Understanding Support Vector Machine Classifications via a Recommender System-Like Approach

David Barbella[1], Sami Benzaid[2], Janara Christensen[3], Bret Jackson[4], X. Victor Qin[5], David Musicant[6]
[1]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA
[2]Department of Computer Science, University of North Carolina, Chapel Hill, NC, USA
[3]Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA
[4]Ultralingua, Inc., Minneapolis, MN, USA
[5]Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI, USA
[6]Department of Computer Science, Carleton College, Northfield, MN, USA

Abstract—*Support vector machines are a valuable tool for making classifications, but their black-box nature means that they lack the natural explanatory value that many other classifiers possess. Alternatively, many popular websites have shown recent success in explaining recommendations based on behavior of other users. Inspired by these ideas, we suggest two novel methods for providing insight into local classifications produced by a support vector machine. In the first, we report the support vectors most influential in the final classification for a particular test point. In the second, we determine which features of that test point would need to be altered (and by how much) in order to be placed on the separating surface between the two classifications. We call the latter technique "border classification." In addition to introducing these explanatory techniques, we also present a free-for-download software tool that enables users to visualize these insights graphically.*

Keywords: SVMs, classification, explanations, recommendations

1. Introduction

Support vector machines (SVMs) [1], [2] are a well-known supervised learning technique for performing binary classification. SVMs are very accurate and generalize well to a wide range of applications. Because support vector machines are "black-box" classifiers, the decisions they make are not always easily explainable. By this we mean that the model produced does not naturally provide any useful intuitive reasons about why a particular point is classified in one class rather than another. For instance, consider an SVM used by a bank to determine to whom they will loan money. If a customer's loan application is rejected and they would like to know why, then it is not very useful to only be able to say that the algorithm came back with a number lower than some required threshold. It would be much more satisfactory to the customer to be able to tell them that they were denied credit because their income is too low and they have six outstanding loans. Decision trees can provide a model that is much more easily interpreted, but cannot always achieve results as accurate as those produced by an SVM.

Recently Fung et al. [3] described the importance of understandable classifications for computer aided diagnosis (CAD). CAD systems provide analysis and interpretation of medical images. Although these systems have been shown to be highly successful in both research labs and clinical settings, doctors are reluctant to trust a diagnosis without receiving a convincing reason for it. Without understandable reasons for the diagnoses produced by these systems, they face problems when undergoing regulation and acceptance in the medical community.

Online recommendation systems have shown recent success in convincing users that the recommendations make sense [4]. Websites such as Amazon.com, MovieLens, Yahoo Launchcast, Netflix, and others explain recommendations in part by telling users "Other people that are similar to you have rated this item highly." This approach for explanations is easy to understand, and is becoming a familiar approach. Inspired in part by such systems, we propose two techniques to explain classifications provided by support vector machine classifiers for continuous data of relatively low dimensionality. A key distinction between our approach and other approaches for explaining SVM classifiers (see Section 2) is that our techniques explain decisions at the *local level*, i.e. for an individual test point. The first technique involves finding the most relevant support vectors for an individual point, that is those that were most influential in determining the class into which the point was placed. The second technique involves determining the change necessary in a test point's features to place that point on the separating surface between the two classes. The idea here is that any further change in this direction will place the point on the opposite side of the separating surface, and thus serves as an explanation of how much a test point would need to change in order to be classified in the other class. This technique, which we call border classification, is a variation on the inverse classification technique described below.

2. Related Work

Previous research done in the field of explaining support vector machine classifications includes work in sensitivity analysis [5], inverse classification [6], and rule extraction [3], [7]–[12]. Of these methods, rule extraction has received the most attention of late. Previously applied to neural networks [13], rule extraction applications have been extended to now include SVMs. SVM rule extraction techniques fall into two broad categories: pedagogical techniques and decomposition techniques [7]. Pedagogical techniques [8] extract rules relating the inputs and outputs of the model, using the trained model to generate training examples which can then be used by a comprehensible learning algorithm (such as decision trees) to create a comprehensible classification model. The principle is that the trained model can create examples that better represent the data than the original dataset which contains errors and noise. Decomposition techniques, on the other hand, are much more reliant on the structure of the SVM. For example, in one approach [3], the SVM hyperplane is approximated with axis-parallel surfaces. Although rule extraction has been known to produce classifiers that are easily comprehensible, they produce secondary models of worse accuracy. Moreover, even though these models may be reasonably understandable from a management perspective, they still lack the simplicity and familiarity to an individual user that recommendation systems have managed to provide. Therefore, instead of attempting to create a new, more easily explainable model based on the model made by the SVM, both of our proposed techniques explain the model made by the SVM directly for an individual point, eliminating the need for an additional model.

