign exponentially larger focus on the insensitive neighborhood (Figure 2.1a). This can be potentially problematic, due to the fact that predicted probabilities of the samples will not have enough variation in the neighborhood to accurately estimate a meaningful direction of importance. The kernel width in such cases should be larger, although as discussed in the previous point, there is no formal rule to evaluate how large it should be for each case.

Point 4 - Learning labels in LIME setting is in general not feasible.

When the explained point lies far away from the decision boundary (according to a metric), cases of the same class as x_0 will receive much larger weight in the loss than opposite cases. If one of the classes of the simulation dominates in number (Figure 2.1a - most points are blue), the solution will be unstable due to the vanishing contribution of the opposite class points in the loss. Since the loss is weighted exponentially in the neighborhood, the linear model will have no reason to learn how to actually discriminate the two classes in an accurate manner. As a result, the solution will be highly unstable when learning the labels. In addition, if multiple decision boundary regions lie in different directions around x_0 (Figure 2.1b) the



(a) 2 mirror regions (b) 4 mirror regions

Figure 2.2: Toy illustration of Points 5 and 6. The decision tree has fitted two (Fig a) and four (Fig b) symmetrical regions to separate the two classes (red and blue). Size of the points indicates the weight assigned by LIME's exponential kernel. The explanation for x_0 (green point) is influenced by the surrounding regions and does not reflect the approximation of a single region.

situation will turn even worse: the linear fit in this case is less meaningful and the solution will be more sensitive to randomness of the simulation step. Learning the probabilities alleviates this issue since points with large contribution to the loss on average have a significant variation on their predicted probabilities. However, in this framework the problem of insensitivity arises for some cases (Point 3).

Point 5 - When a point is surrounded by multiple boundary regions the explanation is a pooled result of all regions

Consider the toy example in Figure 2.2a. A Decision Tree has been employed to fit two parallel to y-axis regions (green lines) to discriminate the blue from the red points. Decision boundary regions of this kind will be called *mirror* regions and their interpretation is that both increasing and decreasing feature V_1 will increase the probability of the red class.

Which is the estimate that LIME will give for the effect of V_1, in the explanation of x_0 (green point)? The solution is represented by the dotted line. LIME's exponential kernel assigns weights that decrease symmetrically towards all directions. Observe that for both increasing or decreasing V_1, the predicted probability of the blue class will decrease in the same manner. The explanation of LIME (dotted line) though, suggests that V_2 also plays a small part in the classification although this is not correct. The reason of this outcome should be attributed to the mirror region: simulated points from the opposite side affect the solution of the explainer.

To motivate even more the impact of this observation, consider the more extreme situation of Figure 2.2b, where the blue class is distributed in the area of a square and x_0 lies in its center. The decision boundary fitted by the tree consists of four different regions (sides of the square). In this set-up, LIME is unable to approximate a meaningful solution (black dotted line). Instead of approximating one of the regions, the algorithm averages the effects of all sides (diagonal dotted line). Thus, LIME explanations are a pooled result of all decision boundary regions, a fact which can lead into misleading explanations. In cases like this, the neighborhood of x_0 cannot be properly linearized. For a more suitable explanation one should attempt to approximate each decision boundary region separately.

Point 6 - Coefficients produced cannot be interpreted as a decision boundary linearization in the general case

LIME is designed to produce sparse explanations. When it comes to the full explanation the estimated coefficients cannot be interpreted as an approximation of the decision boundary. LIME learns the effects on predicted probabilities in the neighborhood of x_0. When x_0 is not close to the decision boundary or when the predicted probabilities do not vary much in the locality of x_0, these estimates do not necessarily agree with the local classification behaviour: in a highly nonlinear framework, the local classification behaviour can be a *pooled* result from multiple regions that might cancel each other in the neighborhood of x_0 (Figure 2.2b). As a result, the fitted linear model will not try to approximate one of the regions or all regions separately but it will rather fit a linear solution that reflects a pooled result from all regions. Thus, LIME does not linearize the decision boundary in the general case but tries to identify important features in the neighborhood. This is not necessarily unwanted, yet the resulting explanation may strongly rely on the randomness of the simulation and on the width of the kernel.

2.4 Motivation of DBA

In this research we would like to urge the need of a direct approximation of the actual decision boundary of the classifier in order to obtain a faithful surrogate explanation. We argue that in complex classification tasks and continuous feature spaces, learning predicted probabilities in the neighborhood can be misleading for some structures (e.g Neural Networks). Moreover not all classifiers are probabilistic, thus we attempt to create a framework where learning the labels is feasible (resolving Point 4).

In the next chapter we will show that in DBA, instances of the opposite class of the explained example also participate in the explanation, providing a solution for Point 1. We will also show that explaining the decision boundary rather than the neighborhood of the explained instance, resolves the kernel width selection problem (Point 2). That is because we sample on the decision boundary, aiming to simulate a set in which both classes are equal in size. We also provide a solution to Point 5, by explaining one single decision boundary region, which in general will be easier to linearize.

Chapter 3

DBA: Theory and Methods

In the previous chapter we highlighted the importance of pursuing a decision boundary approximation rather than sampling in the neighborhood of x_0 to derive an explanation for a classification task. In this research, a new explainability algorithm is developed specifically for binary classification with numerical (standardized) features. The current chapter will explain our approach. In section 3.1 preliminary definitions and concepts are discussed. Sections 3.2 - 3.4 correspond to detailed descriptions of the three main steps of the algorithm, while in section 3.5 an evaluation framework will be presented. Finally, in section 3.6 several diagnostics for DBA are proposed.

3.1 Preliminaries

Consider a binary classification task and a *classifier* $f : \mathbb{R}^p \to \{0, 1\}$ that fits in the training set,

$$\begin{pmatrix} y_1 \\ x_1 \end{pmatrix}, \dots, \begin{pmatrix} y_N \\ x_N \end{pmatrix},$$

where $x_i \in \mathbb{R}^p$ and $y_i \in \{0, 1\}$. The discrimination solution is encoded in a *decision boundary*, i.e. a $p-1$- dimensional surface, which partitions $\mathbb{R}^p$ in two classes. The surface can algebraically be expressed through an equation $G(x) = 0$ for any point $x \in \mathbb{R}^p$. Then the decision boundary can be expressed as,

$$D = \{x \in \mathbb{R}^p : G(x) = 0\}. \tag{3.1}$$

If G is a linear function, then the decision boundary defines a linear hyper-plane. For example, the decision boundary of logistic regression is the hyperplane $\{x \in \mathbb{R}^p : x^T\beta - 0.5 = 0\}$. Here, the estimated vector of the regression weights β represents the feature importance in classification. That is a *universal* coefficient vector for all cases, which represents an explanation for their predictions. Whether this coefficient vector is interpretable for a decision maker, depends on dimensionality, the nature of the data and visualization options.

On the other hand, non-linear classifiers produce non-linear black box decision boundaries. Consequently, there is no analogue of the coefficient vector β, since

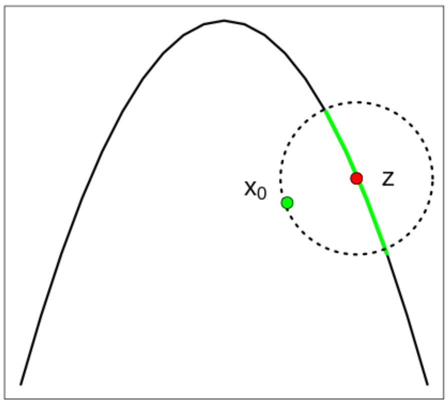


Figure 3.1: Illustration of the most relevant region concept. The explained point x_0 maps to a ball (dotted circle) centered on the closest decision boundary point z. The selected region (green line) is the intersection of the ball with the decision boundary (black curve).

different parts of the decision boundary suggest different effect on the calculated odds. In fact, each region would be approximated by a different linear boundary, suggesting a different vector β.

This work relies on the assumption that for each data instance x_0 to be explained, there exists one region of the decision boundary that applies the most (most *relevant*). We attempt to explain single predictions of complex binary classification models, by applying the following for an instance of interest x_0:

- **Detection** step, which locates the closest decision boundary point of the example to be explained.
- **Simulation** step, which samples instances in a region around this point and labels them according to the complex model.
- **Explanation** step, which fits a linear decision boundary to the simulated data.

The algorithm relies on the choice of a metric to define the concept of *closest*. For this thesis, only Euclidean metric will be considered as a measure of dissimilarity. Assuming Euclidean distances leads in approximating the feature space with a continuous linear metric space, which will not be a sufficient approximation when the true space is highly non-linear. However, in Experiment 4, we will show that approximating a discrete feature space by assuming continuity, will produce valid results. In sections 3.2 - 3.4, the technical details of the three steps are elaborated.

3.2 Detection step

The term *locality* as used by Ribeiro, Singh, and Guestrin 2016, is translated into *relevance* under DBA framework. To formalize this, consider a data instance x_0,

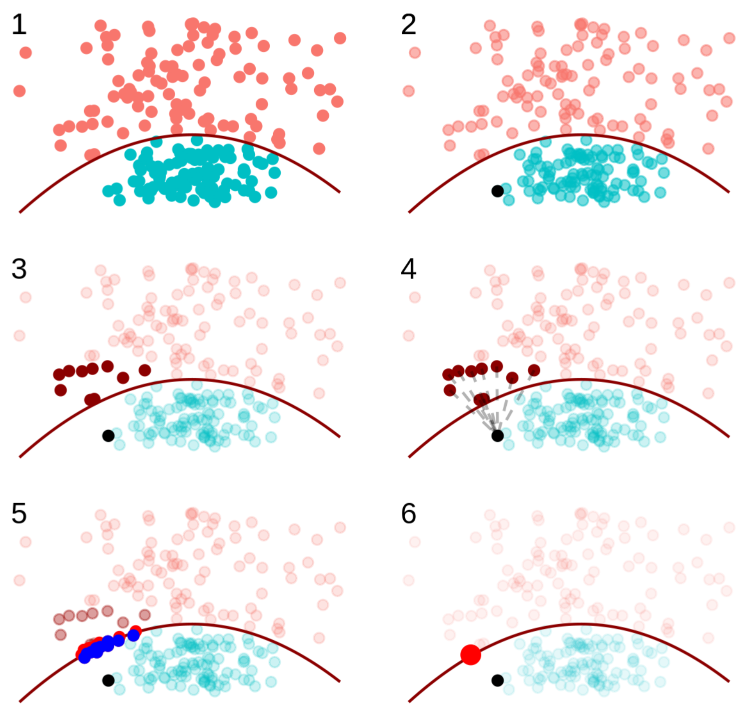


Figure 3.2: Illustration of detection step through a toy example. To detect relevant region of the black point x_0 (2), we select the 10-closest rivals (3). These rivals are bisected with x_0 (4) to produce pairs of points close to the boundary (5). The bisected rival which is closest to x_0 is eventually selected (6)

a decision boundary D and a metric d. If $z = \arg\min_{x\in D}\{d(x_0, x)\}$ is the closest point to x_0 on the decision boundary, then DBA aims to explain the region $D \cap B(x, d(z, x_0))$, where B denotes a hyperball defined by d (Figure 3.1). This will be called *the most relevant* decision boundary region to x_0.

The Detection step of the algorithm selects the most relevant decision boundary region by minimizing the distance $d(x, x_0)$ with respect to x, subject to the constraint $f(x) \neq f(x_0)$. For simplicity we name all points for which this constraint holds, *rivals* of x_0. In other words, we pursue to simulate the closest rival of x_0, a point z which will be assigned as the center of the ball B (red point in Figure 3.1). That is achieved by the following process, developed for this project.

1. Select the K-closest rivals of x_0 from the training set.
2. Collapse all rivals on the decision boundary towards x_0.
3. Select the closest collapsed rival according to Euclidean metric.

The full process is illustrated through a toy example in Figure 3.2. For collapsing the rivals on the boundary (second part of Detection step), we employ a variant of bisection rule, an old-fashioned method of Numerical Analysis used to approximate solutions of non-linear algebraic equations of the general form $f(x) = 0$. Bisection method is a simple iterative algorithm which initializes with a given interval $[a, b]$ in

```
Initialization;
$a$ : a rival of $x_0$;
$b = x_0$;
$\epsilon << 1$;
$\delta >> 1$
while $\delta > \epsilon$ do
    $m \leftarrow 0.5 * (a + b)$;
    if $f(m) \neq f(x_0)$ then
        $a = m$ ;
    end
    else
        $b = m$ ;
    end
    $\delta = d(a, b)$
end
Result: $a$
```

Algorithm 1: Predictive Bisection

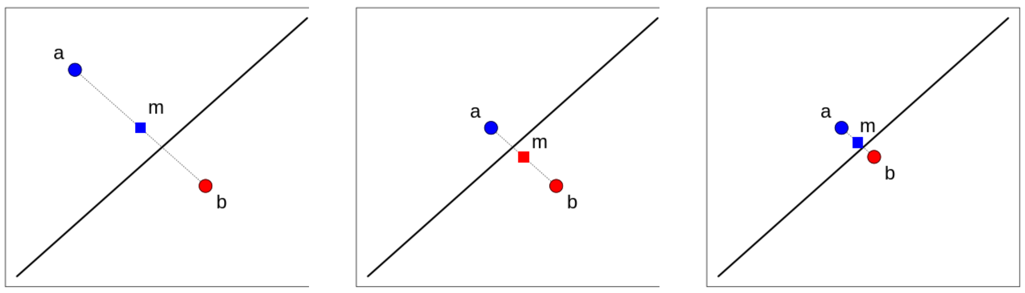
which the solution lies. On each step the average value m, of a and b is computed. Then, Bolzano Theorem is applied on both $[a, m]$ and $[m, b]$ to determine on which interval the solution belongs and set m as either b or a respectively. The process is repeated until $a - b$ is smaller than a specified tolerance ϵ.

For the case of approximating the boundary point z, the Bolzano test is replaced with a simple prediction of the complex classifier to acquire the label of the midpoint m. Initialization involves predicting a and $b = x_0$ with the classifier. Then the condition $f(m) = f(x_0)$ is tested (Algorithm 1). If the condition holds, this means that the midpoint is on the same side of the decision boundary as x_0 so we set $b = m$ to approach the boundary even more. Otherwise, if $f(m) \neq f(x_0)$, a is set equal to m. Repeating the process creates two sequences of points $\{a_k\}$, $\{b_k\}$ which naturally belong on the opposite class and the segment connecting them intersects the boundary. Thus (at least) one true decision boundary point always exists on this segment and the algorithm approximates it. We will refer to this procedure as *Predictive Bisection* method. An illustration with 3 iterations of Predictive Bisection is shown in Figure 3.3.

The whole process is repeated for all K-rivals simultaneously [1]. As K increases, the approximation of z becomes more accurate, due to the fact that the resulting pairs of points will tend to cluster on the boundary and thus build a sufficient coverage of the decision boundary region (Figure 3.2, picture 5). We will call the rival of which the projection is z, the *most relevant training rival* of x_0.

Motivation for employing this procedure lies on the fact that the resulting decision boundary points will carry the information of x_0 and its K-closest rivals on the boundary, thus we ensure that when data lie on a low dimensional manifold, the midpoints will lie on the same manifold: we do not explore all dimensions uniformly but we rather *manipulate* relevant data instances to become as similar as possible

[1]The actual implementation of the process involves vectorization for speeding up.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 3.3: Three iterations of predictive bisection. On each step the midpoint m (square) of a (blue) and b (red) is computed. If the midpoint is predicted to be in the opposite class of x_0 (red point in Fig.a) then we set it as a (Fig.b), otherwise as b (Fig.c). The algorithm converges on a decision boundary point.

with x_0, under the constraint $f(x) \neq f(x_0)$. Consequently, we will explore only features that appear in x_0 and its most relevant training rival. In sparse cases this will decrease the number of features substantially, thus increase the interpretability of the explanation. Moreover, the most relevant training rival can be perceived as a counterfactual explanation, i.e. a data point that if we average x_0 with, it will produce a faster change in the class than moving towards any other training rival. Last we note that the process has a significantly lower time complexity than uniform search in the full feature space.

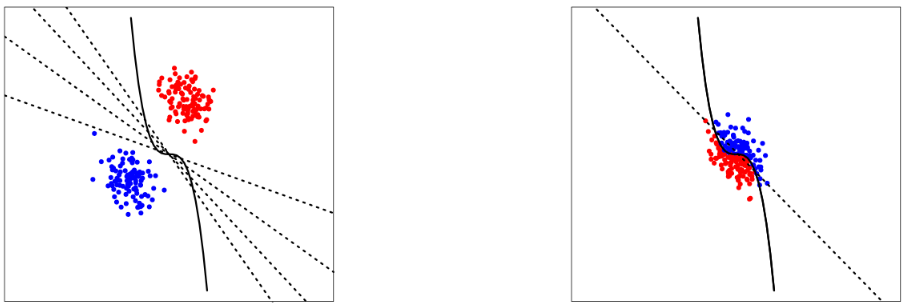
The output of detection step z will be processed by the next step of the algorithm to sample instances around it. The next section will illustrate the theoretical concepts of the sampling procedure (Simulation step).

3.3 Simulation step

Once the region of interest is detected, we would like to generate example cases around it. These cases will be then introduced in the Explanation step to produce the final explanation.

Let us denote with $\mathscr{B}$ the ball $B(z, d(z, x_0))$, where z the output of detection step. A first idea would be to exploit all training samples within $\mathscr{B}$, however if data are sparse or x_0 is very close to the decision boundary then $\mathscr{B}$ might include only a few (possibly zero) training examples other than x_0. Thus, an alternative option would be simulation. Some important characteristics of the simulation set S that we need to consider are the following:

1. *Class balance*, i.e. the proportion of the two classes in the simulation set.
2. *Separability*, i.e. the extent to which the simulated data can be linearly separable. We will call a simulation set *separable* when a linear boundary discriminates the two classes perfectly.



(a) Simulation away from the boundary. (b) Simulation close to the boundary

Figure 3.4: Illustration of the impact of violation of property 2. When there are not enough points close to the boundary (left), all dotted lines are potentially an explainer fit. When a sufficent amount of points are sampled close to the boundary, the linearization stabilizes (right).

