 doi:10.20944/preprints202201.0232.v1

Tutorial on Support Vector Machines

Raj Bridgelall, Ph.D.
Transportation, Logistics & Finance, College of Business
North Dakota State University
PO Box 6050, Fargo ND 58108-6050
Email: raj@bridgelall.com
ORCiD: 0000-0003-3743-6652

Abstract

The aim of this tutorial is to help students grasp the theory and applicability of support vector machines (SVMs). The contribution is an intuitive style tutorial that helped students gain insights into SVM from a unique perspective. An internet search will reveal many videos and articles on SVM, but many of them give simplified explanations that leave gaps in the derivations that beginning students cannot fill. Most free tutorials lack guidance on practical applications and considerations. The software wrappers in popular programming libraries such as Python and R hide many of the operational complexities. Free software tools often use default parameters that ignore domain knowledge or leave knowledge gaps about the important effects of SVM hyperparameters, resulting in misuse and subpar outcomes. The author uses this tutorial as a course reference for students studying artificial intelligence and machine learning. The tutorial derives the classic SVM classifier from first principles and then derives the practical form that a computer uses to train a classification model. An intuitive explanation about confusion matrices, F1 score, and the AUC metric extend insights into the inherent tradeoff between sensitivity and specificity. A discussion about cross-validation provides a basic understanding of hyperparameter tuning to maximize generalization by balancing underfitting and overfitting. Even seasoned self-learners with advanced statistical backgrounds have gained insights from this tutorial style of intuitive explanations, with all related considerations for tuning and performance evaluations in one place.

Keywords: artificial intelligence; data science; Kernel trick; machine learning; pedagogy

Declarations: None

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

1 Introduction

Data scientists can apply support vector machines (SVMs) to solve both regression and classification problems. This tutorial focuses on the SVM classifier, which is a non-probabilistic ***binary*** classifier. The non-probabilistic aspect is in contrast with probabilistic classifiers, such as the Naïve Bayes, that compute class membership likelihood based on the training examples (Aggarwal 2015). An SVM separates data across a decision boundary, which is a plane in multidimensional feature space. Only a small subset of the data touches or supports the decision boundary, which is why the inventors named them ***support vectors***.

An SVM cannot classify data into more than two classes. Handling multiclass data with SVMs is still an active area of research. Nevertheless, there are some workarounds. Methods involve creating multiple SVMs that compare feature vectors among themselves by using various techniques such as one-versus-Rest (OVR) or one-versus-one (OVO) (Bhavsar and Ganatra 2012). For *k* classes, the OVR method trains *k* classifiers so that each class discriminates against the remaining *k*-1 classes. OVO creates one binary classification problem for all possible pairings of classes, so it requires *k*(*k*-1)/2 classifiers. After constructing the number of required binary classifiers for either the OVR or OVO methods, the algorithm classifies a new object according the majority vote among the set of classifiers.

The SVM is a supervised machine learning (ML) method where each data point is a set of features $\{x_1 \ldots x_n\}$ and a class label y_i. SVM treats each data object as a point in feature space that belongs to one of only two classes. SVM defines the class labels are either $y_i = 1$ or $y_i = -1$. Hence, the mathematical representation of the dataset is

$$\text{Data} = \{(x_i, y_i) | x_i \in \Re^p, y_i \in (-1, +1)\}_{i=1}^{n} \quad (1)$$

where *p* is the dimension of the feature vector and *n* is the number of vectors.

During training, the SVM classifier finds a linear decision boundary in feature space that best separates the data objects into the two classes. The equivalent optimization problem finds two parallel *hyperplanes* that form the widest gap that is void of data objects. A hyperplane is a subspace with one dimension less than the ambient space. The perpendicular distance between the parallel hyperplanes is the margin. The hyperplane that equidistantly bifurcates the space between the parallel hyperplanes defines a multidimensional decision boundary that separates the data into two parts.