Two other methods of understanding classifiers are sensitivity analysis and inverse classification. Sensitivity analysis techniques measure the rate of change in the output of a model caused by the changes in the inputs of the model. This analysis can then be used to measure which input features are most important to achieve accurate output values [5]. Inverse classification techniques take an entirely different approach. Rather than attempting to explain the model, inverse classification describes how one point can be moved into another classification. More formally, the inverse classification problem consists of attempting to find the change in attribute values necessary to move a member of one classification into another classification. A set of prototype cases of each category is required, as well as specification of a similarity function. Our second proposed technique, border classification, is similar to inverse classification in that it finds the minimum change of attributes to switch the classification. However, instead of switching the point to another class, we move the point to the border between classes. Both inverse classification and border classification seek to solve an optimization problem that is highly nonlinear, and thus a global minimum value cannot readily be found. Previous work on inverse classification [6] uses genetic algorithms to solve the problem. We describe how to solve border classification as a nonlinear program, which ensures that the result will be at least a local minimum. Other nonlinear optimization strategies, including genetic algorithms, could be applied to our approach as well.

The key difference in the conceptualization of inverse classification and border classification can be alternatively explained by considering the algorithms directly. In the case of inverse classification, a set of test points in the opposite class from the test point under consideration is used to seed a genetic algorithm. The goal of the genetic algorithm is to evolve a series of points in this opposite class over a series of generations, and ultimately choose one that is relatively close to the original test point. Thus, the inverse classification paper describes inverse classification as that of finding a point in the opposite class [6]. In some sense, one would hope to do better than this: if one were to find "a point in the opposite class close to a test point," one could easily find a point even closer by moving in between that point and the separating surface. Therefore, we address this by formulating an optimization problem which constrains the solution to being on the separating surface itself.

Finally, Subianto et al. recently proposed a method for explaining classifiers whose input features are nominal [14]. This idea explains the classifier both at the local level (for an individual data point) and at the global level (for the entire model). The approach ensures local understandability by explaining why each test point was classified the way it was. More specifically, local explanations are defined as "a minimal set of attributes, such that there exists a change of values for the attributes in that set that would change the assigned class" [14]. This goal is similar to the idea of inverse classification and also to our border classification technique. The key difference is that the approach by Subianto et al. only works on discrete-valued datasets, and thus consists of finding a "nearest point" in the opposite class. Border classification differs from this approach in that it is defined for continuous-valued features, and thus it also differs (just as border classification differs from inverse classification) in that it finds a point on the separating surface, not on the opposite side. Subianto et al. also describe a technique for global explainability, which is maintained by attribute weights — i.e., the importance of an attribute in the global classifier in a similar fashion to sensitivity analysis. It is important to note that this method is not restricted to support vector machines, but can be used for any classifier. However, this method is limited by its restriction to discrete data.

3. Notation and Review of SVMs

In this section we define the notation that we will be using to describe support vector machines [1]–[3], [7], and related expressions. Suppose we have a set of m data points $\vec{x}_i \in \mathbb{R}^n$ along with each point's classification y_i, where y_i can take on one of two possible values: -1 or 1.

The linear support vector machine is defined as the following optimization problem:

$$\min_{\vec{w},b,\xi_i \geq 0} \quad \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^{m}\xi_i \\ \text{such that } y_i(\vec{w}\cdot\vec{x}_i - b) + \xi_i \geq 1 \tag{1}$$

where ξ_i is the error for a given training point $\vec{x}_i$, $\vec{w}$ is the vector of coefficients for the best separating hyperplane, b is the offset for that hyperplane, and C is a (usually experimentally determined) constant that represents the emphasis that is to be placed on minimizing the error. If C is too large, the emphasis on error avoidance can overwhelm the regularization term ($\|\vec{w}\|_2^2$) and result in overfitting. If C is too small, classification accuracy might be sacrificed.