3. *Variability*, i.e. the extent to which the simulated examples vary across different dimensions.

Undesirable implications might arise if the simulation does not respect some fundamental properties regarding class balance, separability and variability among others. We discuss four such properties that need to be satisfied to ensure the trustworthiness of the final explanation.

Property 1. *Simulated points are within the ball $\mathscr{B}$.*

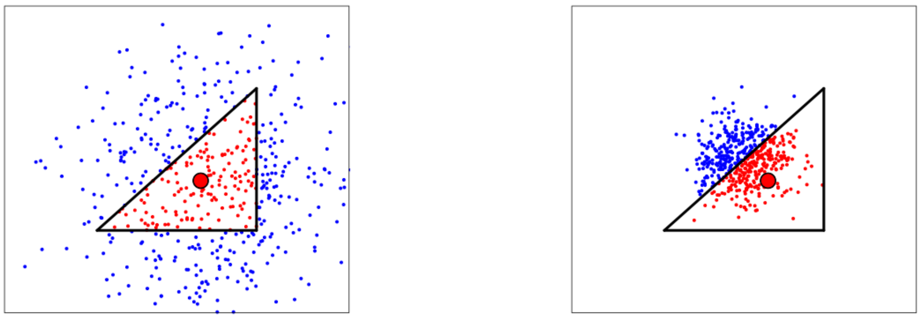
This property highlights that $d(x, z) \leq d(z, x_0)$ should hold for any simulated point x, in order to ensure relevance of the explanation. We would like to simulate all points within the ball, to guarantee that we explain only one specific region. If multiple points lie outside the ball then the solution can be strongly influenced.

Property 2. *There is a sufficient amount of points near the decision boundary.*

If this property is not satisfied then the stability of the solution is not guaranteed. For illustration, consider the simulation set of Figure 3.4a. The dotted lines represent possible linearizations of the relevant decision boundary region. We argue that in order to learn the decision boundary curvature as reasonably as possible, we should sample close to the boundary (Figure 3.4b).

Property 3. *Simulated points, labeled by the predictions of the complex model are reasonably separable and have class balance approximately 50%.*

We argue that the explainer must fit on a (reasonably) separable simulation set in order to provide a meaningful approximation. By *reasonably* we mean that the linear model will be able to produce a proper fit as the one shown in Figure 3.4b. If it is not possible for the linear model to fit properly (Figure 3.5a), then the explanation will not represent a decision boundary approximation but merely a



(a) Non-separable simulation.

(b) Separable simulation

Figure 3.5: Non-separable vs separable simulation. This toy example shows a triangular boundary with x_0 in the middle (red). Simulating centered on x_0 would result in a non-separable set and it is not clear how the linear model would fit. By sampling centered on the closest decision boundary region the linear fit will be more meaningful, but relevant only for points separated only from that side of the triangle.

pooled result of an abstract number of regions (see also section 2.3.2). It is argued that only by approximating a single region of the boundary will the additive model capture a meaningful solution (Figure 3.5b). As the degree of separability decreases, so does the trustworthiness of the explanation.

Regarding class balance, (i.e. $P(C = 1|S)$) a linear boundary would correspond to 0.5. As non-linearity increases the class balance of the simulation is expected to change. An unbalanced simulation design might influence the estimates for some features of one of the classes. In general, when balance strongly deviates from 0.5 then the explanation should not be trusted. A trustworthy explanation is an explanation for which the approximated behaviour is nearly linear.

Property 4. *The simulation has equal variability across all explored dimensions.*

It is important that we simulate across all dimensions of interest with equal variability, since the estimates of the explainer depend on the sample. If there are dimensions that are not explored sufficiently, the explainer will underestimate their importance and the explanation will not be accurate. It is also of great interest to examine whether the simulation should cover the whole volume of $\mathscr{B}$ or if it suffices to sample close to z, i.e. the decision boundary point approximated by detection step. In Experiment 3 we will explore the impact of this matter on the final explanation. We will show that sampling with a higher concentration in the center of the ball, will result in the same solution by means of directions of importance, but different intensity of estimates.

In the remainder of this section we will introduce the simulation procedure of DBA and motivate its selection. In general there are multiple options that one may want to consider and research for simulation. Uniform sampling within the ball $B(z, d(z, x_0))$ would be a primary option, however filling a p-dimensional ball with points when p is large, can be computationally intensive (e.g with rejection sampling). For low dimensional cases an option is to sample from multivariariate gaussian distribution. However, simulating from a Gaussian in a higher dimensional space will result on most points having a large distance from the center (*rare* events are not that rare as dimensionality increases) (Dasgupta 2013). That is not desirable since we would like to sample close to the decision boundary point, to avoid violation of Property 2. A solution could be to decrease the variance components of the covariance matrix Σ, nevertheless there is initially no formal rule to evaluate the values for these components.

Although future research should investigate different simulation alternatives, in this thesis we propose an approach which attempts to respect properties 1-4 as much as possible. We start with the following definiton.

Definition 1 (Convex Hull)**.** *Let* $V = \{V_i\}_{i=1}^{n} \subset R^p$ *be a set of* n *points in* p*-dimensional space. The* ***convex hull*** *of points* $V_1, .., V_n$*, is the set* $H_w = \{\sum_{i=1}^{n} w_i V_i : \sum_{i=1}^{n} w_i = 1, w_i \geq 0\}$ *i.e. all convex combinations of points in* V*.*

A convex hull is by definition a convex set. This means that if we assume a (random) convex combination of any points of the hull, it will lie within the set. The idea behind the algorithm implemented by Simulation step is the following:

1. Select symmetrical points on the surface of $\mathscr{B}$ to define a hyperpolygon centered on the decision boundary (**Vertex creation**).
2. Sample from the convex hull of the vertices with random weights to generate data in the polygon (**Weighting procedure**).

In the following, the technical details of the two parts are elaborated.

3.3.1 Vertex Creation

In the default version of the algorithm (on which most experiments run) the vertices are created by the following procedure: For each feature i set $V_{1i} = z + r$ and $V_{2i} = z - r$, where $r_j = 0$ if $j \neq i$ and $r_j = d(z, x_0)$ if $j = i$ (i.e. increase and decrease only one coordinate by the distance from the decision boundary). Thus $2p$ points are created symmetrically on the surface of $\mathscr{B}$ (Figure 3.6). One could argue that if we vary only one feature out of thousands, there will not be much change in the classifier's odds. The experiments of this thesis will illustrate, that this argument is not correct when we vary features of a decision boundary point: in this state of uncertainty if an important feature varies there will be a significant change in the odds.

However, as it will be shown in Experiment 3 (section 4.3), due to the *curse of dimensionality*, simulating from the convex hull of a large number of vertices will result in high concentration in the neighborhood of z. To correct for that, we propose two different high dimensional vertex simulation alternatives that will be compared in Experiment 3.

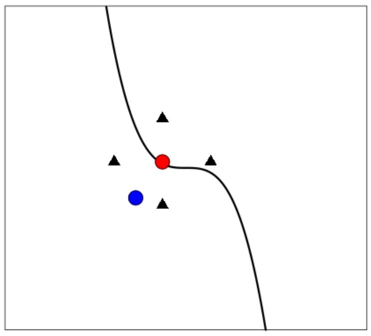


Figure 3.6: Illustration of vertex creation (default method). To simulate across a relevant region of x_0 (blue), we locate the closest decision boundary point z (red). The vertices (black triangles) are created by adding and subtracting the euclidean distance $d(z, x_0)$ from each coordinate of z.

In the first alternative, the number of vertices/dimensions explored is reduced by assuming *interpretable data representations.* For instance, image data can be reduced through superpixel segmentation. In this version, two vertices per superpixel (batch) are created. In contrast to LIME, DBA does not drop the assumption of continuity and increases/decreases simultaneously the values of each batch such that resulting vertices lie on the ball. For segmenting image data, the SLICO method is used (Bakkari 2015).

The second alternative involves assuming more distant vertices, to result in a larger convex hull (such that it results in sampling in the ball). Same as the default method we create $2p$ vertices. The correction tested in the experiments will be to set $V_{1i} = z + \sqrt{p}r$ and $V_{2i} = z - \sqrt{p}r$, where p is the dimensionality. This was motivated by the fact that the distance between two vertices of a hypercube increase as $\sqrt{p}$. In experiment 3 we will explore how these two modifications affect the explanation and whether they are desirable in the DBA framework.

To encounter interpretability issues when the number of features is very large we propose another extension of this step which applies to sparse data. We decrease the dimensionality by considering only the non-zero elements of the decision boundary point z and create vertices only for these points. Since the bisection method transfers the information of the most relevant rival to the decision boundary, the explanation will refer to x_0 and the corresponding rival (and points alike them). Features not present in these two points are not considered.

The vertex creation framework is a fast alternative for uniformly simulating points that by definition belong in $\mathscr{B}$ due to convexity. More importantly the vertices can represent batches of features rather than single features. For instance we could assume that the vertices are all possible *masks* (filters) of an image and then simulate random convex combinations of these masks (see Chapter 5, section 5.1.4, Figure 5.2). These facts motivated our selection of this approach over other uniform simulation approaches.

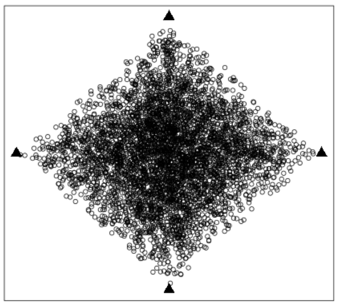


Figure 3.7: Illustration of the resulting simulation by sampling with the described procedure. More points are sampled close to the center, than the boundary of the polygon.

3.3.2 Weighting procedure

Once the vertices have been created, the next step is to sample from their convex hull by assigning a vector of weights w such as $\sum_i w_i = 1$ and $w_i \geq 0$.

For a uniform sampling from the convex hull of p vertices, we employ the following procedure (Devroye 1986):

- Sample from $U(0,1)$, $p-1$ numbers $u_1, u_2, ..., u_{p-1}$.
- Order them in place.
- Set $u_0 = 0$ and $u_p = 1$.
- Let $w_i = u_i - u_{i-1}$ be the weights for sampling a convex combination of the vertices.

If each u_i is drawn from $U(0,1)$, w_i have the property $\sum_{i=1}^{p} w_i = 1$. The weights w_i are then used to draw a sample $x = \sum w_i V_i \in H_w$. The method will result in uniform sampling in the convex hull of the vertices, under the assumption that the vertices are affinely independent. Apparently under the set-up described in section 3.3.1 this assumption is violated and in general the sampling will not be exactly uniform. Specifically the sampled points will show the tendency to cluster towards the center of mass of the polygon (in this case the decision boundary point z). However this is not necessarily unwanted since we guarantee that there is a sufficient amount of points in the decision boundary. Figure 3.7 shows a 2-D simulated example of the process. As it can be seen, more points are sampled in the center than the boundary of the polygon. The complete procedure is also shown in Algorithm 2.

3.3.3 Labeling the simulated data

After a simulation set S is obtained, we label it with the predictions of the complex model. Augmenting the simulation set with the prediction vector $f(S)$, results in the input of the explanation step. The predictions in the default set-up are the labels, however for probabilistic classifiers it is also possible to augment the simulation with

```
Initialize: $S = \{x, x_0\}$
for $k$ in 1:B do
    1. Sample $u_1, u_2, ..., u_{p-1} \sim U(0,1)$ and sort in increasing order.
    2. Set $u_0 = 0$ and $u_p = 1$
    3. Set $w_i = u_i - u_{i-1}$
    4. Set $x = \sum w_i V_i$
    5. $S = S \cup \{x\}$
end
Result: $S$, $f(S)$
```

Algorithm 2: Convex Hull Sampling

the predicted probabilities and employ a regressor as an explainer. Experiment 2 will study the impact of such a modification. Theoretically, since the simulation will tend to be reasonably separable, the probability field would be easier approximated by a linear model than in LIME framework.

In section 3.3 we have described the Simulation step of DBA. The whole process is summarized in Algorithm 2. In the next section of this chapter the Explanation step is discussed.

3.4 Explanation Step

The Detection and Simulation step yield a labeled simulated dataset with feature values within the constraint domain $\mathscr{B}$. The problem now drops into fitting an explainer g , i.e. a linear boundary in the simulated sample to *learn* the predictions of the complex model. Thus, g will be trained on the output of the Simulation step, $(S, f(S))$.

In the experiments of this thesis two different linear classifiers [2] will be introduced as explainers, L_2 Logistic Regression and Partial Least Squares Logistic Regression. We briefly discuss these models in the sections below.

3.4.1 L_2 penalized Logistic Regression (Ridge)

The log-likelihood of Logistic Regression can be augmented with a L_2 (quadratic) penalty term, $\frac{1}{2}\lambda\beta^T\beta$ to encounter *supercollinearity* which arises when $n << p$. The penalty term will restrict the estimates in a constrained (spherical) domain. Minimization of the negative log-likelihood is performed via Newton-Raphson algorithm. For this thesis we will employ the results of (Rosset, J. Zhu, and T. Hastie 2003) that for separable data, Logistic Regression with vanishing amount of ridge regularization will give the hard margin SVM solution. Thereby, a fixed penalty of 0.001 is introduced in all experiments.

[2]When learning the probabilities the corresponding linear regression models are employed.

3.4.2 Partial Least Squares Logistic Regression

In 1985, Herman Wald introduced Partial Least Squares (PLS) (Herman 1985), an alternative version of Principal Component Regression. The algorithm employs dimensionality reduction to compute the k orthogonal latent variables (principal components) that maximize correlations with the response. In addition correlations of the original variables with all components are computed. As a result the variable weights of the original features are estimated. The choice of k, successfully results in natural regularization of the model. For classification purposes, two main options are available: Partial Least Squares Discriminant Analysis and Generalized Partial Least Squares Regression, concept similar to Generalized Linear Models. For this thesis the latter is employed.

Under the role of the explainer, PLS is in general able to provide more stable and accurate estimates of the coefficient vector β, for high dimensional cases where (super-)collinearity occurs (Herman 1985). The method is very popular in chemometrics where sample size is in general small compared to the dimensionality. This advantage can be a benefit for an explainer which fits in a small sized simulation set. This way estimating in high dimensions would be more feasible if we consider computation limitations. Another advantage of employing PLS as an explainer for large p is the option of visualization of the dimensionally reduced simulation in the first 2 computed components (see section 3.6).

3.5 Evaluation process

In this section we will propose a framework for evaluating explanations. The process applies to probabilistic classifiers, where predicted probabilities are available.

Let β be the local explanation of x_0 and assume that $f(x_0)$ is the class of reference. The explanation corresponds to a decision boundary region and β is the orthogonal vector on the linear hyperplane which approximates it.

It is crucial to explore whether the explanation implies a *change of class*. Assuming a grid of values we can compute a *field* of predicted probabilities of the complex model on the grid. In an optimal case scenario the direction $v = -\frac{\beta}{||\beta||}$ should maximize the gradient of the field and point-out the fastest way to the decision boundary. Hence, starting from x_0, a step h towards v should decrease $P(C = f(x_0)|x_0)$.

However it is not clear how large should be the step taken. A very small step might not produce change in the model's odds if the local gradient is not very steep, still the explanation could be a meaningful approximation. Moreover a local change in the odds does not always imply a meaningful classification approximation, since the explainer could have learned random fluctuations of the probability field. On the other hand, a large step might cross the decision boundary, yet lead the point to the same class: in the direction of the explanation vector, multiple parallel regions might be encountered, where class labels change periodically (Figure 3.8). As a result, moving x_0 will not produce a change in the odds, however this does not mean that the region has not been approximated correctly (the green line in Figure

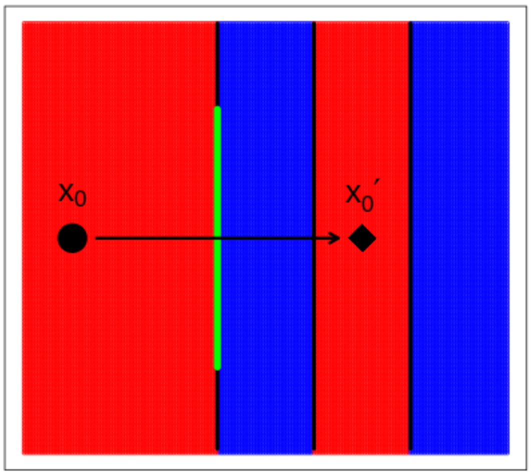


Figure 3.8: Illustration of the inefficiency of evaluating explanations with a single step h. The explained instance x_0 is moved towards the explanation direction (black arrow). If the step is large then $x_0^{'}$ could result in the same class when parallel regions are present.

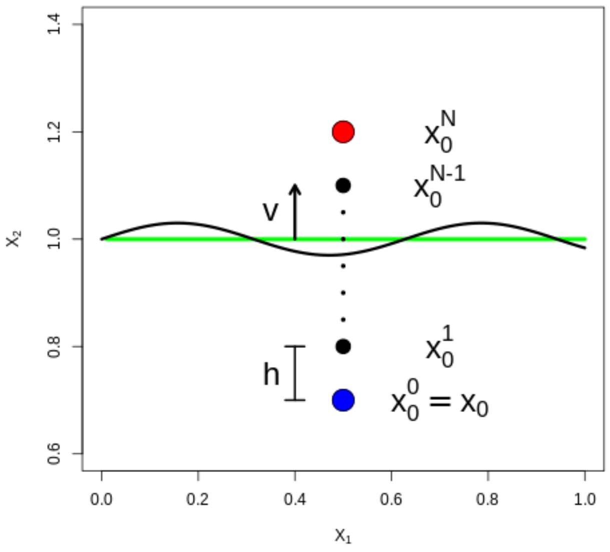


Figure 3.9: Illutration of evaluation procedure through a toy example. The decision boundary (black curve) has been approximated by an explainer (green line) that is associated with an orthogonal direction v (black arrow). Moving x_0 (blue point) towards v with step h creates the sequence of points x_0^k, of which the class posteriors can be obtain through the complex model.

```
**Initialize:** $path = \{P(C = f(x_0)|x_0)\}$
$0 < h << 1$
$x_0^0 = x_0$
**for** $k$ in 1:N **do**
    **1.**. $x_0^k \leftarrow x_0^{k-1} - h\beta/||\beta||$
    **2.** $path \leftarrow path \cup \{P(C = f(x_0^k)|x_0^k)\}$
**end**
**Result:** path
```

Algorithm 3: Evaluation process

3.8 approximates the decision boundary region sufficiently).