2 Methods

This first subsection introduces the notion of a hyperplane in mathematical terms to help setup the optimization problem that an SVM solves. The second subsection presents the practical implementation of an SVM by introducing Lagrangian multipliers, Kernels, and slack variables that make them suitable for real-world data. The third subsection discusses various scoring metrics that users must consider when evaluating the performance of the trained SVM classifier.

2.1 Classifier Derivation

2.1.1 Hyperplanes

An SVM uses a linear hyperplane rather than a non-linear one because in practice the latter will tend to too tightly fit (overfit) a boundary that would perfectly separate the training data, but not necessarily new data. That is, overfitting the model on the training data can cause the classifier to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

generalize poorly by inaccurately predicting the class of new data (Aggarwal 2015). The general equation for a linear hyperplane is

$$\boldsymbol{w} \bullet \boldsymbol{x} = 0 \tag{2}$$

A familiar hyperplane in 2D space is a line

$$y = ax + b. \tag{3}$$

Rewritten in the standard form for a hyperplane gives

$$y - ax - b = 0. \tag{4}$$

The equivalent vectors are

$$\boldsymbol{w} = \begin{pmatrix} -b \\ -a \\ 1 \end{pmatrix} \quad \text{and} \quad \mathbf{x} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix} \tag{5}$$

where $w_0 = -b$, $w_1 = -a$, and $w_2 = 1$. So, taking the dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$ and setting that equal to zero produces the standard form equation for a line. Specific values for the vector components of $\boldsymbol{w}$ defines a specific line (linear hyperplane) and any point defined by an (x, y), which is the feature vector, must be located somewhere on that line. Given that the vectors are column vectors, the dot product is equivalent to the matrix operation

$$\boldsymbol{w}^T \boldsymbol{x} = 0. \tag{6}$$

The vector and matrix forms are easier to represent in computer memory for rapid computations.

The dot product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x} \bullet \boldsymbol{y} = \|\boldsymbol{x}\| \times \|\boldsymbol{y}\| \times cos(\theta) = \sum_i x_i y_i = |\boldsymbol{x}\boldsymbol{y}^T| = \alpha \tag{7}$$

which is a scalar value. Hence, for the dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$ to be zero, the vectors must be perpendicular to each other because cos(90) = 0. That is, if $\boldsymbol{x}$ lies on a plane of the space, then the $\boldsymbol{w}$ vector that defines that plane must be perpendicular (normal) to both $\boldsymbol{x}$ and the plane. Hence, the unit vector

$$\boldsymbol{u} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \tag{8}$$

where $\|\boldsymbol{w}\|$ is the vector norm or length must also be perpendicular to the plane. For the 2D example,

$$\boldsymbol{u} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \left(\frac{w_1}{\|\boldsymbol{w}\|}, \frac{w_2}{\|\boldsymbol{w}\|} \right). \tag{9}$$

This unit vector of $\boldsymbol{w}$ is important for finding the distance of any point (feature) from the hyperplane by projecting that point to a vector that is normal to the hyperplane. For example, vector **p** in Figure 1 is the projection of point A onto the plane of the $\boldsymbol{w}$ vector which is normal to the hyperplane. Hence the distance from point A to the hyperplane is the same as the length of **p**, which is ||**p**||. In this example, the projection of vector **a** onto the plane of $\boldsymbol{w}$ is

$$\boldsymbol{p} = (\boldsymbol{u} \bullet \boldsymbol{a})\boldsymbol{u} \tag{10}$$

The dot product produces a scalar, which is the magnitude (length) of the vector such that

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$\boldsymbol{u} \bullet \boldsymbol{a} = \sum_i u_i a_i \tag{11}$$

and the direction of the vector is $\boldsymbol{u}$.

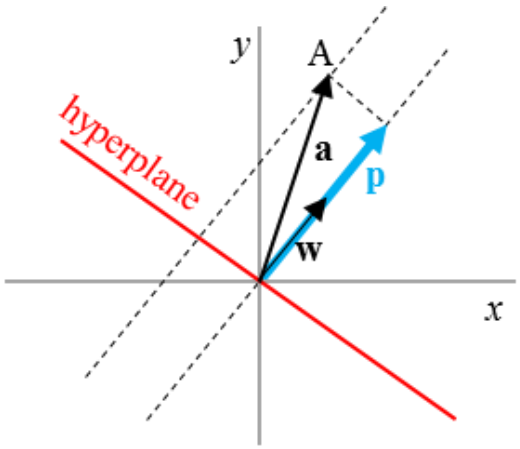

Figure 1: Projection of a vector to compute the distance to a hyperplane.