It is often useful to consider the support vector machine in its equivalent dual formulation [1], [2]:

$$\max_{\alpha_i} \quad -\frac{1}{2}\sum_{i,j}^{m} y_i y_j \vec{x}_i\cdot\vec{x}_j \alpha_i\alpha_j + \sum_{i=1}^{m}\alpha_i \\ \text{such that } \sum_{i=1}^{m}\alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C \tag{2}$$

By replacing the dot product in the objective function above with a kernel function, we obtain the nonlinear support vector machine [1], [2]:

$$\max_{\alpha_i} \quad -\frac{1}{2}\sum_{i,j}^{m} y_i y_j K(\vec{x}_i,\vec{x}_j)\alpha_i\alpha_j + \sum_{i=1}^{m}\alpha_i \\ \text{such that } \textstyle\sum_{i=1}^{m}\alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C \tag{3}$$

Once this problem has been solved, α_i and b (which can be determined from the solution to this problem [1], [2]) can be used to classify test point $\vec{x}$ based on:

$$f(\vec{x}) = \text{sign}[\sum_{i=1}^{m}\alpha_i y_i K(\vec{x}_i,\vec{x}) - b] \tag{4}$$

which classifies $\vec{x}$ as either 1 or -1.

To interpret equation (4), we see that the dual variables α_i represent the importance of a particular support vector within the model. A high value of α_i associated with a particular support vector $\vec{x}_i$ indicates that the kernel value calculated using this support vector will have more influence on the final value for $f(\vec{x})$.

While the kernel function represents an implicit mapping of the points into a higher dimensional space, [1], [2], one can also interpret the kernel $K(\vec{x}_i,\vec{x}_j)$ as a similarity measure between points $\vec{x}_i$ and $\vec{x}_j$. Different kernels yield different interpretations of similarity. The most popular non-linear kernel in practice is the Gaussian radial basis kernel:

$$K(\vec{x}_i,\vec{x}_j) = e^{-\gamma||\vec{x}_i-\vec{x}_j||_2^2} \tag{5}$$

where γ is a user-chosen parameter. This kernel provides a natural sense of similarity, where a value of 1 indicates that the two points are identical, and a value of 0 indicates that the points are infinitely far apart. If we choose the dot product as the kernel function, we obtain the linear case of the SVM, as mentioned above. In this case similarity is understood to be related to the cosine of the angle between the two vectors. Since the Gaussian radial basis sense of similarity is somewhat more intuitive, we focus on it throughout our experiments, but our notions of explainability apply to any kernel function.

A number of free and high-quality software packages exist to solve this optimization problem. In our case, we have chosen to leverage SVM^{light} [15] for this purpose, as it is well-known and has several convenient features.

4. Finding important support vectors

When a support vector machine makes a classification, some of the support vectors are more influential than others. This is a function of the support vector's corresponding value of α and of its "similarity" to the test point as determined by the kernel function. Consider the case of an individual applying for a loan. If we are able to identify the support vectors with the largest numerical effect on the classification of the test point (in the direction of its final classification), they allow us to offer an explanation of the form "the individual was rejected for a loan in large part due to his or her similarity to these other individuals that were previously identified as bad credit risks." We therefore define the **pull** $p_i(\vec{x})$ of support vector $\vec{x}_i$ on test point $\vec{x}$ to be a measure of the role $\vec{x}_i$ played in categorizing $\vec{x}$. Specifically, let:

$$p_i(\vec{x}) = \alpha_i y_i K(\vec{x}_i,\vec{x}) \tag{6}$$

The support vectors with the largest pull values (in magnitude) are the ones that contribute the most to the classification that it receives, and it is these support vectors that provide an "explanation" for the classification for a given test point. The pull is based on the kernel-function similarity between the support vector and the test point, as well as on the α value, which is a measure of how important for classification the support vector is in general. Therefore, presenting a user with these particular support vectors (in an appropriate visualization environment) yields fairly simple information to help a user understand why a classification has been made as it has. In the credit example suggested in Section 1, the support vectors with the largest pull are those individuals who have a strong combination of two factors: they are generally good indicators of whether or not an individual similar to them should receive credit — they have high α values — and they are similar to the loan-seeker in the attributes used to construct the model.