Thus, we pick a small step h and apply the transformation $x_0^k \leftarrow x_0^{k-1} + hv$ repeatedly (Figure 3.9). We validate explanations in an iterative process where on each iteration k, we predict the probability $P(C = f(x_0^k)|x_0^k)$ (Algorithm 3). As a result, the probability path of x_0 is monitored. We argue that this is a safer approach to evaluate explanations than moving x_0 with a single step h or *masking* [3] like Ribeiro, Singh, and Guestrin 2016 suggest, to avoid pathological cases such as Figure 3.8. Moreover in the framework we propose, we can draw conclusions about the linearity of the probability field by plotting the path.

This evaluation process will be employed in Experiment 2 to compare DBA with LIME. In Experiment 3 (MNIST) the method will be introduced again to evaluate performance of the three different vertex creation alternatives of DBA.

3.6 Diagnostics

In this final section of the chapter we propose several diagnostic measures that can provide insight for the performance of the various steps of DBA. There are many things that we have to consider for the explanation to be trustworthy, such as simulation class balance, the explainer's fit and the distance from the decision boundary. We summarize these measures in the following paragraphs.

3.6.1 Simulation class balance

We define class balance as $b = \frac{1}{n}\sum_{i=1}^{n} I(x_i)$, where x_i is a convex hull sample, n the simulation size and $I(x) = 1$ if $f(x) = f(x_0)$ and $I(x) = 0$ otherwise. That is, the proportion of simulated points that belong on the same class as the example. Assuming that we simulate uniformly, a class balance close to 0.5 corresponds to a (locally) linear decision boundary. A low balance means that most points simulated are predicted on the opposite class of the example, while a balance close to 1 on the same class. In general a balance between 0.3 and 0.7 should be considered acceptable. However an extreme balance can be attributed either to insufficiency of the simulation process or high non-linearity of the decision boundary (e.g. small over-

[3]For example, an image can be masked with a filter associated with the explanation to reduce the intensity of its pixels by a certain percentage.

fitted regions or multiple parallel regions). Either way an extreme balance implies that an insufficient fit will follow and explanations should be treated with caution.

3.6.2 Explainer faithfulness

Under the framework in which the explainer learns the labels, the faithfulness is defined as $\frac{1}{n}\sum_{i=1}^{n} L(f(x_i), g(x_i))$ where $L(x, y) = 1$ if $x = y$ and 0 otherwise, g the prediction function of the explainer and $x_i \in S$. That is the proportion of simulated cases on which the explainer agrees with the complex model. A high faithfulness does not always imply a correct result: if the balance of the simulation is close to 0, then a faithfulness of 1 means nothing more than the explainer classifying all instances in one class because of the unbalanced simulation design. When balance is close to 0.5 and faithfulness is high it can be claimed that the decision boundary is locally linear and the explainer approximates successfully.

3.6.3 Distance from the boundary

The distance of x_0 from the decision boundary, estimated by $d(z, x_0)$ is a crucial parameter that should be also considered. When the point is close to the decision boundary it means that we can apply a small variation to force it to change class. Moreover, the corresponding region will be approximately linear. When the point lies far away from the decision boundary and the boundary is not globally linear, this means that multiple regions should be explored in order to capture the complete picture. To employ the distance from the boundary as a diagnostic measure we should scale with the average distance between two points in the training set.

3.6.4 Local gradient

The Gateaux derivative of the probability field,

$$dF(x_0; v) = \lim_{h \to 0} \frac{P(C = 1|x_0 + hv) - P(C = 1|x_0)}{h} \tag{3.2}$$

can be approximated by $(P(C = 1|x_0^1) - P(C = 1|x_0))/h$ for a small h. This measures the effect of moving in the neighborhood of x_0 in the direction of the explanation. Although the local gradient does not always coincide with the average gradient of the whole path it is useful to know if it does. It is argued that h should depend on dimensionality, since distances become larger as p increases. In the experiments we apply the rule $h = 0.001p$.

3.6.5 Dimensionally reduced explainer

As discussed in section 3.4.2, in high dimensional cases Partial Least Squares will dimensionally reduce the simulation, such that the correlation with the log-odds of the predicted classes is maximized. As a result, a visualization of the first two components is possible. We may color the projected points with the predictions of the complex models and inspect if the process simulated a separable uniform simulation indeed. This will be a visual assessment of the algorithm in high dimensions. The dimensionally reduced linearization (projected line) can also be plotted, since the effect of each component in the odds is computed by PLS.

3.6.6 Proportion of Variance Explained

A measure of fit calculated by PLS is R_Y^2, i.e. the proportion of variability of the response Y that a number of components can explain. For instance, when two components result in $R_Y^2 = 1$ it will imply that these components explain 100% of the variance of Y. An explainer which produces high values for this measure has identified the directions of separation succesfully. Whether these directions are correct or not should be evaluated with processes such as Algorithm 3.

3.7 Overview

In this chapter we illustrated the fundamental principles of Decision Boundary Approximation framework. In Detection step the algorithm bisects the K-closest rivals with x_0 to obtain z, an estimate for the closest decision boundary point of x_0 (Algorithm 1). Simulation step creates $2p$ vertices by adding/subtracting $d(z, x_0)$ on each coordinate of z. It then employs convex hull sampling to simulate data in the hyper-polygon defined by the vertices (Algorithm 2). This will create a sample centered on the most relevant region of x_0. After obtaining the predictions of the complex model on the simulation, an explainer fits a linear hyperplane to approximate the decision boundary. The whole process is summarized in Algorithm 4.

In the next chapter, experiments will test the assumptions made in this chapter as well as the feasibility of the algorithm in practice.

```
(I) Initialization (Detection step);
$R_k(x_0)$: the K-closest rivals of $x_0$
for $i$ in $1:K$ do
    $a_i \leftarrow$ Predictive Bisection of $R_i(x_0)$ with $x_0$
end
$z = \underset{i}{\text{argmin}}\{d(a_i, x_0)\}$
(II) Initialization (Vertex creation);
$V = \varnothing$
for $i$ in $1:p$ do
    $v_1 = v_2 = z$ ;
    $v_1[i] = z[i] + d(z, x_0)$;
    $v_2[i] = z[i] - d(z, x_0)$;
    $V = V \cup \{v_1, v_2\}$;
end
(III) Initialization (Simulation step);
$S = \{z, x_0\}$
for $k$ in $1:B$ do
    1. Sample $u_1, u_2, ..., u_{p-1} \sim U(0,1)$ and sort in increasing order.
    2. Set $u_0 = 0$ and $u_p = 1$
    3. Set $w_i = u_i - u_{i-1}$
    4. Set $x = \sum w_i V_i$
    5. $S = S \cup \{x\}$
end
(IV) Predict $S \rightarrow$f(S)
(V) Fit explainer
Result: $\beta = \underset{\beta}{\text{argmin}}\{L(\beta^T S, f(S))\}$
```

Algorithm 4: Decision Boundary Approximation

Chapter 4

Experiments

In this chapter the algorithm will be tested through a series of experiments of increasing dimensionality. We begin with a simple 2-dimensional case (Experiment 1) followed by a comparison with LIME for low dimensionalities (Experiment 2). The algorithm will be also tested for sparse high dimensional data in Experiment 3 (MNIST) and Experiment 4 (Naive Bayes SMS classification).

4.1 Experiment 1 - A 2D simulated case

In this experiment the algorithm will be tested in a toy 2-D simulated case where the quality of the explanation can be assessed visually. Experimental purposes are summarized as follows:

- Illustrate the basic operations of the algorithm for boundaries of various complexities.
- Show a visual representation of the solutions to conclude whether explanations are meaningful.
- Investigate the effect of boundary complexity on stability of the estimates, class balance of the simulation and faithfulness of the explainer.
- Illustrate pathological cases where the algorithm fails to output a meaningful solution.

4.1.1 Data

The algorithm is tested on the *moons* data simulated with the use of scikit-learn package in Python 3.7. This results in a toy 2-dimensional dataset with 200 observations of two classes (Figure 4.1). The population of each class is distributed as a *moon*, thus a non-linear boundary is in general expected to outperform a linear choice.

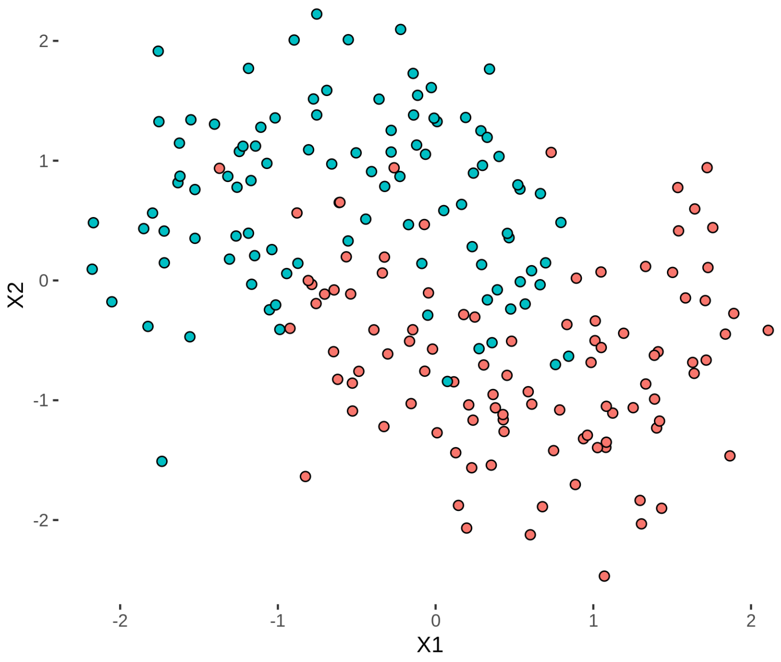


Figure 4.1: Simulated data of the experiment. The “moon” shape of the classes will result in general into curved boundaries.

4.1.2 Classifiers

An SVM with a radial kernel and three different k-nearest neighbor classifiers (k = 15, k = 10, k = 1) fit in the moons simulated set.

The smooth SVM boundary contrasts to the non-differentiable K-NN solutions of which the complexity increases with decreasing K. In any case all resulting boundaries are non-linear and thus the explanation will vary among instances.

4.1.3 Experimental set-up

In order to fit the experimental purposes, 10 representative instances will be explained for all models. For each instance, an estimate for the expected coefficient values and their standard error will be obtained with 100 Monte Carlo repetitions. The class balance and faithfulness on each replicate are also recorded. For studying stability with varying simulation size N_{sim}, the whole process runs for $N_{sim} \in \{50, 100, 500, 1000, 2000, 4000, 6000\}$.

Vectorized Predictive bisection runs on the 50-closest rivals with a tolerance of $\epsilon = 10^{-6}$. For this experiment a Ridge classifier will be employed as an explainer with a fixed penalty $\lambda = 0.001$ (see section 3.4.1).

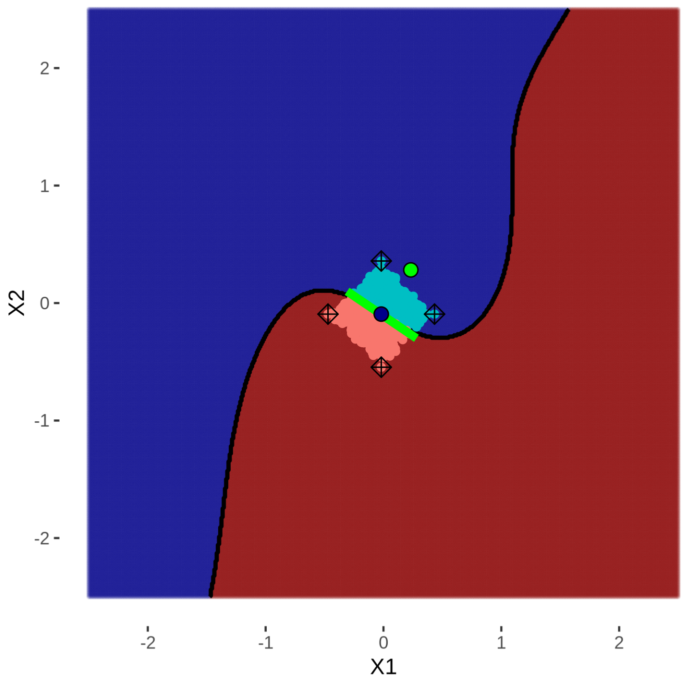


Figure 4.2: Illustration of DBA explanation procedure through a toy 2-D example. The example to be explained (green) is bisected with its 50 closest rivals to yield the closest decision boundary point (”darkblue”). Sampling from the convex hull of the vertices(square points) results on the simulation set which is labeled by the complex model (”red” and ”blue”). The green segment represents the decision boundary linearization learned by the explainer.

4.1.4 Results

Basic function - Illustration

Figure 4.2 depicts the resulting process of explaining one of the 10 instances (green point) for the SVM case. To enhance visualization, the training set is not shown. Detection step returns an approximation for the closest decision boundary point $z = (z_1, z_2)$ to this instance (darkblue). The square points represent the four simulated vertices i.e. $V_1 = (z_1 \pm r, z_2)$ and $V_2 = (z_1, z_2 \pm r)$, where r the euclidean distance $d(z, x_0)$. Sampling from their convex hull results on the simulation shown with light red and light blue, where color represents the predictions of the complex model. Eventually, the explainer learns a linearization (green line) that minimizes a loss which involves the complex model’s predicted labels.

The plot makes clear how the algorithm operates: it attempts to linearize a part of the decision boundary which is returned as relevant from the Detection step. That is not (always) in agreement with LIME approach which maximizes the faithfulness in the neighborhood, rather than a ball of relevance. Thus, DBA explanation is in a sense biased towards the direction which minimizes $d(z, x_0)$, rather than the average discrimination rule.

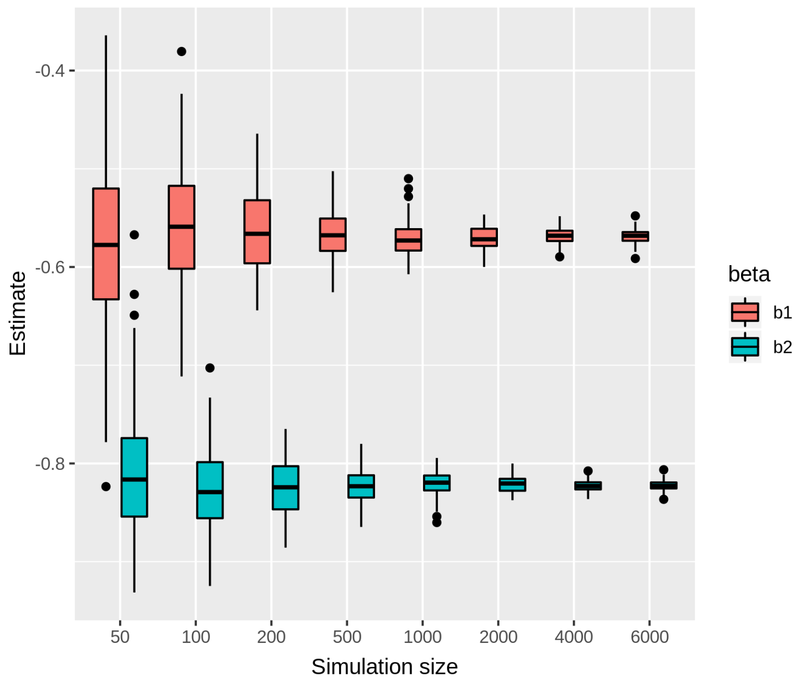


Figure 4.3: Monte Carlo estimates for the two coefficients of the SVM decision boundary linearization for the illustrated example. With a small simulation size the ordering of the estimates is not guaranteed. Increasing the size stabilizes the solution.

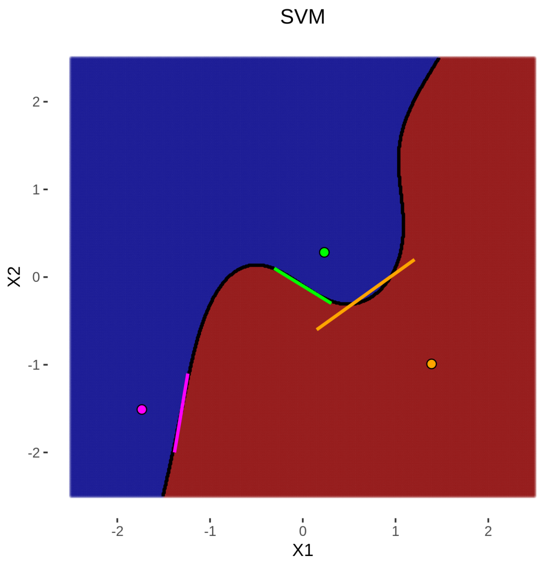
Estimates

The slope of the linearization shown in the previous visualization, implied that for this instance, feature X_2 is more important than X_1 although both features significantly contribute to the classification result. In this paragraph we will show how stable are the estimates for this specific case as N_{sim} increases.