2.1.2 Margin Optimization

The illustration of Figure 2 shows that the region bounded by the two hyperplanes H1 and H2 is void of data points. The "support vectors" are the data objects at the boundaries of the two hyperplanes. Figure 2 highlights the four support vectors (points) of this example with thick black borders. The decision boundary is a linear hyperplane H0 that is equidistant between the two hyperplanes H1 and H2. The hyperplane margin is the separation distance between the two hyperplanes H1 and H2. Equivalently, hyperplanes H1 and H2 are offset by some amount δ from the decision boundary hyperplane H0. Associating $\boldsymbol{w}$ with the hyperplane H0, the parallel hyperplane with offset +δ to one side is

$$\boldsymbol{w} \bullet \boldsymbol{x} = +\delta \tag{12}$$

and an equal and opposite offset to the other side of the hyperplane is

$$\boldsymbol{w} \bullet \boldsymbol{x} = -\delta \cdot \tag{13}$$

Given a feature vector $\mathbf{x}_i$, the assigned class must satisfy

$$\boldsymbol{w} \bullet \boldsymbol{x}_i \geq +\delta \tag{14}$$

for a class label of $y_i = 1$ and

$$\boldsymbol{w} \bullet \boldsymbol{x}_i \leq -\delta \tag{15}$$

for a class label of $y_i = -1$. Solving the optimization problem will find the optimum $\boldsymbol{w}$ that maximizes the offset or margin. For mathematical convenience, setting the offset to $\delta = 1$ facilitates combination of the two classification constraints into a single constraint. That is,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

multiplying both sides of equation (14) by y_i, and assigning the class label value of +1 to the right yields

$$y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i) \geq +1(+1) \Leftrightarrow y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i) \geq 1 \tag{16}$$

Doing the same for equation (15) with its y_i label value at -1 yields

$$y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i) \leq -1(-1) \Leftrightarrow y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i) \geq 1 \tag{17}$$

Note that per rule for inequalities, multiplying by -1 flips the inequality sign. Consequently, the single constraint of the optimization problem becomes

$$y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i) \geq 1 \quad \forall \quad 1 \leq i \leq n \cdot \tag{18}$$

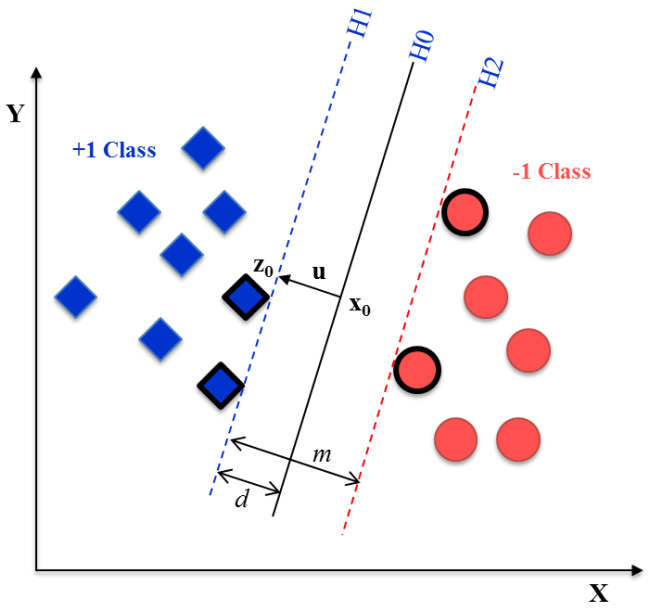

Figure 2: Maximizing the margin of a hyperplane.