A related problem is that of creating a "report set" of support vectors to report as "explanatory." One extreme is to report all of them, in a list ordered by pull. This list would be sorted in descending order if the test point $\vec{x}$ were classified in class 1, and ascending order if classified in -1; this would then place the most relevant support vectors at the top of the list. The other extreme is to simply report the support vector at the top of that list, which is the one single support vector that had the greatest pull in the direction of the test point's final classification. There are various compromises. After sorting the support vectors by pull values, one could simply pick the top n support vectors on that list. Alternatively, one could look for a transition point when the pull values changed significantly, and report all support vectors above this transition point. In our demonstration software (Section 6), we have chosen to present the top five support vectors on that list. Different applications might find different alternatives to be clearer.

The above approach for choosing explanatory support vectors makes more intuitive sense when using a kernel that always yields non-negative values, such as the Gaussian RBF kernel. In the case of this kernel in particular (and some others), the kernel can be easily thought of as a similarity metric. The top n explanatory support vectors are understood to be the support vectors that dominate due to large α values (the support vectors are globally important), large kernel values (they are similar to the test point), or some combination of the two. This interpretation is more complicated when a kernel that may produce negative values is used, such as the linear kernel. In this case, a support vector in the *opposite* class as the test point might actually sort near the top of the list. Users might consider this confusing; on the other hand, there is a natural interpretation. In the linear case, for example, a negative kernel value indicates some degree of difference between the test point and the support vector, as it requires that the angle between the two vectors is between 90 and 180 degrees. This situation could effectively be described as the support vector "pushing" the test point towards the other class. When using the linear kernel in our software, we currently suppress these "opposite-class" support vectors from being shown, but we believe with proper user training a case could be made for showing them.

5. Finding a nearby border point

Consider again the example of someone seeking a loan who has been rejected. One way of helping this individual understand why a rejection decision was made would be to provide the profile of a fictional individual that would be as similar as possible to this loan-seeker, yet would be just on the border of being "acceptable." Seeing the differences between oneself and a "an approximation to you that would get you accepted for a loan" could provide insight and believability to the classification.

Therefore, given a test point of one classification, we wish to find a nearby point to it on the separating surface. In principle, finding the closest point on the separating surface would be optimal, but due to the highly nonlinear nature of the problem this is not feasible in practice. We therefore focus on finding a point on the border whose distance is *locally* optimal. This nearby point provides a set of relatively minimal feature changes that would result in the point being ambiguously classified. The process of finding this nearby point, a process that we call "border classification," is found by solving a nonlinear constrained optimization problem. Let $\vec{x}$ be the test point in question, and let $\vec{a}$ be a closest-point candidate. We attempt to find the closest point to $\vec{x}$ while constraining $\vec{a}$ to lie on the separating surface. This can be done by minimizing the square of the Euclidean distance between the two points $\vec{a}$ and $\vec{x}$:

$$D^2(\vec{a},\vec{x}) = \sum_{j=1}^{n} (\vec{a}[j] - \vec{x}[j])^2 \tag{7}$$

Here $\vec{a}[j]$ is the j-th component of $\vec{a}$, and $\vec{x}[j]$ is the j-th component of $\vec{x}$. (Minimizing the squared distance simplifies the optimization problem by eliminating the square root that would otherwise be present.) Note that we have chosen to minimize the distance in the original input space, and not in the feature space represented by the kernel. This is intentional: the goal is to present a user with the (locally) minimal change necessary in order to reach the border. While the nearest border point in the feature space has relevance for classification, it doesn't well answer the question for the end-user of "what are the minimal changes that put me on the border?" In order for this information to be understood easily as a comparison, we argue that this calculation should be done in the input space. Of course, if one argued otherwise, our proposed technique could be modified to do so.