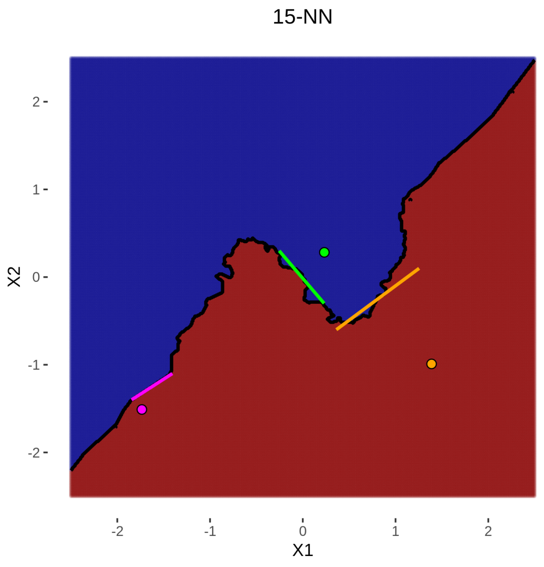
The boxplot in Figure 4.3 shows the coefficient estimates for both features. The plot implies that although the estimate range is reasonable for all sizes, for $N_{sim} < 100$ the actual ordering of the features is not guaranteed (whiskers of the boxes overlap). For $N_{sim} > 100$ estimates stabilize on the true values with a reasonably small standard deviation. Sampling more than 4000 points does not seem to significantly increase performance for this case. It is concluded that for this instance, classifier and dimensionality, increasing simulation size results in increasing stability of the estimates. Simulation sizes $100 < N_{sim} < 1000$ seem to be sufficient for providing a representative explanation.

Effect of complexity

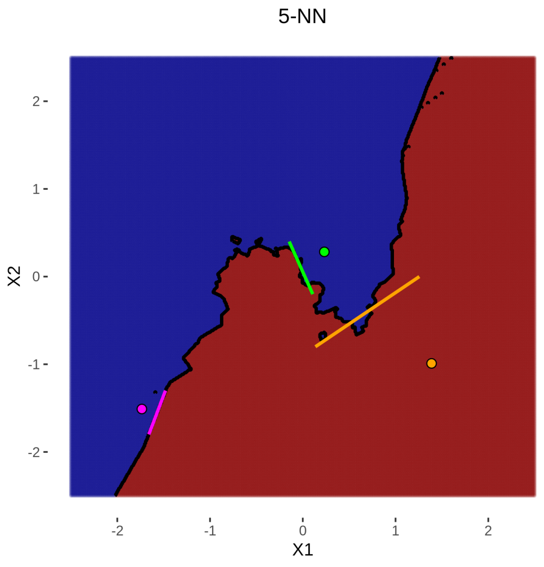
Illustration of the previous paragraph has provided evidence to support that DBA is capable of successfully locally linearizing the smooth Supported Vector Machine decision boundary. It is of interest to explore whether this holds for less smooth decision boundaries where the class balance of the simulation is not always guaranteed. Figure 4.4 shows explanations for 3 explained cases for all models ($N_{sim} = 1000$). It is clear that for these cases the explanation linearizes a relevant region of the



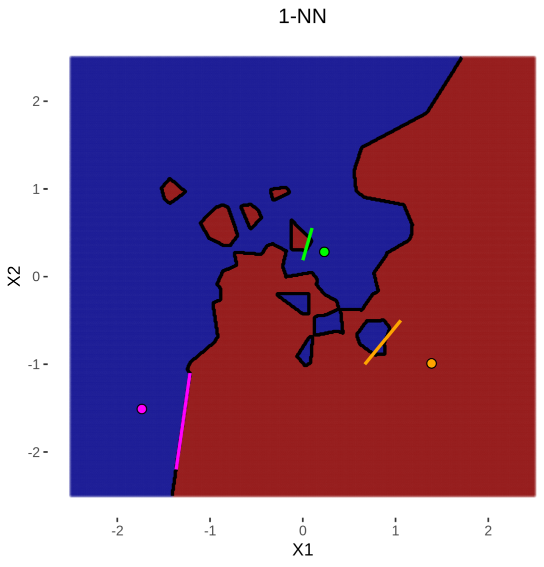
(a) Radial SVM.



(b) 15-Nearest Neighbors.



(c) 5-Nearest Neighbors.



(d) 1-Nearest Neighbors (overfitting).

Figure 4.4: Decision boundary linearization of the SVM and the 3 versions of KNN for the 3 representantive points ($N_{sim} = 1000$). DBA seems to capture meaningful solutions.

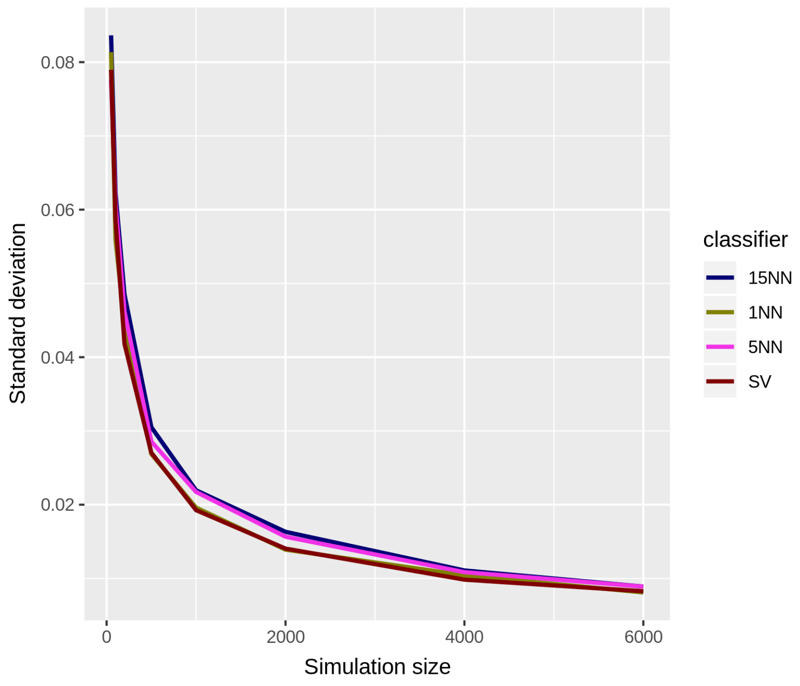


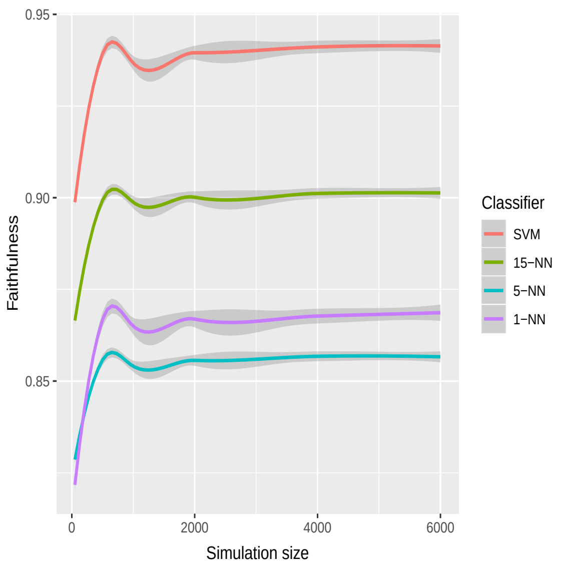
Figure 4.5: Average (over 10 explained instances) standard deviation of the estimates produced by DBA for the 4 models. 15-NN and 5-NN produces slightly more unstable results than SVM and 1-NN. Stability increases with increasing simulation size for all cases.

decision boundary and attempts to approximate the complex boundaries. However the approximation of *spiky* regions that appear in the Nearest Neighbors models is potentially unstable especially for the cases shown as *green* and *orange*. For the magenta case the boundary of all classifiers is locally linear and easier approximated by DBA. More interestingly for the *green* point of 1-NN, DBA has explained a small region caused by overfitting. However an overfitted classifier is not desirable. If features had a natural meaning, then comparing the explanations of this point for all models might contribute to an empirical model selection.

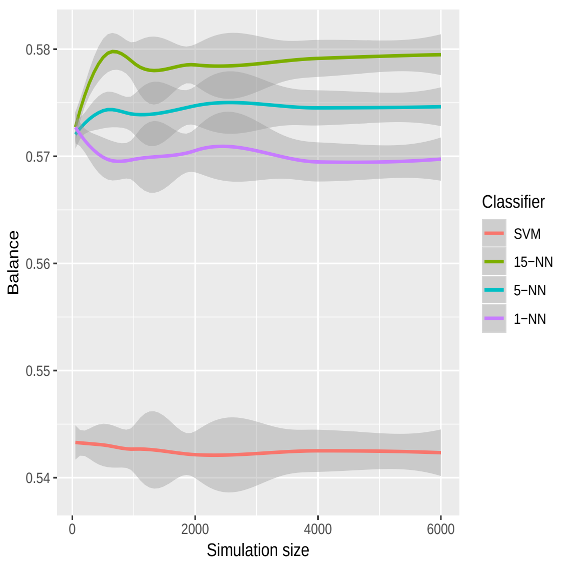
For a more formal stability evaluation, the average (over all 10 instances) standard deviation of the estimates is calculated for all models and sample sizes. Results are summarized in Figure 4.5. All lines decrease exponentially fast with increasing simulation size to converge towards zero, indicating that large N_{sim}, yields on average stable estimates. Surprisingly, the lines for SVM (brown) and 1-NN (green) coincide and are steeper compared to the 15NN (blue) and 5NN (pink). This could be attributed to the fact that the two latter appear more *spiky* local regions than the two former (Figure 4.4). 1NN might yield in general a more complicated decision rule, however in many cases locally linear. For a more concrete conclusion the experiment should be repeated for a larger number of explained instances.

Balance and Faithfulness

The local non-linearity indicated in the above paragraph can be also highligted if one studies the relation between simulation size and balance/faithfulness of the



(a) Average Faithfulness



(b) Average Class balance.

Figure 4.6: Average (over 10 instances) class balance of the simulation and faithfulness of the explainer for all models. Grey areas represent confidence intervals estimated with 100-MC replicates. Both measures score on average better values for SVM compared to the K-NN models.

algorithm. In Figure 4.6a the average faithfulness over the 10 explained instances is plotted along with its Monte Carlo (MC) estimated standard deviation. Linearizing the smooth SVM results on average in more faithful solution than Nearest Neighbors, followed by 15-NN. On the other hand for 1-NN case results are more faithful than 5-NN for these 10 instances. Similar conclusion is drawn from Figure 4.6b where the class balance of the simulation is shown in the same manner. Simulation for the SVM is more balanced (closer to 0.5) than NN models.

4.1.5 Pathological cases

The previous paragraphs suggest that DBA can provide meaningful explanations for some cases. We have shown that the algorithm can also provide stable results for 1-NN despite its overfitted boundary. However this is not always the case: overall such boundaries is possible to be explained but outlying cases might arise. In this section a pathological case will be illustrated and discussed.

Figure 4.7 shows a case where a point lies far away from the 1-NN decision boundary while multiple small regions lie in between. This results in a highly unbalanced simulation set (class balance 0.11) with a high faithfulness (0.89). Specifically, the explainer has classified all points as *blue* in order to maximize the agreement with the complex model. That is not a meaningful explanation, however the low balance can indicate the plausibility of such an effect to be encountered. This illustrates that explanations of low balance should not be trusted, not even if they have high faithfulness.

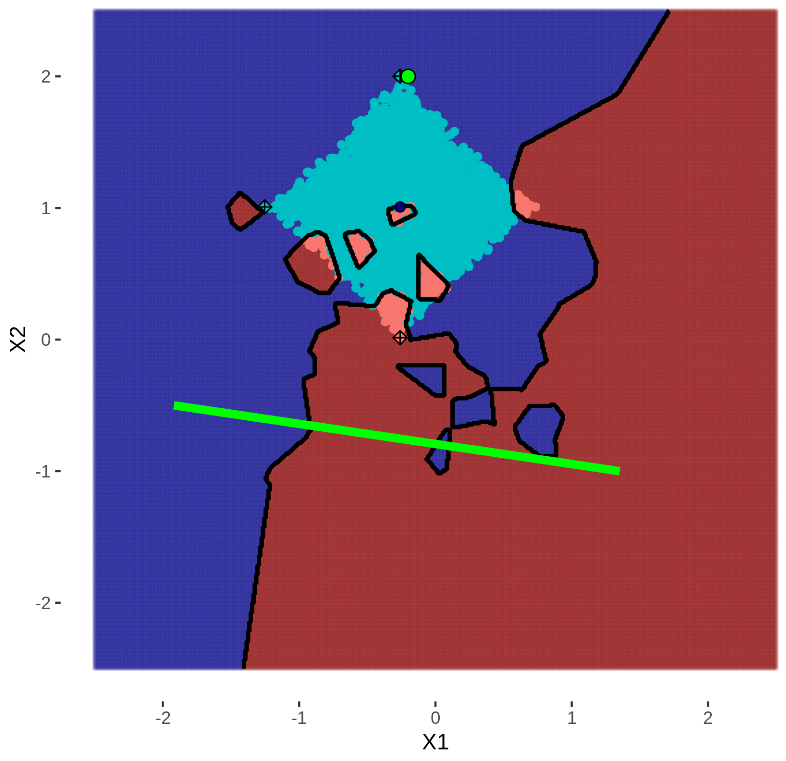


Figure 4.7: Illustration of a case where DBA fails to capture a meaningful solution. The explanation refers to an instance which lies far away from the decision boundary, while multiple small regions exist in the space between. The unbalanced simulation design leads the explainer to classify all points in one class.

4.1.6 Experiment overview

This experiment has revealed the basic operation of DBA through a toy 2-dimensional case. It can be concluded that increasing complexity of the boundary affects the stability of the explanation on a small degree, however the algorithm can provide faithful local explanations for cases where the decision boundary is locally linear. When multiple regions lie around the point the explanation is not complete, yet it refers to a specific direction i.e. it compares the classification result of x_0 with a specific subset of rivals. The illustrated pathological case shows that points that lie far away from the boundary when multiple small regions lie in between, explanations can be problematic. This can be indicated by an unbalanced simulation design in combination with a high resulting faithfulness. Overall, the algorithm behaves on average rather stable for this toy case.

4.2 Experiment 2 - A low dimensional comparison with LIME

In this experiment two different versions of DBA will be compared with LIME algorithm for various low dimensionality binary classification problems. In sections 4.2.1, 4.2.2 the data and models used in the experiment are presented respectively. The evaluation procedure is discussed in section 4.2.3, while results will be illustrated in section 4.2.4.

4.2.1 Data

For the purposes of the experiment the following datasets were considered:

- **Titanic**, a well known Kaggle dataset [1] containing the features of 891 passengers of the Titanic disaster, labeled by survival status (0 not survived / 1 survived). Since DBA operates with numerical features, categorical features were omitted. This resulted in a toy dataset of **4** features: "Age", "Passenger Class", "Number of Siblings/Spouses" and "Fare", where the ordinal factor "Passenger Class" was treated as continuous.
- **Magic**, a UCI Machine Learning repository dataset [2] produced by Cherenkov gamma telescope which captures gamma radiation signals which leak the atmosphere. The telescope collects image data repeatedly pre-processed through Principal Component Analysis to produce an elongated ellipse. Eventually each measurement is represented by **10** features of this ellipse such as "Principal Axis", "Elongation" etc. The characteristic parameters of the ellipse are used for discriminating "gamma" signals from "background" radiation.
- **Breast cancer**, another dataset [3] from UCI ML repository, of a binary classification task which involves classifying malignant over benign state of a cell nuclei, based on image data (569 samples). Feature extraction yields a set of **30** numerical predictors representing characteristics of the nuclei.

All features of the aforementioned datasets where standardized, by subtracting the average and dividing by the standard deviation of the feature. In the next paragraph the trained classifiers are mentioned along with their parameters.

4.2.2 Classifiers

For the purposes of this experiment, the following classifiers will be trained and explained.

[1]https://www.kaggle.com/c/titanic
[2]https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope
[3]https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

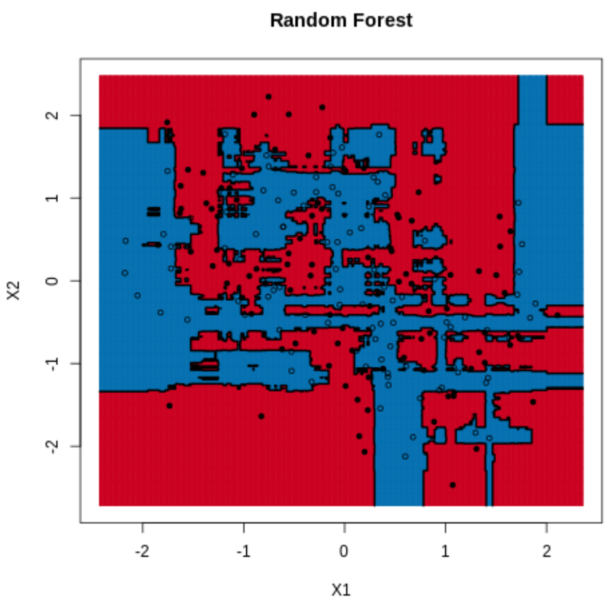


Figure 4.8: A toy example of a complex Random Forest decision boundary. In such cases multiple parallel regions (sandwitch regions) or small radial regions (holes) might appear.

- **Logistic Regression**, one of the simplest and most interpretable classifiers, to investigate whether the algorithms can recall with precision the (available) solution of a linear boundary.
- **Random Forest**, a classifier with a rather complex boundary structure. For cases where classes highly overlap *holes* and *sandwich* regions might be encountered (Figure 4.8). In this structure, a prediction of an instance might be a pooled result of multiple surrounding boundary regions, rather than a single region. The variables sampled by each tree were tuned with 5-CV in all cases. The number of trees was set to 100 for all datasets.
- **Neural Network**, a more complex structure which can fit boundaries of various types. All networks were employed with a single hidden layer. Number of units, learning rate and decay parameter were fine-tuned with 3 fold cross validation.

One parameter that should be introduced, is the *average degree of certainty* of the classifier, that is the average predicted probability for each class. A classifier which is highly uncertain about most of the points might be associated with multiple decision regions: for boundaries such as Figure 4.8, it is more likely for a point to be closer to a decision boundary region, eventually making the classifier less certain for most of the instances. It is of interest to study how the behavior of different explainers varies under different levels of certainty.

4.2.3 Experimental set-up

The experiment runs the following processes for all three datasets:

Explanation process After training, predicted probabilities of all instances from the positive class, are obtained for all models. For each classifier, we uniformly sample 100 of these positive outcomes and produce their explanations with LIME and two different versions of DBA, one in which the explainer learns the classifiers' labels and one that learns the predicted probabilities. The reason for employing these two versions, is to investigate whether learning probabilities on the decision boundary differs substantially than learning labels and compare with LIME which learns probabilities.