Given that the vector $\boldsymbol{w}$ is perpendicular (normal) to the hyperplane H0, its unit vector $\boldsymbol{u} = \boldsymbol{w}/\|\boldsymbol{w}\|$ must be perpendicular to H0 with magnitude 1. The vector $d\boldsymbol{u}$ is the perpendicular vector from hyperplane H0 to a parallel hyperplane H1 some distance d away. Let $\boldsymbol{x}_0$ be the base coordinate of the $d\boldsymbol{u}$ vector on the hyperplane, and $\boldsymbol{z}_0$ be the tip coordinate that lies on the hyperplane H1. The distance vector is

$$\boldsymbol{z}_0 = \boldsymbol{x}_0 + d\boldsymbol{u} \tag{19}$$

The fact that $\boldsymbol{z}_0$ is on H1 means that

$$\boldsymbol{w} \bullet \boldsymbol{z}_0 = +1 \tag{20}$$

Multiplying both sides of Equation (19) by the dot product of $\boldsymbol{w}$ and substituting $\boldsymbol{z}_0$ yields

$$\boldsymbol{w} \bullet (\boldsymbol{x}_0 + d\boldsymbol{u}) = +1 \tag{21}$$

Substituting $\boldsymbol{u}$ from Equation (8) yields

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$\boldsymbol{w} \bullet \left(\boldsymbol{x}_0 + d \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right) = +1 \tag{22}$$

Expanding Equation (22) yields

$$\boldsymbol{w} \bullet \boldsymbol{x}_0 + d \frac{\boldsymbol{w} \bullet \boldsymbol{w}}{\|\boldsymbol{w}\|} = +1 \tag{23}$$

Given that

$$\boldsymbol{w} \bullet \boldsymbol{w} = \|\boldsymbol{w}\|^2 \tag{24}$$

Equation (23) becomes

$$\boldsymbol{w} \bullet \boldsymbol{x}_0 + d\|\boldsymbol{w}\| = +1 \tag{25}$$

Hence,

$$\boldsymbol{w} \bullet \boldsymbol{x}_0 = 1 - d\|\boldsymbol{w}\| \tag{26}$$

The fact that $\boldsymbol{x}_0$ is on H0 means that

$$\boldsymbol{w} \bullet \boldsymbol{x}_0 = 0 \tag{27}$$

Substituting Equation (27) into Equation (26) yields

$$0 = 1 - d\|\boldsymbol{w}\| \tag{28}$$

Solving for distance *d* yields

$$d = \frac{1}{\|\boldsymbol{w}\|} \tag{29}$$

Therefore, the margin, which is the distance between the hyperplanes H1 and H2 must be 2/||**w**||. Hence the margin is

$$m = \frac{2}{\|\boldsymbol{w}\|} \tag{30}$$

Note again that using the class labels of +1 and -1 as the decision boundaries for vectors above and below H1, respectively, allowed for a simple derivation of the margin in terms of only the length of the ***w*** vector. From Equation (30) the optimization problem of maximizing the margin becomes equivalent to minimizing the length of the ***w*** vector. Hence, the optimization problem is to minimize the norm of ***w*** subject to the constraint of Equation (18), which keeps the hyperplane void of data objects. That is, the optimization problem becomes

$$\begin{aligned} &\min_{} f(w) = \|\boldsymbol{w}\| \\ &s.t. \qquad g: y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i + w_0) \geq 1, \quad 1 \leq i \leq n. \end{aligned} \tag{31}$$

Note that this notation for the constraint separates w_0, the bias component, from the ***w*** vector, which means that the iteration indices must be between 1 and *n* rather than 0 and *n*. For intuition, one can mentally visualize the optimization in 2D space as finding the orientation and maximum size of a rectangle that is void of data points.

Upon converging to the optimum solution, the trained SVM classifier becomes

$$\boldsymbol{x} \mapsto signum(\boldsymbol{w} \bullet \boldsymbol{x}) \tag{32}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

That is, the predicted class label for a new feature vector $\boldsymbol{x}$ is the sign of the dot product between the $\boldsymbol{w}$ vector and the new vector. The signum function produces -1 if its argument is negative (less than zero), 0 if its argument is equal to 0, and +1 if its argument is positive (greater than zero).