The minimization described in (7) is subject to the constraint that $\vec{a}$ lies on the separating surface. Based on the classifier given in (4), the equation for that surface is:

$$\sum_{i=1}^{m} \alpha_i y_i K(\vec{x}_i, \vec{a}) - b = 0. \tag{8}$$

One convenient side-effect of our formulation of this problem is that we can easily add additional constraints to enhance explainability by taking into account real-world restrictions. For example, certain features (such as age) may be difficult or impossible to change, and therefore one may desire a border classification point whose values for those features are constrained to be the same as for $\vec{x}$. A solution requiring a change in these "unchangeable" features might not be considered to be particularly useful as an explanation. Furthermore, features can be constrained in other ways to impose physical or other restrictions. For example, physical quantities might be restricted to non-negative values only.

To illustrate an example: suppose that for our loan-seeker, certain features such as age, salary, and outstanding loans might be constrained. It is not helpful to suggest to the user that changing age will help get a loan, as it is not a feature that can be readily adjusted. It is also impossible to earn a negative salary, or have negative outstanding loans. These constraints, combined with the equation for the separating surface, give us the following specific optimization problem:

$$\min_{\vec{a}} \sum_{j}^{n} (\vec{a}[j] - \vec{x}[j])^2, \tag{9}$$

Subject to the constraints:

$$\begin{aligned}\sum_{i=1}^{m} \alpha_i y_i K(\vec{x}_i, \vec{a}) - b &= 0 \\ \vec{a}[q] &= \vec{x}[q] \\ \vec{a}[r] &\geq 0 \\ \vec{a}[s] &\geq 0\end{aligned} \tag{10}$$

where q, r, and s are the indexes of the dimensions that correspond to age, salary, and outstanding loans, respectively.

One could go further by observing that, for example, it might be easier to pay off loans than to get one's salary increased. Therefore, weights could be added to the objective function (9) to do this. Though we have not implemented this in the current version of our software, it would be a straightforward extension.

In order to solve these minimization problems, the General Algebraic Modeling System (GAMS) was used. We read in the solution values for α_i and b from the model file generated by SVM^{light} during training, and combine them with the other information necessary to format an appropriate GAMS [16] input file with the appropriate variables, constants, and constraints designated. The non-linear solver CONOPT3 [17] is then used to perform a non-linear minimization on the optimization problem, and returns the given coordinates of the locally optimal point $\vec{a}$. We note that there are a variety of approaches that one might use to improve the quality of this local minimum, including simulated annealing, randomized restarts, genetic algorithms, and other strategies [18]. The inverse classification approach [6], for example, uses genetic algorithms to solve a related problem.

Once a particular value for the border classification point $\vec{a}$ is found, the features for both the user and this nearby point can then be compared in the graphical user interface, granting the user further insight.

We again point out that when solving this non-linear optimization problem, we achieve a distinct goal when compared to the work done on inverse classification using genetic algorithms [6]. Instead of evaluating discrete points of an opposite classification to find a minimal-cost result, we attempt to find a minimal solution by solving an optimization problem with constraints. Furthermore, we are interested in a point on the classification boundary, as opposed to some point of the opposite classification.

6. Results

In order to demonstrate the ideas presented above, we have created a software tool called "SVMzen." SVMzen is specifically targeted to the domain specialist rather than the machine learning specialist. The user interface, shown in a large screenshot at `www.cs.carleton.edu/faculty/dmusican/expsvms/screenshot.png`, was designed with the intent to be as easy to use and understand as possible. (We have not included the screenshot directly in this paper due to lack of space.) SVMzen serves as a graphical user interface to build classification models and classify test points (as do Weka [19], LIBSVM [20], and others), but it also displays the *explanations* for these classification models by implementing the proposed techniques.