For LIME we use the full feature set (no feature selection) and skip data discretization to increase accuracy of the estimates. The kernel width σ was set to the default value that LIME authors suggest in their software implementation [4], that is $0.75\sqrt{p}$. Simulation is performed through sampling from the center of mass of the training set (see section 2.3.3, paragraph "Numerical features"). DBA runs bisection with 100 rivals, uniform convex hull sampling and a ridge explainer with small penalty ($\lambda = 0.001$). In probability version, the models' class posteriors are directly regressed while when learning the labels a logit link is applied. All explainability algorithms run with 1000 simulated samples per explanation (re-sampling on each explanation).

Evaluation process For evaluating the explanations produced by all methods, we employ the evaluation procedure described in section 3.5. We move each explained instance iteratively with a small step towards the explanation direction and on each iteration we let the classifier to predict its class probability. In this way, we monitor the probabilities on the *path* of the instance towards the decision boundary where they are expected to decrease. Eventually a visual assessment will be made after averaging the paths of all explained instances for each algorithm. Moreover an approximation of the local gradient in the direction of the explanation is reported (section 3.6), to measure the effect of moving in the neighborhood of x_0.

Regarding the step h of the evaluation process, it is argued that it should depend on dimensionality, since Euclidean distances become larger as p increases. We fix the number of iterations to $N = 100$ and set $h = 0.01$ for titanic and magic, while $h = 0.1$ for breast cancer, resulting in one unit step for the two former and 10 unit steps for the latter.

The primary aim of this procedure is to investigate if the explanations produced by the algorithms result on average in a significant (but also meaningful) change of the class posterior and compare the behaviour of DBA with LIME's by means of explainability *nature* and accuracy of the estimates. More accurate estimates will lead to a faster change of class (steeper path). Solutions that do not result in class change but merely model random local probability fluctuations should be treated with caution. In addition, the experiment aims to show how the behaviour of such algorithms varies with respect to different levels of the classifier's certainty (boundary instances, mediocre certainty, high certainty), as well as different boundary and data structures.

[4]https://github.com/marcotcr/lime/blob/master/lime/lime_tabular.py

To introduce a *benchmark*, we include in the process a move towards a randomly sampled direction to infer whether the explanations are on average (significantly) better than a random guess.

4.2.4 Results

Figure 4.9 shows the average probability path over all instances for each model and data. The dotted black lines in the plots indicate the class cut-off point (0.5). An overall impression is that both versions of DBA ("red": labels, "green": probabilities) reach the dotted line faster than LIME ("blue") and significantly differ by random guess ("black").

Moving towards the decision boundary of Logistic Regression, produces an approximately linear decrease of the class probabilities up to the boundary for all cases. The gradient of the path shows no differences for DBA and LIME which successfully recover the coefficients of the model. A random explanation does not produce on average a decrease in the odds, a fact which increases trust on the explanations. This result is in an agreement with the linearity of Logistic Regression's decision boundary which leads to a smooth decrease of the probability field, while moving towards the boundary.

Results for Random Forest vary by data case. The paths on the titanic RF probability field appear to be rather unstable in comparison with the rest of the cases. That is possibly due to the relative high complexity of the model which (over)fits the 4 dimensional dataset. In order to maximize its accuracy, the forest seems to have fitted multiple regions resulting in a complicated decision boundary. Thus, moving towards a direction in the feature space will result in an unstable change of the class posterior due to the effect of the multiple surrounding regions. The corresponding plot implies that explanations provided by all algorithms do not differ significantly by the random explanation: if the decision boundary lies in multiple directions, sampling one at random would on average decrease the odds (for instance consider Figure 4.8). However, DBA still manages to produce on average a faster change of class than LIME, although after crossing the boundary the probabilities stabilize. It is thus plausible that DBA explains closer but smaller regions than LIME which averages the effect of all regions. In magic dataset the forest seems to behave more stable in variations towards the explanations' suggestion. A random guess does not produce a change of class for this case, although on average it still manages to decrease the odds. DBA produces reasonably steeper paths than LIME implying that estimates are more accurate. Finally for breast cancer data, DBA and LIME share similar behaviour which is significantly better than random.

Neural Networks results favor DBA for all cases, since it suggests paths that on average, lead the points faster to the decision boundary. More interestingly, results suggest that explanations produced by LIME for breast cancer's data are on average worse than assigning random explanations. That is due to the fact that, predictions of this network appear quite insensitive to changes: the opinion of the network is not going to change much unless a meaningful variation is applied (see section 2.3.5

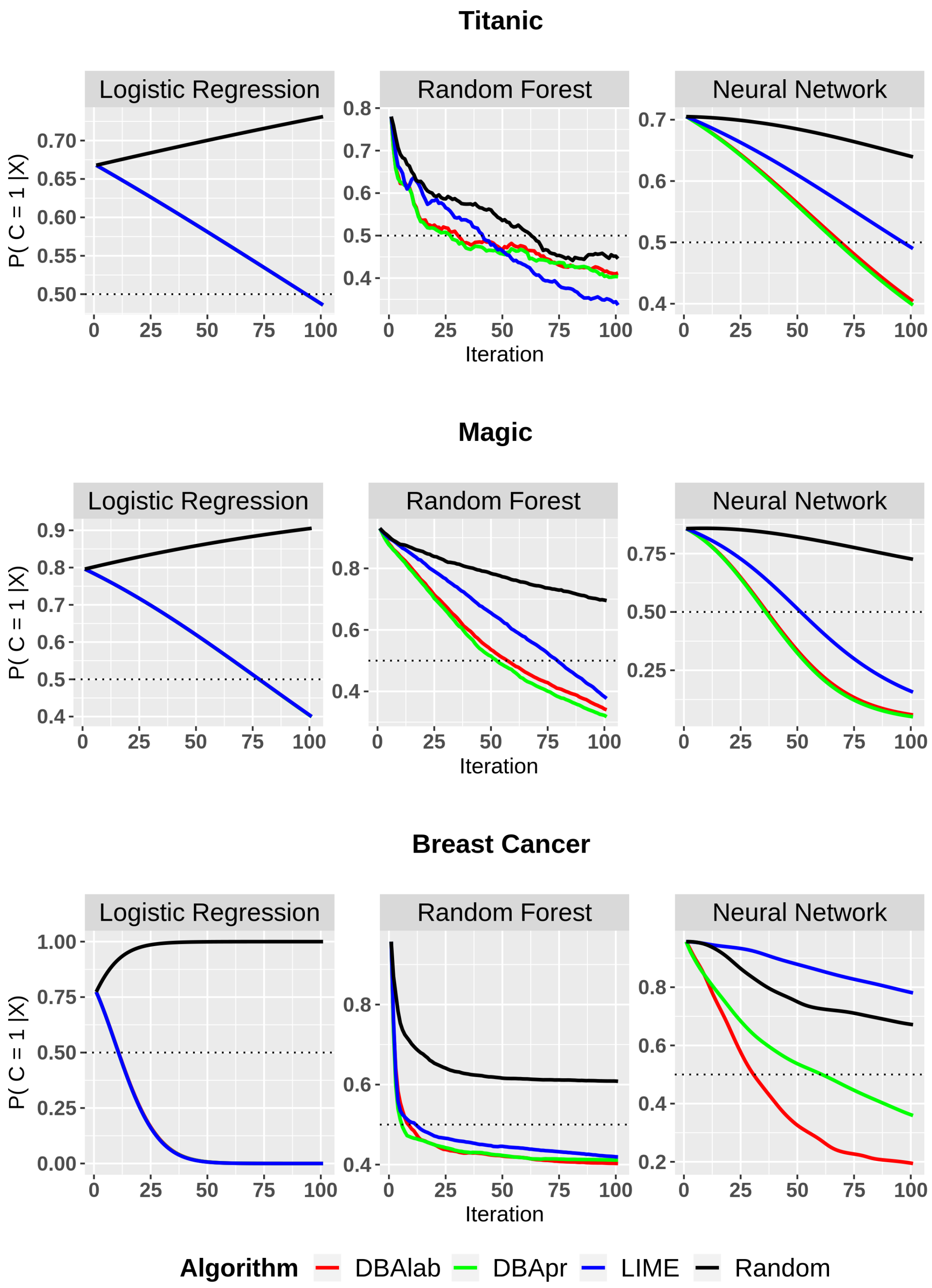


Figure 4.9: Average predicted probability path of 100 instances, produced by following the direction of the explanation vector for each instance. Overall both versions of DBA ("red" and "green") produce a faster change of class than LIME ("blue") and a random guess ("black"). For Logistic Regression all lines apart from random coincide. The dotted line represents the threshold where class changes (0.5).

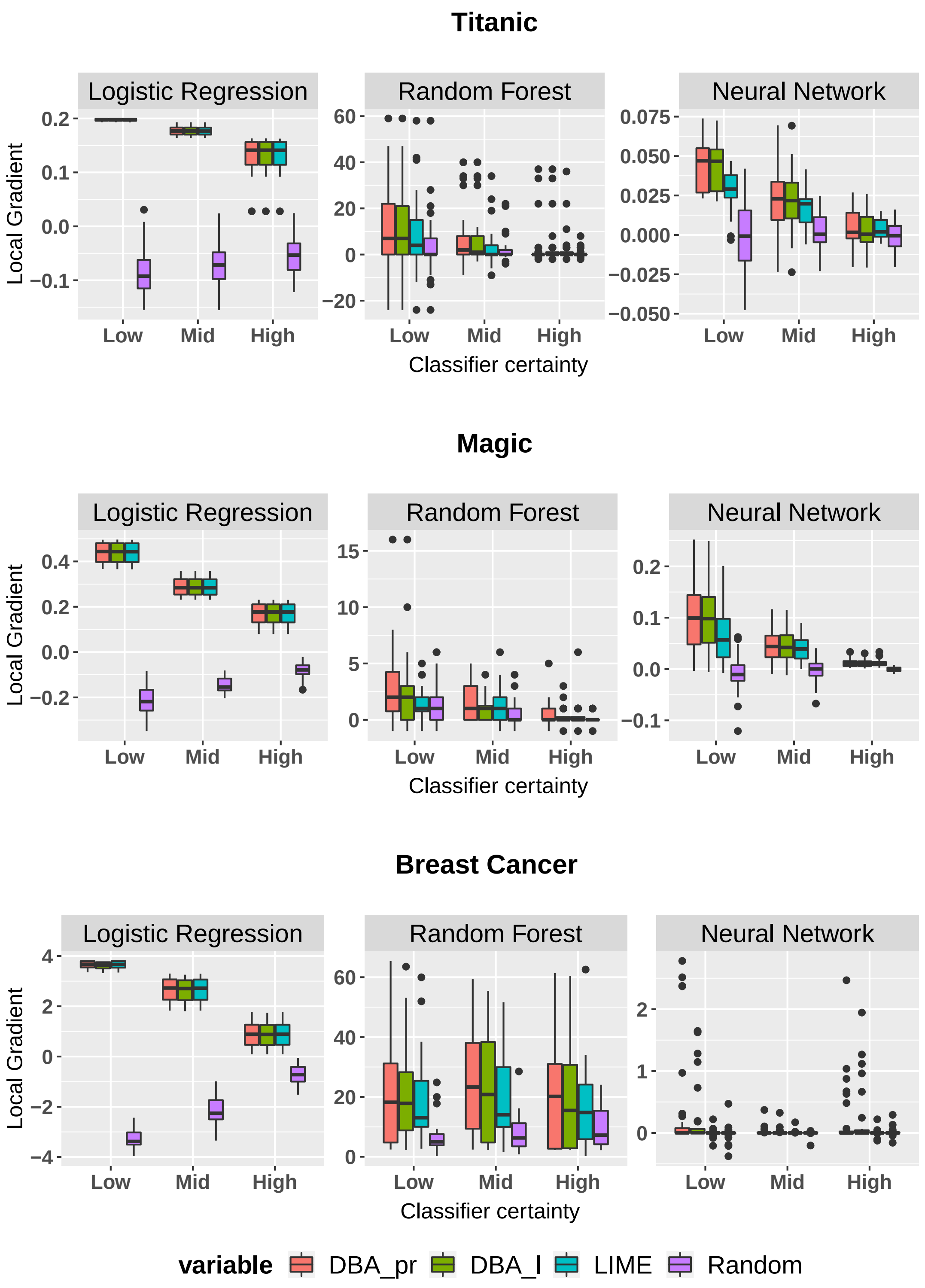


Figure 4.10: Estimates of the gateaux derivative for three different levels of classifier certainty (Low, Medium and High). On average, for low and medium classifier certainties, both versions of DBA suggest directions which lead to a more efficient class change than LIME's.

Point3). On the other hand, the labels version of DBA clearly outperforms all other methods, resulting in the steepest decrease. LIME weights instances in the neighborhood of the example exponentially and if the neighborhood's gradient is small, then the solution cannot be properly captured. However, if we sample on a relevant deision boundary region, the probabilities can be learned more efficiently and as a result the explanation of the discrimination rule would be more representative. Nevertheless the results on this network suggest that approximating the decision boundary by learning the labels is the most efficient approach among the others.

Finally, Figure 4.10, shows the distribution of the approximated local gradients over the classifier's certainty. This reflects how steep is the change in the predictions in the neighborhood of x_0. Although for some cases the medians of the gradients do not statistically differ, DBA suggests - on average - directions with a steeper gradient in the neighborhood. For points closer to the boundary (Low certainty), DBA provides explanations which correspond to a more extreme local gradient (more outliers and larger *whiskers* in the plot). Moreover, the aforementioned insensitivity of the network becomes now more transparent, since the gradient range in the neighborhood is much smaller compared to the corresponding forests'. It should be noted that DBA outperforms LIME for cases where the gradient is not that steep, since it approximates the decision boundary rather than minimizing the loss in an insensitive neighborhood.

4.2.5 Experiment overview

This experiment has revealed that for the low dimensional cases illustrated here, both DBA versions on average either outperform significantly LIME, or behave statistically equal depending on how steep the probability field can be. LIME can be adapted by varying parameters such as kernel width, however as mentioned in Chapter 2 (section 2.3.5, Point 2), it is not clear what should be the optimal width for the kernel since this may vary for different classifiers, data, but also single instances.

4.3 Experiment 3 - MNIST

Although low dimensional cases are useful for examining the basic functions of the algorithm, it is not a given that the behaviour shown in the previous experiments will appear for high dimensional cases and more complicated data structures. The concept of "closest" becomes more and more abstract when dimensionality increases and it is of great interest to examine how this algorithm reacts in higher dimensional spaces.

This experiment aims to provide insight on the following topics.

- Test the algorithm in a high dimensional case with grey-scaled images, where the coefficient estimates can be visualized on the image.
- Compare the default *vertex* creation method with the two different alternatives

discussed in section 3.3.1. Study the impact of introducing superpixels in the simulation process.

- Employ Partial Least Squares as an explainer to estimate in a high dimensional space through dimensionality reduction. Visualize the dimensionally reduced simulation and explainer to evaluate the explanation process.

- Extract visual intuitive interpetation to examine the nature of the explanations.

In order to research these topics, the following data (4.3.1), classifier (4.3.2) and experimental set-up (4.3.3) are considered.

4.3.1 Data

In this section the case of MNIST data [5] is considered, in an attempt to explain the predictions of a neural network, trained for a simple binary classification task: discriminate between ”1” and ”4” handwritten digits, based on their pixel intensity. This is a rather simple task of well separable sparse data. In fact, each digit lies on a separate lower dimensional manifold, since most of the pixels in an image are zero and the position of the zero elements differ from case to case.

Data consist of 14,701 grey-scaled 28×28 images, resulting in a 1-dimensional array of 784 entries. Thus, each image is represented by 784 features of which the majority is zero (sparse data). Features are normalized with the maximum pixel intensity (255) so that their values fall in the interval $[0, 1]$. A randomly drawn class balanced portion of 20% of the data is assigned as test set, while the rest is used to train the following network.

4.3.2 Classifier

An artificial neural network with one hidden layer is trained on the aforementioned MNIST subset. The learning rate and the optimal number of neurons in the hidden layer (5 neurons) was estimated through 3-fold cross validation. The decay parameter was set to 0. Hidden units employ the sigmoid function to transform the input and model non-linear relationships.

The classifier performs with great accuracy measures for this relatively easy task misclassifying only 4 test examples (99.8% test accuracy). The reason of this high performance is the well separability of the data.

4.3.3 Experimental Process

The experiment is divided into two main parts; first part aims to test and illustrate the technical details of the algorithm and compare three different vertex creation methods. The second part of the experiment has the purpose of interpreting the network with DBA and shed light on its prediction rule for misclassified instances.

[5]http://yann.lecun.com/exdb/mnist/

For the first part of the experiment a single instance is considered and the three steps of the algorithm are illustrated in a close up analysis of this instance. DBA runs three times, each one with a different vertex creation set-up. The three methods are summarized as follows:

Method 1 - Default Each feature will map into two vertices, $z \pm d(z, x_0)$ which lie on the surface of the ball $B(z, d(z, x_0))$. As dimensionality increases it is expected that sampling under this set-up will result in a simulation set close to the center z (decision boundary point).

Method 2 - Distant vertices Same as Method 1 each feature will map into two vertices. However in this case each pair will be set as $z \pm d(z, x_0)\sqrt{p}$, to scale for dimensionality, as discussed in section 3.3.1. This set-up aims to investigate the extent on which the explanation is affected by sampling more uniformly within the ball of relevance, compared to the default method.

Method 3 - Superpixel vertex creation In this version the detected decision boundary point is segmented through SLICO method (Bakkari 2015). The number of superpixels chosen depends on the user preference and different choices can result in different explanations. In this experiment $\sqrt{784} = 28$ superpixels are employed. The value of each batch is set such as the vertices lie on the ball of relevance (see section 3.3.1). Thus all features within a certain batch are varied simulatenously to create the vertices. As a result, the samples will be convex combinations of vertices resulted by varying batches of pixels rather than single pixels. Hence PLS will then estimate each batch as a whole due to the inherent correlations within each batch. The extent of which the explanation differs from the two other methods will be studied in the first part of the experiment.