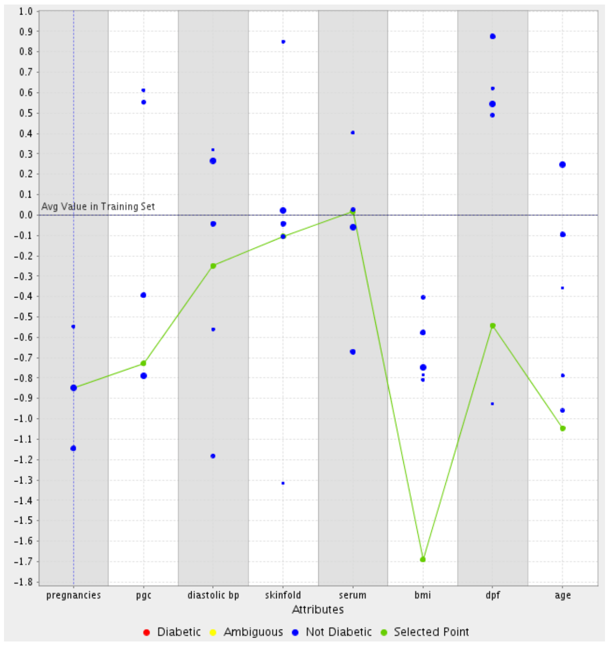

Figure 1: A non-diabetic patient and its five most influential support vectors.

After a user selects a training set file and chooses parameters for the SVM, SVMzen creates a model file by running SVM^{light} [15]. Next, the user can import a test set file and examine each of the test points individually in the user interface for local understanding of the model. For each selected test point, SVMzen displays the classification and the values for each of its attributes. It also displays an interactive chart which was implemented using jFreeChart [21]. This chart, seen in Figure 1, shows the values for each attribute of the support vectors that had the largest effect on classifying that particular point. Since each feature can have dramatically different scales and ranges, the chart we show has normalized values for each dimension, where we have subtracted the mean and divided by the standard deviation. Sliders to the left of the chart allow the user to modify the values of the test point's attributes and re-classify the point with the new values. This functionality allows the user to manually test the amount of change necessary for each attribute in order to change the classification, and also allows the user to see how sensitive each attribute is to change. Additionally, for a selected test point, SVMzen performs border classification in order to calculate changes that would give the test point an ambiguous classification.

In order to show examples of how the system works, we focus on the Pima Indian Diabetes dataset [22]. The dataset

Attribute	Values
Number of times pregnant	1.00
Plasma glucose	97.00
Diastolic blood pressure	64.00
Triceps skin fold (mm)	19.00
2-Hour serum insulin	82.00
Body mass index	18.2
Diabetes pedigree function	0.30
Age (years)	21.00

Table 1: Medical Data Test Point Values

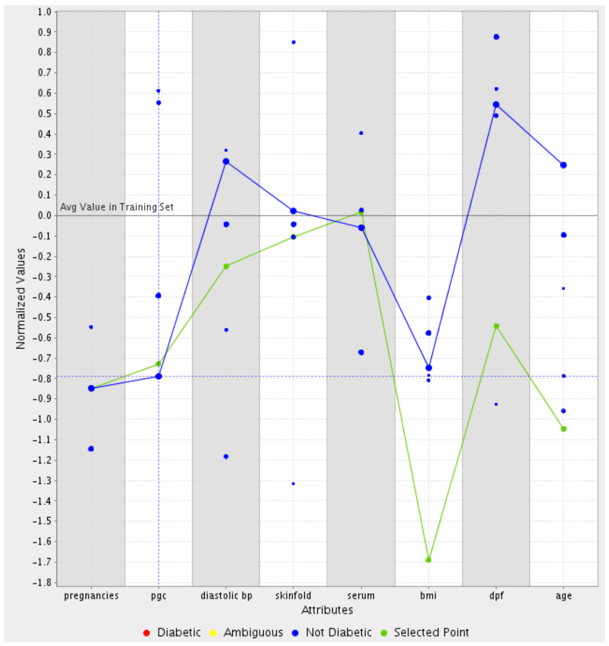

Figure 2: A non-diabetic point and its most influential support vector.

contains eight attributes (as specified in Table 1), and two classifications. We chose to work with the Gaussian radial basis function as a kernel. Based upon accuracies determined by SVM^{light}, we performed some rough experiments on training and test sets in order to pick values for γ and C that yielded reasonable results. For purposes of these visualizations, then, we chose γ to be 0.009, and C to be 1100. In practice, γ and C should be chosen to optimize tuning accuracy via a process such as tenfold cross validation.

We now choose a sample point from this dataset, and discuss the various ways in which SVMzen can help the user understand its classification. The particular point that we have chosen has the attributes listed in Table 1.

Examining the original chart that is generated by SVMzen (Fig. 1) allows us to infer a number of possible explanations for this classification. We notice that the majority of influential support vectors shown have comparatively similar values to our test point for three attributes (triceps skin fold, 2-hour serum insulin, and plasma glucose). Since they have the same classification as our test point, we might begin

Attribute	Suggested Value	Change From Original
Number of times pregnant	3.82	+2.82
Plasma glucose	149.65	+52.65
Diastolic blood pressure	68.84	+4.84
Triceps skin fold (mm)	27.13	+8.13
2-Hour serum insulin	79.96	-2.04
Body mass index	32.01	+13.81
Diabetes pedigree function	0.63	+0.33
Age (years)	21.00	N/A

Table 2: Medical Data Border Classification Values

to conclude that these may be characteristics of a patient with this classification. Our interface allows the user to explore the other features as well. For example, sliding the "pregnancies" slider to 13 (see above-mentioned web-based screenshot) will reclassify this point as diabetic.

Finally, the domain expert might wish to examine some of the risk factors associated with becoming diabetic in the context of this particular patient. Border classification, in a locally minimal fashion, determines the changes required to place the test point on the separating surface between the two classes. On the click of a button, SVMzen provides the values listed in Table 2. (Note that in a similar fashion to the example in Section 5, we have constrained the age of the patient to stay fixed.) This gives the domain expert insight into the patient's current risk of becoming diabetic by seeing what changes in the patient's medical records would cause a switch in classification. This also helps the domain expert confirm the diagnosis of a particular patient as the support vector machine is no longer completely a black box, but is now able to provide evidence for why patients are classified as they are.

7. Further Study

There are many directions that additional work on the topic could take. First, while selecting the support vectors with the most significant pull is relatively simple, it is not necessarily the best indicator of which support vectors were actually responsible for putting one particular test point in its current classification. Support vectors with high values of α_i may appear on the list of important support vectors virtually all of the time. In a sense, a test point would not be placed in a given class particularly because of the influence of those support vectors, as they have a strong pull in their direction for *any* test point. This situation would be particularly aggravated when using a Gaussian radial basis kernel with a very small γ, as this tends to homogenize the kernel values across all support vectors. Future work might consider approaches for "filtering out" those support vectors that provide a strong contribution to every test point, and not just to the one under consideration in particular.

Another potential direction for our approach would be to use it in order to determine which features are more important than others. In the current incarnation of our system, users can use the sliders in SVMzen's interface to modify the value of one feature at a time in order to observe what sorts of changes cause the point to be reclassified. Through such exploratory behavior, the user can gain some insight as to which features play a significant role in the classification. It would be considerably more desirable, however, to automate this process. We also note that our approach in its current form is most usable when the number of features of the dataset is of a size that the user can eyeball all at once (perhaps 25-30 or so). With an automated approach for determining important features, as suggested above, one might show the user only the particularly significant ones, allowing SVMzen to be used on datasets with considerably larger numbers of features.

There are other classification algorithms that share enough similarities with SVMs such that they might also benefit from methods similar to those presented in this paper. Boosting, for example, also results in a classifier that is a weighted combination of terms [23], [24]. The boosted classifier is not a linear combination of kernel functions; rather, it is a combination of simple classifiers. One could look at the "pull" of each simple classifier, attempting again to use it to provide some interpretation as to which simple classifiers played a significant role in classifying an individual point.

There are numerous other variations of support vector machines, and the procedures described here could easily be adapted to many of them. For example, it would be possible to use a similar process to explain the decisions produced by a support vector machine with more than two classes.