For the second part of the experiment we *explain misclassifications*. The network misclassifies in total only 4 instances from the test set which are explained all via DBA under the default set up (Method 1).

In both parts the detection step of the algortihm runs with 1000 closest rivals and a tolerance equal to 10^{-9}. In simulation step 5000 samples are drawn from the convex hull of the vertices created by each method. Partial least squares runs with a (fixed) number of two components in all experiments to regularize the solution.

4.3.4 Results - Explaining a single prediction

Figure 4.11 (top left corner) shows a *1* that is accurately predicted by the network. Next to it, its 7 closest rivals are shown. Similarity in this case, is based on Euclidean distance, i.e. according to pixel intensity squared differences of two images. On the top left corner of each image the predicted probability $P(C = 1|X)$ is given. The network predicts with very high certainty the corrects labels and successfully discriminates these points. A qualitative interpretation can be drawn by visually

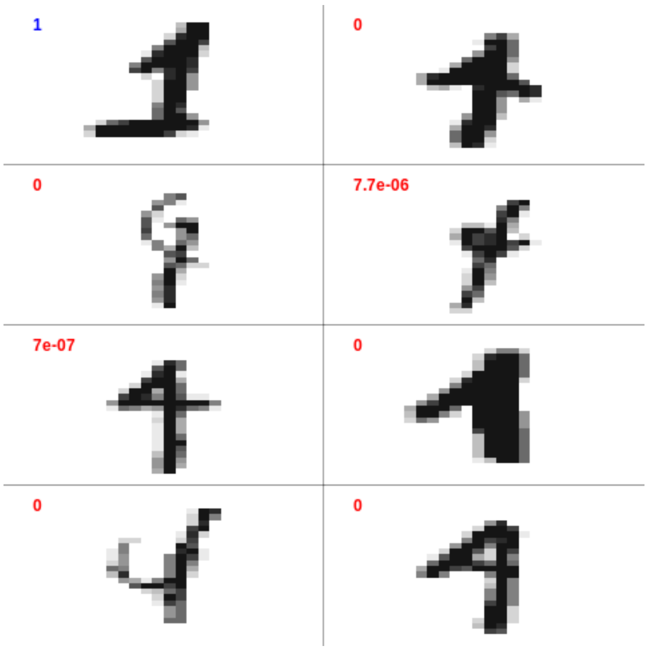


Figure 4.11: The example to be explained (top left image), next to its 7 closest rivals. The number on the top left of each image represents the predicted probability of being "1". Images do not look very similar and the network discriminates them with high certainty.

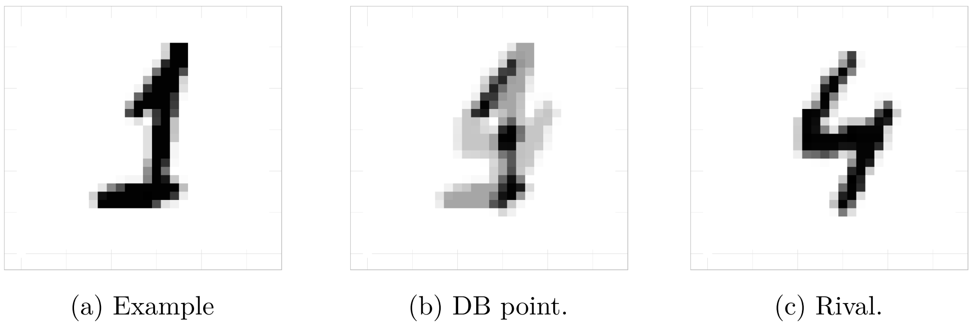


Figure 4.12: The example (left) opposed to the rival (right) that results in the closest decision boundary point (middle).

comparing these cases, however it is not clear which features contribute to each class exactly. In this section, DBA will be applied to extract the coefficient vector which will imply the explanation of this single instance.

Detection step

Detection step returns the decision boundary point depicted in Figure 4.12b. That point represents an (initial) estimate for the fastest way to force the instance change class. In Figure 4.12c the corresponding rival is shown. Comparing this rival with the rivals of Figure 4.11, it can be said that the similarity with the example is somewhat smaller, yet the bisection procedure results in a closer decision boundary point.

Simulation step

Figure 4.13a shows the three different types of vertices, one for each of the three methods. It should be noted that in order to lie on the surface of the ball, the resulting vertices may have pixels with values out of the the data range (< 0 or > 1). For instance the depicted vertex for method 1 has values equal to the decision boundary point's in all entries apart for one on the top right which is equal to 2.8. From this figure one can understand how the three methods differ: Method 1 and 2 increase/decrease single pixels of the image while Method 3 superpixels. Method 2 results in vertices further away from the decision boundary point (the value of the black pixel equals 78.4). The impact of each method can be seen in Figure 4.13b where a simulation example is shown. Method 2 results in samples that are less similar to the decision boundary point than Methods 1 and 2. It seems that Method 1 results is samples that sligthly vary, however since the classifier is in state of complete uncertainty a tiny variation is enough to produce a change in the odds. Histograms of Figure 4.13c further support this argument since the predicted probabilities of the samples of Method 1 actually (almost) uniformly distributed across the interval (0,1). Method 3 produces samples with more extreme probabilities and Method 2 even more. It can be argued that the default method simulates closer to the decision boundary than the rest of the methods. In the next paragraph, the impact of this result on the final explanation will be explored.

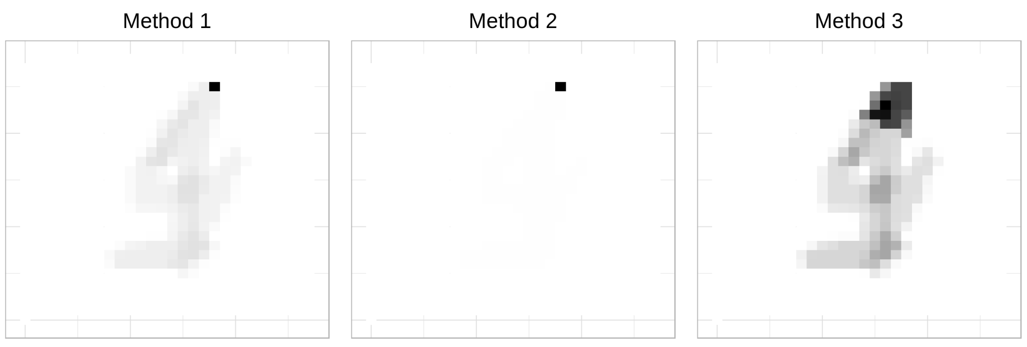
Explanation step

For all cases the two components of PLS explain more than 90% of information of the whole feature space. Partial Least Squares estimates are plotted on Figure 4.14. Blue regions of the image represent features that increase the odds towards *4* while red towards *1*. We can observe that the estimates between methods differ, however the directions of importance are more or less maintained.

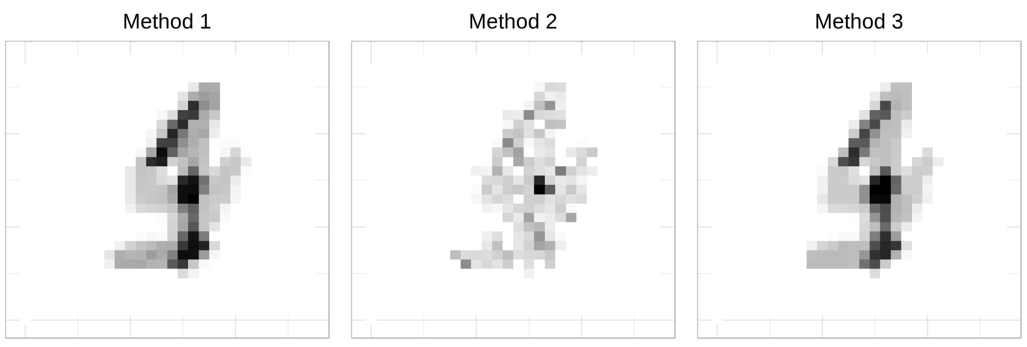
More specifically estimates of Method 1 and 2 differ only by intensity, since all estimates have the same sign. Hence, normalizing the explanations results in (approximately) the same coefficient vector. Introducing superpixels (Method 3) seems to drop the accuracy of the approximation since each batch averages the effect of all single pixels. Thus, if the effects within a batch cancel, the estimate of the batch will be close to zero.

Figure 4.15 depicts the projections of the simulation colored by predicted label. The black point represents the projection of the example, while the green line the dimensionally reduced explainer. It can be said that indeed Method 2 samples more uniformly within the ball of relevance, while the default method very close to the decision boundary. The behaviour of Method 3 lies somewhere in between. We also notice that the simulation is in all cases separated by the network's boundary with a class balance close to 0.5 and high faithfulness (about 0.9 for all cases).

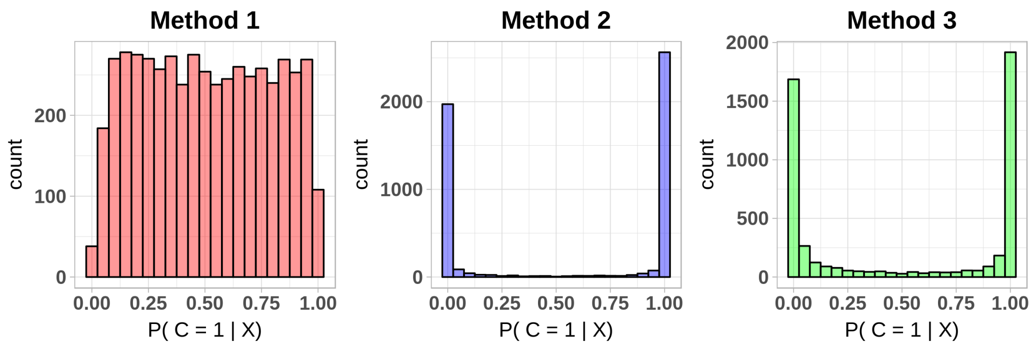
We may conclude that the three methods differ on simulation nature however the explanation agrees by means of interpretability: the network classifies this digit as *1* because of high intensities on its bottom tail and head and low intensities on



(a) Vertex example



(b) Simulation example.



(c) Histogram of predicted probabilities of simulation.

Figure 4.13: Comparison of simulation results for the three different vertex creation methods. Method 2 sets distant vertices to sample more uniformly in the ball of relevance. Methods 1 and 3 simulate instances very similar to the decision boundary point. The histograms show the distribution of the predicted probabilities of the simulation. Method 1 samples points closer to the boundary than the two other methods, however the sample is class balanced and separable.

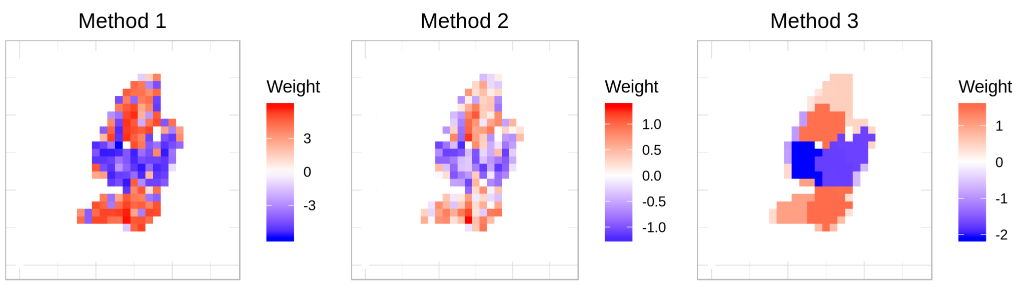


Figure 4.14: Explanations estimated by DBA for the three different vertex creation methods. Increased blue regions are associated with increased odds towards predicting *4* while red *1*.

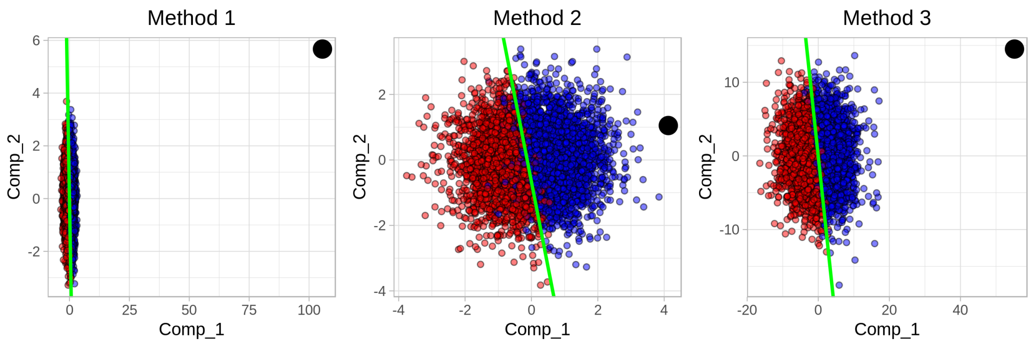


Figure 4.15: Projections of the explained example (black), simulation set (red and blue) on the first two components computed by PLS. Green line represents the projection of the explainer.

the left blue region which increases the odds towards *4*. If the top edge of this *1* was considerably closer to the bottom (e.g Figure 4.11, fifth rival) the classifier would have classified as *4*. In the next paragraph we will produce a more detailed picture of how class changes if we apply a *move* towards the explanation direction.

Evaluation

To evaluate whether the explanations reflect reality we may *move* the example towards the direction of the coefficient vector and inspect the class posterior. In a process similar to Experiment 2, for each vertex creation method we assume the probability path of the instance with a step of 0.1 (Figure 4.16)). The plot suggests that methods 1 and 2 result in approximately the same path with Method 2 performing slightly better. Both methods change the class after 3.5 unit steps. On the other hand, superpixel vertex creation results in a slower change of class (6 unit steps), due to the loss of information.

In a second attempt for evaluation Figure 4.17a shows the masked example (only non-zero entries varied) according to the explanation. This can be compared with Figure 4.17b where only zero entries are increased and Figure 4.17c where both zero and non-zero entries are altered. The probability shown (blue number on the top left of the image) implies that the posterior changes more with adding zero entries than masking non-zero elements at the same extent. This is further evidence to

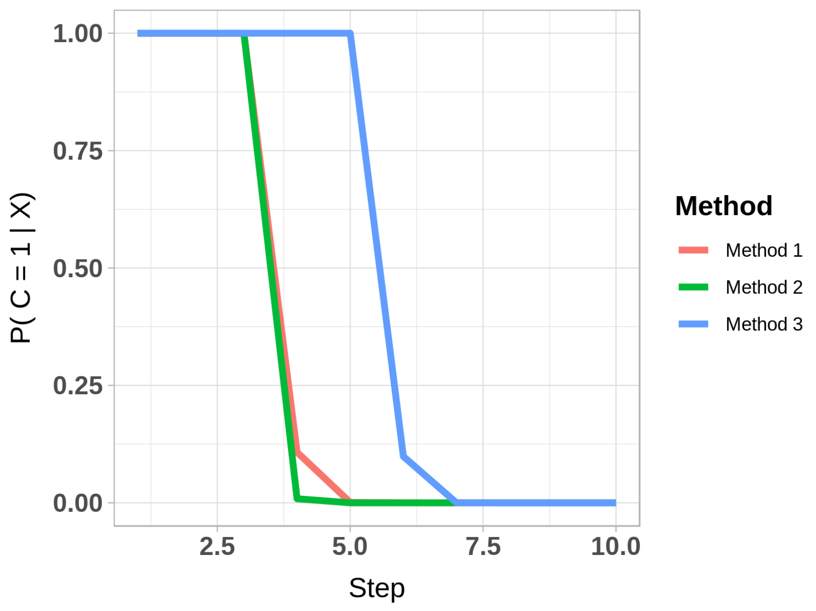
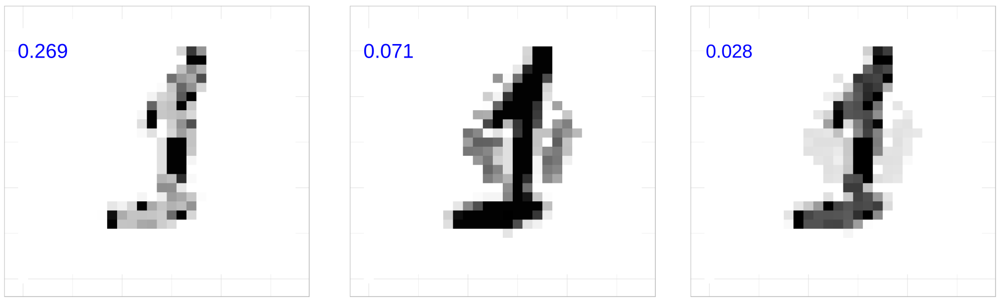


Figure 4.16: Probability paths for the three methods employed. x axis corresponds to the step taken towards the direction of the explanation. Method 2 (green) slightly outperforms Method 1 (red). Method 3 (blue) results in a slower change of class.



(a) Masking non-zero entries. (b) Adding zero entries. (c) Both.

Figure 4.17: Resulting images after increasing or decreasing pixels of the original image, according to the feature weights estimated by DBA. On the top left corner of each image $P(C = 1|X)$ is given. Left image shows masking of only the non-zero entries, in contrast to the middle image where only zero entries of the image are increased. On the right, values are increased and decreased according to the full explanation. In all cases the image changes class.

support that sampling by dropping only non-zero entries (LIME) is not sufficient to provide a complete local explanation when data are sparse. Considering both increasing and decreasing all entries results in a more complete picture.

4.3.5 Results - Explaining misclassifications

To further understand the behaviour of the network the four misclassified examples of the test set are explained. These cases are shown in Figure 4.18, along with their closest decision boundary point, the corresponding rival and their explanation. It is interesting to notice that all cases lie close to the decision boundary since the closest DB point looks very similar to them.

Case 1 is a *4* with an extreme angle classified as *1* by the network. The explanation implies that the reason is mainly its extreme left part which ends up in a red region (important for *1*). If the digit looked more alike its corresponding rival or had less intensity in this region it would have passed as *4*.

Case 2 is a *1* with a random scribble appearing to the right of the image. It turns out that this scribble increases the odds of the opposite class. More importantly the right skewness of this digit results in more pixels in the middle, a region important for predicting *4*. Moreover there is also lack of the bottom tail which would increase the probability of being *1*.

Case 3 represents a misclassified *4* due to its extreme slope. In the training set there are multiple *1* digits that share this extreme slope (e.g the rival of case 4) and the network seems to have associated this slope with class *1*. The example should look more like its corresponding rival in orde to be classified correctly, or have less intensity on some of the red regions.

Finally case 4 is a misclassified thin *1*, which it would pass as *1* if it only were *fatter*. It turns out that its non-zero entries favor *4*.

Overall it can be said that this network classifies *4* based on middle parts of the image while *1* based on top and bottom. The exact way depends on the decision boundary region and differs for each instance.

4.3.6 Experiment overview

This experiment has shown that through PLS the full explanation can be derived for a higher dimensional case and the dimensionally reduced solution can produce a visual diagnostic for the explanation fit. Comparing the three vertex creation methods has shown difference in simulation nature yet interpretation (directions of effects) are in agreement. Introducing superpixels in vertex creation considerably drops the explanation accuracy and strongly depends on the segmentation procedure. However we have shown that if batches are meaningful, we may recall a result similar to the full explanation. Last it has been shown that it is possible to employ DBA to understand the local behaviour of the network by explaining misclassified examples. More importantly we conclude that when data are sparse, sampling from the example by dropping non-zero entries will not estimate effects of important

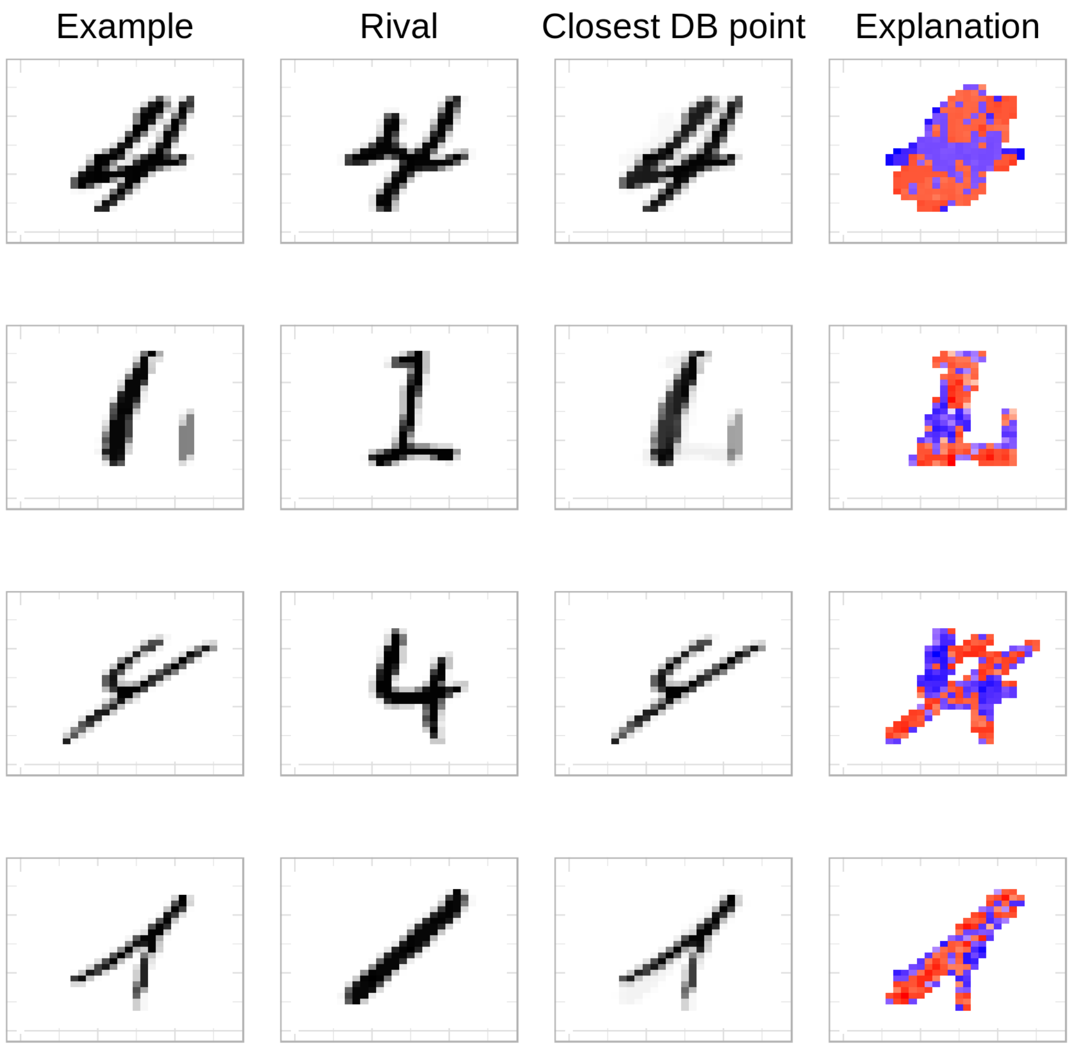


Figure 4.18: The explained misclassified examples (column 1), along with the rivals (column 2) that if bisected with, it will result in the closest decision boundary point (column 3). In the fourth column the DBA explanation is shown. Blue regions favor the *4* class while red regions the *1* class. Studying these regions can provide interpretation on how were these examples misclassified.

features that correspond to zero entries.

4.4 Experiment 4 - Explaining Naive Bayes

In this experiment we will approximate the decision boundary of a Naive Bayes classifier, where class posteriors for each feature are available. For this purpose, the SMS Spam Collection dataset [6] from Kaggle will be employed, a sparse high dimensional discrete dataset of short-length text messages.

Experiment's purposes are the following:

- Test the feasibility of DBA on a discrete data case while DBA performs a continuous approximation.
- Validate the yielded explanations by comparing with the global interpretation available by Naive Bayes classifier.
- Investigate possible biases of Naive Bayes by explaining misclassified cases.

For these purposes, a close up analysis of a single explanation will be illustrated followed by a pooled explanation of all misclassified cases.

4.4.1 Dataset

The SMS spam dataset contains 5574 short messages of two categories namely "ham" (86.6 % of total messages) and "spam" (13.4 % of total messages). All messages are processed in a corpus and the document feature matrix is created - a sparse matrix of numerical counts each one corresponding to a specific feature (word, digit or symbol). There are 9171 different features in total. However, for this experiment the classifier will be trained only with terms which appear at least three times, resulting in 1923 predictors. To simplify the case, we will assume that we know a-priori that the feature space consists only of these features.

4.4.2 Naive Bayes

Data are randomly shuffled and partitioned (80% train - 20% test) and a Naive Bayes classifier with smoothness parameter equal to 1 fits on the training set. The classifier performs with a 97.9% sensitivity and a 95.9% specificity on the test set. To investigate whether this accuracy is managed through a meaningful feature set, the Naive Bayes posteriors for individual features can be studied. In the following, explanations will be derived through the DBA algorithm and compared with the Naive Bayes posteriors to formally evaluate the explainability performance. The set-up of DBA is provided in the next section.

[6]https://www.kaggle.com/uciml/sms-spam-collection-dataset

4.4.3 DBA set-up

The results of the experiment will be presented in two phases, one in which a single instance is explained in a close up analysis and one in which a pooled explanation is derived for all misclassifications. In both cases DBA runs with the following settings.

Detection step runs with a fixed number of 400 closest rivals to approximate the closest decision boundary region of each instance, with a bisection tolerance $\epsilon = 10^{-9}$. Due to the small number of words of each message, the number of vertices will be in general small, therefore we employ the default vertex creation method (see section 3.3.1). Hence Simulation step yields $2p$ vertices, where p is the number of non-zero entries of the decision boundary point. Sampling from the convex hull of the vertices with uniform correction creates 1000 additional samples within the ball of relevance of each example. After obtaining the predictions of Naive Bayes on the simulation, a $l2$-Logistic Regression explainer approximates the decision boundary region associated with the instance.

For deriving a global explanation for the misclassified cases, DBA runs over all M misclassifications under the same set-up as above, to extract M explanations $\beta^1, \beta^2, ..., \beta^M$. To make all explanations have the same length, we set all lengths equal to the number of the total estimated features l and we pad with zeroes the non-estimated entries of each explanation. The pooled explanation is computed through averaging over all explained examples,

$$\beta^{pooled} = \frac{1}{M} \sum_{i=1}^{M} \frac{\beta^i}{||\beta^i||_\infty} \tag{4.1}$$

where $||\beta^i||_\infty = \max\{|\beta^i_1|, \ldots, |\beta^i_l|\}$. Each explanation is normalized with the absolute value of the corresponding maximum weight such that all coefficients lie between -1 and 1. Explanations are eventually pooled by averaging the normalized weights of each word present in the sample. Thus, if a term appears in multiple examples as the most important feature, its absolute value will be close to 1, while if the term does not appear as important its weight will vanish.

4.4.4 Results

Explaining a single prediction

Let us focus on the prediction of the following instance:

> *"You have 1 new voicemail. Please call 08719181513."* (**spam**)

The above SMS is predicted accurately as *spam* by the classifier with a probability of 95.5%. The tokenized version of this example (Table 4.1) omits the word "voicemail" and the phone number since they appear in the whole training set less than three times. Applying the predictive bisection rule to the 400 closest rivals of the target spam SMS, yields the most relevant training rival - i.e. the rival that if we average with the example it will produce the fastest change of class. The algorithm returns the following rival:

Table 4.1: Tokenization of the example explained by DBA. Terms that appear less than three times in the whole collection are omitted (e.g phone numbers).

	you	have	1	new	.	please	call
1	1.00	1.00	1.00	1.00	2.00	1.00	1.00

Table 4.2: Generated Decision boundary point by Detection step applied on the given example. The set of features is the union of non-zero elements of the example and its most relevant training rival.

	you	have	1	new	.	please	call	it	do	ok	right	later
1	1.00	0.82	0.82	0.82	2.00	0.82	0.82	0.18	0.18	0.18	0.18	0.18

"Ok. You do it right later." (**ham**)

A first impression is that this rival is not that similar with the example. The word "you" and the character "." are the only common features in the two messages. However, in DBA framework locality is not of great interest: detection step returns the rival that if we "move" x_0 towards, the class will change in the most rapid way. For example, adding the word "Ok" might lead to a much larger change in the odds of the classifier than words existing in more similar opposite messages. This will lead to an explanation which approximates the most relevant decision boundary region of the given example.

The corresponding decision boundary point (Table 4.2), being produced by averaging has no natural meaning (words cannot appear 0.82 times). However, that is a theoretical point that *confuses* the classifier and lies on the decision boundary ($P(spam|z) = 0.5001$). In this state of uncertainty, the classifier will be very sensitive if an important feature varies. Thus, varying each feature individually will produce a change in the odds of the classifier according to its importance. That is not necessarily true when we sample from x_0 (explained example) (LIME) and the predicted probabilities are insensitive in a neighborhood of x_0 (section 2.3.5, Point 3).

The vertex creation process creates 24 vertices on the surface of the ball $\mathscr{B}$ with a class balance of 0.48. Convex Hull sampling yields a simulation set with 40% spam messages and 60% ham (class balance 0.6). Eventually the simulation is fed to the Naive Bayes classifier to acquire the class labels. The faithfulness of the explainer for this case is 100%, implying that the boundary region is locally linear and thus well approximated (given the reasonable class balance).

The explanation is shown in Table 4.3 in comparison with the Naive Bayes posterior "ham" class probabilities. It can be observed that the explanation is in general agreement with the Naive Bayes class posteriors, except for the terms "new" and "1". However, these words seem to play an approximately equal role in the classification result and the ordering disagreement should be due to randomness.

Interpretation of the estimates, reveals that the most important feature for the message to be classified as "spam" is **the absence** of the word "later", which if added on the original message, forces the classifier to classify the message as "ham"

	Estimate	NB Posterior	Order Agreement
later	**-50.67**	**0.97**	✓
ok	-49.23	0.95	✓
right	-36.46	0.91	✓
do	-27.34	0.85	✓
it	-24.03	0.84	✓
.	-13.56	0.68	✓
you	-6.68	0.63	✓
have	2.11	0.46	✓
please	13.65	0.30	✓
new	19.31	0.21	✗
1	19.90	0.24	✗
call	24.46	0.15	✓

Table 4.3: Comparison of explainer's estimates with NB posterior probabilities ("ham" class). The ordering of the estimates more or less respects the ordering of feature importance suggested by Naive Bayes.

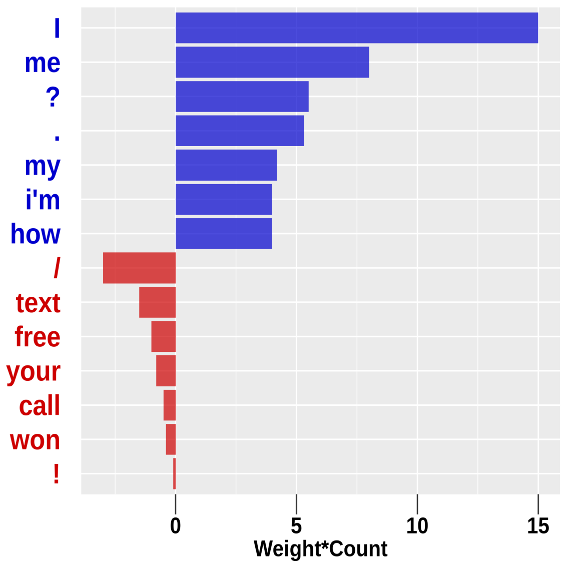
with a probability 57.4 %.

From the non-zero elements of the example, there is no word that can instantly change the prediction for the label if it is removed. Among these features, word "call" seems to increase the "spam" odds the most, followed by the terms "1", "new" and "please".

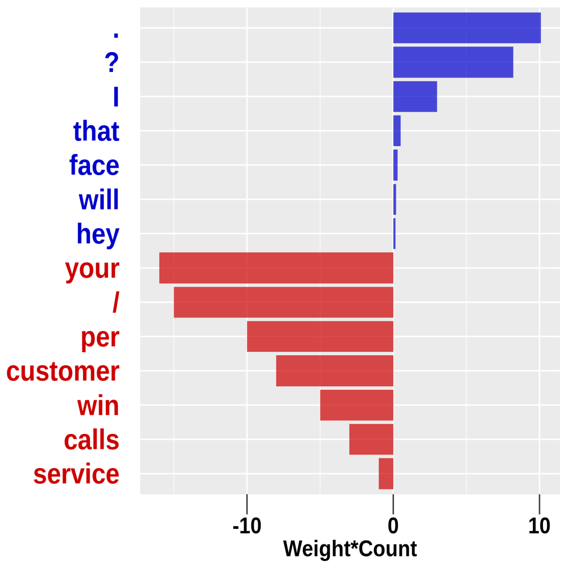
It is concluded that this message was classified as spam due to the presence of the terms "call", "1", "new" and "please" with that order, but more importantly due to absence of terms such as "later", "ok", "right", "do" and "it". On the other hand, features such as "." and "you" play a secondary role in classification, while the word "have" doesn't seem to influence the opinion of the classifier.

Text	P("ham")
"You have 1 new voicemail. Please call 08719181513."	0.044
"You have 1 new voicemail. Please call 08719181513 later."	0.574
"You have new voicemail. Please call 08719181513"	0.120
"You have new voicemail. Please call 08719181513 later."	0.812
"You have 1 voicemail. Please call 0871918513."	0.145
"1 new voicemail. Please call 0871918513."	0.026
"Ok... Right. You have voicemail. Please do call 0871918513 later. Do it."	0.9999998

Table 4.4: Various meaningful modified versions of the original spam example next to their predictions, after removing or adding terms according to the explanation.



(a) Spam classified as “ham”.



(b) Ham classified as “spam”.

Figure 4.19: Barplots depicting the most frequent words in misclassified messages weighted by feature importance calculated by DBA. Misclassified spam messages include words such as “I”, “my” while ham misclassification terms such as “your”, “customer”, “calls”.

Finally, Table 4.4 shows a few modified versions of the example, generated by dropping or adding terms suggested by the explainer. The fastest (meaningful) way for the example to change class is simply by adding the word “later” in the text (second text). That is a modified version of the example which belongs in its locality. The last text of the table represents a modification which is more “human-like” and NB classifies it as “ham” with high probability.

Explaining misclassifications

Overall the Naive Bayes misclassifies 114 messages of which 83 were originally “ham” and 31 “spam”. In this paragraph the pooled results of DBA on these cases are presented. It will be examined if the explanation process can reveal biases which lead Naive Bayes to misclassify examples.

Results are summarized in the barplots of Figure 4.19. Barplot 4.19a refers to the “spam” examples misclassified as “ham” while barplot 4.19b to the “ham” examples misclassified as “spam”. All terms depicted, refer to the non-zero elements of the messages, which are sized by absolute frequency, weighted by the average importance of the feature. Thus, long bars tend to represent frequent words that also play an important role in classification, while short bars represent words that are either not frequent or not important (or both). Blue terms lead the classifier to favor the “ham” class while red terms are important for predicting “spam”. For simplification we show only the 7 most important terms for each class.

The figures imply that on average, “spam” misclassifications carry a lot of important terms for the “ham” category such as “I”, “me”, “?”, “my”, “i’m”, terms that in general imply a more “personal” style of messaging than the average spam message.

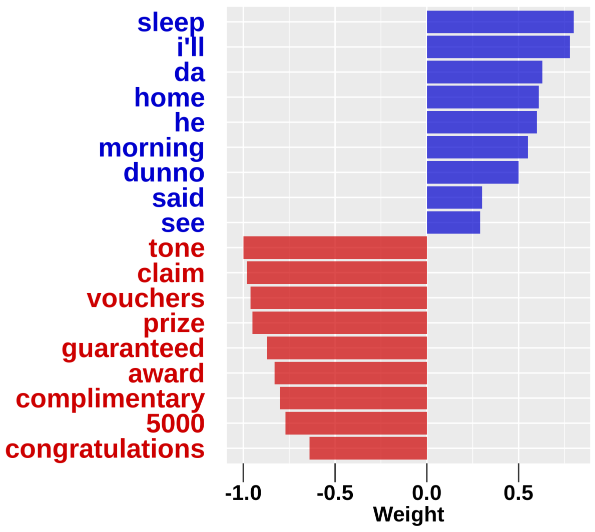


Figure 4.20: Words that do not appear in the misclassified example, but their absence strongly influences the decision of the classifier (Blue - "ham", Red - "spam").