8. Conclusion

In this paper we introduce two new techniques for explaining support vector machines on continuous data. Both techniques explain the model on the local level, i.e. for an individual test point, as a recommender system might. The first technique involves finding the support vectors that make the strongest contribution to the classification of a particular test point. The second technique is similar to inverse classification: the goal is to find a relatively minimal change in order to switch the classification of a test point. However, instead of minimally switching the classification (which is not optimally possible in a continuous space), we propose finding the locally minimal change required to move the point to the separating surface of the classes. These techniques add explainability to the results of an SVM classifier in a format with which users of online recommendation systems are quite familiar. We have presented a software tool named SVMzen that allows users to see these explanations graphically. A user can look at a particular test point and determine that the test point was classified in that class due to a specific group of highly weighted support vectors. Furthermore, the user can examine what changes would be necessary to move the test point onto the edge between the two classes. Additionally, a person who wishes to be placed in the other classification can be told what changes are necessary for that result, which provides prescriptive advice for that individual.

More info about SVMzen is available at `www.cs.carleton.edu/faculty/dmusican/expsvms/`.

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[2] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge: Cambridge University Press, 2000.

[3] G. Fung, S. Sandilya, and R. B. Rao, "Rule extraction from linear support vector machines," *Proceedings of the 11th Intl. Conference on Knowledge Discovery in Data Mining*, pp. 38–40, 2005.

[4] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in *CSCW*, 2000, pp. 241–250.

[5] J. Yao, "Sensitivity analysis for data mining," 2003. [Online]. Available: citeseer.ist.psu.edu/yao03sensitivity.html

[6] M. V. Mannino and M. V. Koushik, "The cost-minimizing inverse classification problem: a genetic algorithm approach," *Decision Support Systems*, vol. 29, no. 3, pp. 283–300, October 2000.

[7] D. Martens, B. Baesens, T. van Gestel, and J. Vanthienen, "Comprehensible credit scoring models using rule extraction from support vector machines," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1466–1476, 2007.

[8] H. Núñez, C. Angulo, and A. Català, "Rule-based learning systems for support vector machines," *Neural Processing Letters*, vol. 24, no. 1, pp. 1–18, 2006.

[9] J. Diederich, Ed., *Rule Extraction from Support Vector Machines*, ser. Studies in Computational Intelligence. Springer, 2008, vol. 80.

[10] N. H. Barakat and A. P. Bradley, "Rule extraction from support vector machines: A sequential covering approach," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 6, pp. 729–741, 2007.

[11] N. H. Barakat and J. Diederich, "Eclectic rule-extraction from support vector machines," *International Journal of Computational Intelligence*, vol. 2, no. 1, pp. 59–62, 2005.

[12] D. Martens, B. Baesens, and T. V. Gestel, "Decompositional rule extraction from support vector machines by active learning," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 2, pp. 178–191, 2009.

[13] R. Andrews, J. Diederich, and A. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-Based Systems*, vol. 8, pp. 373–389, 1995.

[14] M. Subianto and A. Siebes, "Understanding discrete classifiers with a case study in gene prediction," *Proceedings of the Seventh IEEE International Conference on Data Mining*, 2007.

[15] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. J. C. Burges and A. J. Smola, Ed. MIT Press, 1999, pp. 169–184.

[16] GAMS Development Corporation, "GAMS."

[17] ARKI Consulting & Development, "CONOPT 3.1 Solver."

[18] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. New Jersey: Pearson Education, Inc., 2003.

[19] G. Holmes, A. Donkin, and I. H. Witten, "Weka: a machine learning workbench," in *Proceedings of the Second Australian and New Zealand Conference on Intelligent Information Systems*, Brisbane, Australia, 1994, pp. 357–361.

[20] C.-C. Chang and C.-J. Lin, "LIBSVM Software," 2003, available at http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[21] D. Gilbert, *The JFreeChart Class Library*. Object Refinery Limited, 2008.

[22] A. Asuncion and D. Newman, "UCI Machine Learning Repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[23] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," *Advanced Lectures on Machine Learning (LNAI2600)*, 2003.

[24] R. E. Schapire, "A brief introduction to boosting," *Proceedings of the Sixteenth Intl. Joint Conference on Artificial Intelligence*, 1999.