On the other hand "ham" misclassifications include highly important terms for the "spam" class ("your", "/", "per", "customer", "win", "calls", "service") which in general would be more likely to appear in an advertisement-like message.

However, in DBA framework explanations are two-sided: these messages were also misclassified because of the absence of important terms that would increase the probability of correct classification if they were present. Figure 4.20 shows the (pooled) collection of the most important absences evaluated by the DBA. If spam messages had more words such as "tone", "claim", "vouchers", "prize" and ham messages more words like "sleep", "i'll", "home" the misclassification rate would be much smaller.

Overall, it is concluded that this classifier has learned a meaningful yet biased pattern to discriminate "ham" from "spam" messages. If a "spam" message includes more personal expressions (questions, first person pronouns or general words that a human would use more often) and less advertising words ("prize", "vouchers") it will pass as "ham". If a "ham" message is short (less ".", "?") and impersonal, with more symbols and words relevant to offers and mobile services it will be classified as spam. Thus "tricky" messages such as:

> "Are you free now ? Can I call now? !" (true: **ham**, predicted: **spam**)
>
> "Oh my god ! I've found your number again! I'm so glad, text me back xafter this msgs cst std ntwk chg ? 1.50" (true: **spam**, predicted: **ham**)

can *confuse* the classifier which has a biased general rule.

4.4.5 Experiment overview

This experiment has shown that Decision Boundary Approximation is capable of producing explanations for sparse discrete data, even if the algorithm's setting requires a continuous framework. Assuming a continuous feature space does not seem to affect the explanation trustworthiness and interpretability, since results are faithful to Naive Bayes' global picture.

Chapter 5

Discussion

In this final chapter of the thesis, we discuss some fundamental aspects of the algorithm, taking in account the theoretical concepts of Chapter 3 and the results of the experiments of Chapter 4. In section 5.1 we begin with some general remarks on DBA. In section 5.2 suggestions for future modifications are proposed. Finally, in section 5.3 the final conclusion of this research is summarized.

5.1 General Remarks

5.1.1 Explainability nature

Experiment 1 clarified that DBA attempts to approximate the closest decision boundary region of an instance of interest x_0. However in some cases, this is not the complete classification picture and provides partial transparency to the complex model's predictions. If multiple regions lie around x_0 (Figure 5.1), explaining a single region is not sufficient because more regions participate in the prediction result. We may accept though that what we explain is not the overall classification behaviour in the neighborhood, but the discrimination rule that the classifier employs to separate x_0 from a specific set of opposite points. An important fact is that in DBA, we specify which region we linearize through the most relevant rival. In section 5.2 we propose a multiple region Detection step.

5.1.2 Regarding metrics

This thesis focused on Euclidean metric spaces to approximate the solution. This is sufficient when p is not that large and features are numerical and standardized. When features are not bounded in the same range, various issues might arise. For instance, simulating in a convex hull where all vertices are distributed on the ball of relevance can be problematic due to the fact that some features might have a relatively larger range: they should vary more in order to produce a significant change in the odds, compared to features with smaller range. As a result, the concept of the ball becomes more abstract when features have not been standardized prior to training. In this case we should take this fact into account, by scaling the distance

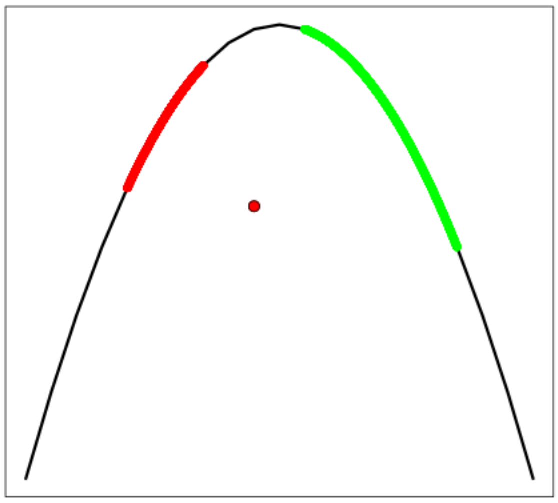


Figure 5.1: Illustration of an inherent drawback of DBA. Both red and green regions lie close to x_0 (red point), yet DBA would explain only the red region.

of the vertices in the vertex creation step with a factor associated with the range of each variable, to simulate with a larger variance in the corresponding dimensions.

When p becomes larger, many other issues might also arise due to the curse of dimensionality. The concept of similarity under the Euclidean framework fades as p increases. This results in equalization of the distances, leading the concept of *closest* to become more abstract. In fact, as p increases, all cases will tend to be equally similar to x_0. Thus the K-closest rivals concept becomes less meaningful. However an important question is: Do also distances from the decision boundary tend to equalize? From experience, we have found that in many cases although x_0 might have all other training points lying on almost equal distances, there exists a certain point that if bisected will yield a decision boundary point very close to x_0. In other words, it is possible that distances between x_0 and its rivals equalize, but this is not necessarily true for the corresponding decision boundary points. In the future formal experiments should take place to validate this assumption. If many decision boundary points lie on equal distances, it means that all regions should be explained. This can be problematic when the user demands a single interpretable explanation. A solution could be to decrease k in L_k metric thus turn into Manhattan metric ($k = 1$) to make distances distribute more uniformly (Aggarwal, Hinneburg, and Keim 2001). However the problem will persist for extremely large p.

Finally, for categorical features the whole concept collapses and a more appropriate metric such as *Jaccard* (Jaccard 1912) should be considered, while for mixed data types, a metric such as *Gower* (Gower 1971) should be employed. Nevertheless, it is not clear from this research how instances will average in the detection step to produce the closest decision boundary point. This necessitates more research in order to adapt the algorithm for such cases (see section 5.2). However, we have shown that DBA can sometimes perform equally well for discrete data cases by assuming a continuous Euclidean metric space (Experiment 4).

5.1.3 Discussion of simulation

Regarding the simulation process, we have proposed Convex Hull sampling to simulate points within $\mathscr{B}$. However, we have shown that under this procedure, most of simulated data points will lie close to the decision boundary (Experiment 2). This will naturally lead to learn a small decision boundary region which is actually a subset of $\mathscr{B}$. We argue that this is not undesirable in a framework where we accept that the explanation refers only to x_0 and the counterfactual rival instance that corresponds to z. It will moreover guarantee that no pathological cases such as the ones described in sections 3.5 (Figure 3.8) and 4.1.5 (Figure 4.7) will be encountered, since sampling close to the boundary will (almost always) result in a reasonably separable simulation set. In section 5.2 we will propose a modification of the algorithm that further exploits this fact to pool explanations that refer to different regions. In addition, we argue that in a high dimensional space within a certain ball of relevance with radius smaller than the distance of x_0 from its closest rival most likely there will be no training points. Therefore, it is reasonable to expect that within such a ball, the decision boundary will be locally linear, since it only discriminates a small set of points.

5.1.4 Discussion of vertex creation

We have proposed a vertex creation approach to initiate the convex hull sampling. An additional reason that motivates this choice is that the vertices can in principle incorporate any meaningful representation. For instance, the vertices can be taken to be all possible *masks* of an image in RGB (Figure 5.2). A mask can be defined as the result of applying a binary filter on the image (e.g clusters of colors). The clusters can be identified with a clustering algorithm. As a result, all possible convex combinations for these masks will be sampled and the effects of each masking will be estimated.

Another motivation of the vertex creation procedure is that it is computationally efficient compared to uniform sampling in the ball of relevance.

5.1.5 Applications

In practice, DBA can be valuable for decision making when research questions involve the component: “Which is the fastest way to change class”? Under the DBA framework, instead of explaining the neighborhood, we approximate the fastest direction to the decision boundary, i.e. the minimal variation needed in order to make the classifier activate the opposite prediction. Thus, Machine Learning models can be employed to produce predictions and then extract with DBA the directions that

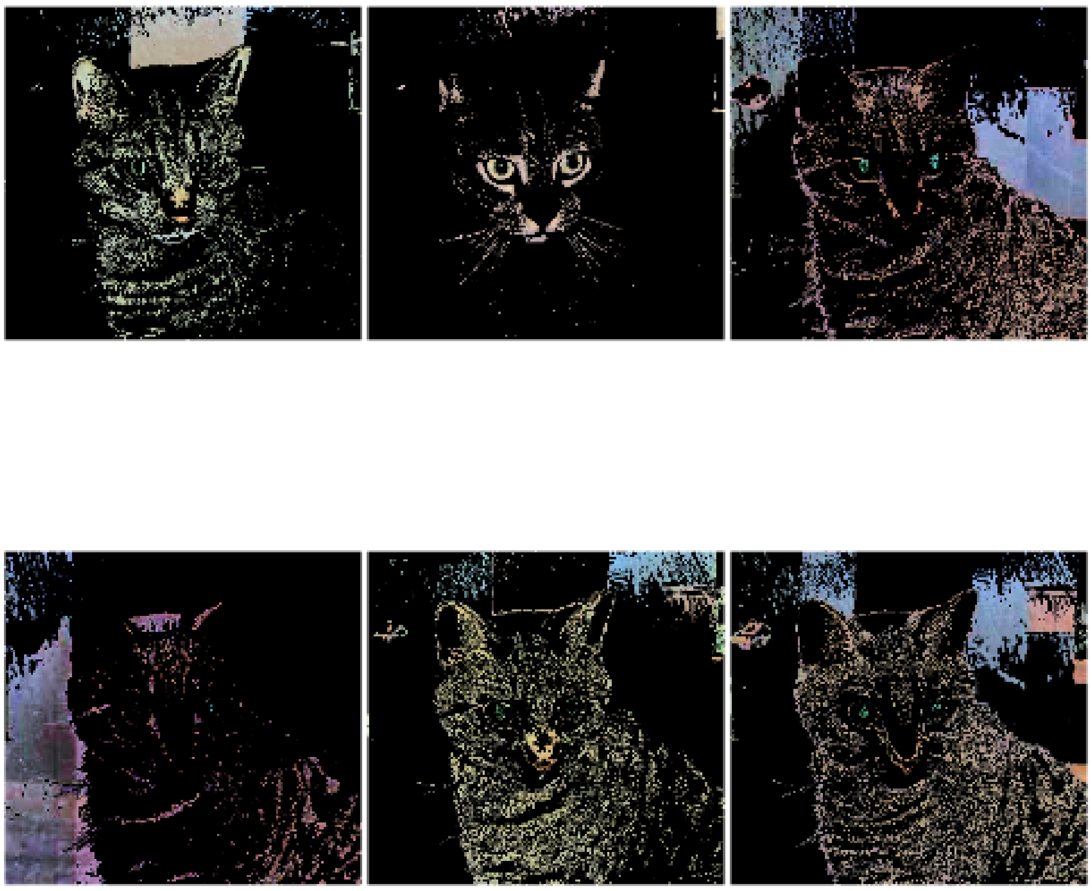


Figure 5.2: Masks of a cat image, computed by the Affinity propagation algorithm (J Frey and Dueck 2007). If the original image was a decision boundary point, these masks could be considered as vertices to sample from their convex hull.

minimize the effort to turn a "failure" into "success". There should be more experiments in a research/business environment to investigate whether these suggestions of directions lead indeed to "successes" after the corresponding variations are applied. We should always consider though that the explanation refers to the model and not reality, and such an approach would only work when it is decided that the model can be trusted completely.

5.2 Modifications and future work

Each one of the three main steps of the algorithm can potentially be extended. Each step can be modified independently from the others, giving degrees of freedom in future research. The Detection step employs predictive bisection that was designed for the purposes of this research. Future studies should consider more advanced optimization approaches to yield the closest decision boundary point. The algorithm can also be modified by introducing different simulation alternatives that respect Properties 1-4 of section 3.3. As for the explanation step, the impact of different explainers should also be explored. Finally, attention should be paid to the feature selection process in order to provide stable sparse explanations. The framework of the algorithm allows any feature selection option to fit in the Explanation step. LASSO or backwards elimination are such approaches. Another (post-hoc) option is to keep the K-features (or batches of features) that correlate the most, with the first two PLS components, or the K-most important features estimated by Ridge Regression. A tree-based feature selection would also be possible. Last, feature selection can be performed directly on the decision boundary by keeping only the vertex pairs that correspond to a posterior change above a certain threshold and simulate on a lower dimensional manifold.

A multiple region detection algorithm would be possible, if we sample multiple rivals of x_0 and for each one run DBA to obtain an explanation. Then, if we cluster the explanation vectors (e.g with K-means), we will receive K different regions that are relevant to x_0. That could provide another topic for future research.

Extensions of Decision Boundary Approximation for different data structures (e.g. categorical features) can be considered under a careful choice of metric. However, this would require redefining some concepts of the algorithm (for example bisection in the detection step).

Last, DBA could be extended to explain multiclass problems (or regression) by turning the task into multiple binary classification components. Thus, in the future more work should be done towards this direction, in order to make the algorithm functioning for all ML tasks.

5.3 Final Conclusion

In this research we demonstrated a novel approach in Explainability, Decision Boundary Approximation for explaining the predictions of any binary classifier trained with numerical features. Furthermore, we introduced diagnostics to help decide whether the explanations are trustworthy. The process unifies the concepts of example based (counterfactual) explanations with surrogate approximation to learn features that express local classification behaviour. Experiments have shown that in the default setting, such an approximation is realistic and more accurate than LIME when it comes to log-odds change. Thus, we may conclude that DBA can point out a faster way to change the opinion of the classifier towards the opposite class, thereby producing a more meaningful explanation. We propose this approach as a basis for further experimenting and modification, to eventually build a solid unified post-hoc explainability framework for classification.

Bibliography

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. Springer, 2001, pp. 420–434.

[2] Daniel W. Apley. "Visualizing the effects of predictor variables in black box supervised learning models". In: (2016). DOI: `arXivpreprintarXiv:1612.08468`.

[3] S. Bach et al. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PloS ONE* 10 (2015).

[4] Abdelkhalek Bakkari. "Segmentation of Cerebrospinal Fluid from 3D CT Brain Scans Using Modified Fuzzy C-Means Based on Super-Voxels". In: *FedCSIS* (Dec. 2015).

[5] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. "Interpretability via Model Extraction". In: *CoRR* abs/1706.09773 (2017). arXiv: `1706.09773`. URL: `http://arxiv.org/abs/1706.09773`.

[6] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *KDD* 2006 (Aug. 2006), pp. 535–541.

[7] Sanjoy Dasgupta. "Experiments with Random Projection". In: *CoRR* (2013).

[8] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

[9] Mengnan Du, Ninghao Liu, and Xia Hu. "Techniques for Interpretable Machine Learning". In: *CoRR* abs/1808.00033 (2018). arXiv: `1808.00033`. URL: `http://arxiv.org/abs/1808.00033`.

[10] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *Ann. Statist.* 29.5 (Oct. 2001), pp. 1189–1232. DOI: `10.1214/aos/1013203451`. URL: `https://doi.org/10.1214/aos/1013203451`.

[11] Alex Goldstein et al. "Peeking inside the black box Visualizing statistical learning with plots of individual conditional expectation." In: *Journal of Computational and Graphical Statistics 24.1* (2015).

[12] J. C. Gower. "A General Coefficient of Similarity and Some of Its Properties". In: *Biometrics* 27.4 (1971), pp. 857–871.

[13] Hastie et al. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer-Verlag, 2009.

[14] Wold Herman. “Partial Least Squares”. In: *Samuel; Johnson, Norman L. Encyclopedia of statistical sciences* (1985), pp. 581–591.

[15] Brendan J Frey and Delbert Dueck. “Clustering by Passing Messages Between Data Points”. In: *Science (New York, N.Y.)* 315 (Mar. 2007), pp. 972–6.

[16] Paul Jaccard. “”The Distribution of the flora in the alpine zone””. In: *New Phytologist* (1912).

[17] Zilke J.R., Loza Mencía E., and Janssen F. “Interpretability via Model Extraction”. In: *In: Calders T., Ceci M., Malerba D. (eds) Discovery Science. Lecture Notes in Computer Science, vol 9956. Springer, Cham* 9956 (2016).

[18] C.-C. Jay Kuo et al. “Interpretable Convolutional Neural Networks via Feedforward Design”. In: *CoRR* abs/1810.02786 (2018). URL: `http://arxiv.org/abs/1810.02786`.

[19] Thibault Laugel et al. “Inverse Classification for Comparison-based Interpretability in Machine Learning”. In: (2017). DOI: `arXivpreprintarXiv:1712.08443`.

[20] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *CoRR* (2016). arXiv: `1602.04938`. URL: `http://arxiv.org/abs/1602.04938`.

[21] Saharon Rosset, Ji Zhu, and Trevor Hastie. “Margin Maximizing Loss Functions”. In: NIPS’03 (2003), pp. 1237–1244. URL: `http://dl.acm.org/citation.cfm?id=2981345.2981498`.

[22] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. In: *CoRR* (2017). arXiv: `1704.02685`. URL: `http://arxiv.org/abs/1704.02685`.

[23] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *CoRR* abs/1703.01365 (2017). arXiv: `1703.01365`. URL: `http://arxiv.org/abs/1703.01365`.

[24] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. “Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR”. In: *CoRR* abs/1711.00399 (2017).

[25] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. “Interpretable Convolutional Neural Networks”. In: *CoRR* abs/1710.00935 (2017). arXiv: `1710.00935`. URL: `http://arxiv.org/abs/1710.00935`.

[26] Bolei Zhou et al. “Learning Deep Features for Discriminative Localization”. In: *CoRR* abs/1512.04150 (2015). arXiv: `1512.04150`. URL: `http://arxiv.org/abs/1512.04150`.