2.2 Practical Implementation

A recasting of the minimization optimization problem presented in Equation (31) results in the *primal* problem, which is a *quadratic* programming problem with constraints as

$$\begin{aligned} &\underset{min}{f(w)} = \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &s.t. \qquad g: y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i + w_0) - 1 = 0, \quad 1 \le i \le n \cdot \end{aligned} \tag{33}$$

Note that the squaring of the length of the $\boldsymbol{w}$ vector produces a parabola with a single minimum. The ½ factor is for the mathematical convenience of cancellation when taking the partial derivative with respect to $\boldsymbol{w}$ such as in a gradient descent search.

2.2.1 Lagrangian Dual Form

The primal quadratic form can become impractical to solve with large datasets because the optimization space considers the dot product of $\boldsymbol{w}$ and all the training. Fortunately, researchers discovered that a dual form based on Lagrangian multipliers provides a practical alternative (Aggarwal 2015). Some benefits of the dual form are a reduction of computational resources and direct support for using Kernel functions to find suitable hyperplanes that can separate non-linearly separable data. The dual problem requires learning only the number of support vectors, which can be significantly fewer than the number of feature space dimensions.

Constructing the dual form involves constructing the Lagrange (see the appendix), by combining both the objective function $f(\boldsymbol{w})$ and the equality constraint $g(\boldsymbol{w})$ such that the Lagrangian function $L(\boldsymbol{w}) = f(\boldsymbol{w}) - \lambda\, g(\boldsymbol{w})$ with λ being the so-called Lagrangian multiplier. Hence, the objective function

$$f \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{34}$$

and the constraint

$$g \equiv y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i + w_0) - 1 = 0 \tag{35}$$

combines to yield the Lagrangian formulation

$$L(\alpha, \boldsymbol{w}, w_0) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1} \alpha_i [y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i + w_0) - 1], \quad \alpha_i \ge 0 \quad \forall i \tag{36}$$

where the α_i parameters are the Lagrange multipliers. Note that the Lagrange is a function of only the $\boldsymbol{w}$ and the α_i parameters because the data objects $\mathbf{x}_i$ and labels y_i are known from the data. Expanding the summation yields

$$L(\alpha, \boldsymbol{w}, w_0) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1} \alpha_i y_i(\boldsymbol{w} \bullet \boldsymbol{x}_i + w_0) + \sum_{i=1} \alpha_i, \quad \alpha_i \ge 0 \quad \forall i \tag{37}$$

The maximum (or minimum) solutions of the Lagrangian that satisfies the constraints are where the gradient is zero such that

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$\nabla L(\alpha, \boldsymbol{w}, w_0) = 0 \cdot \tag{38}$$

Finding the gradient requires taking the partial derivatives with respect to each variable, setting them equal to zero, and then solving the resulting simultaneous equations. So, differentiating with respect to $\boldsymbol{w}$ yields

$$\frac{\partial}{\partial w} L(\alpha, \boldsymbol{w}, w_0) = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x}_i = 0 \tag{39}$$

Rearranging terms yields a solution for $\boldsymbol{w}$ as

$$\boldsymbol{w} = \sum_{i=1} \alpha_i y_i \boldsymbol{x}_i \,, \quad \alpha_i \geq 0 \quad \forall i \tag{40}$$

Then differentiating with respect to the bias component w_0 yields

$$\frac{\partial}{\partial w_0} L(\alpha, \boldsymbol{w}, w_0) = \sum_{i=1} \alpha_i y_i = 0 \tag{41}$$

This solution forms a constraint

$$\sum_{i=1} \alpha_i y_i = 0, \quad \alpha_i \geq 0 \tag{42}$$

Substituting the solutions for $\boldsymbol{w}$ from Equation (40) and the constraint from Equation (42) back into the Lagrangian Equation (37) yields

$$L(\alpha) = \sum_i \alpha_i + \frac{1}{2} \left\| \sum_i \alpha_i y_i \boldsymbol{x}_i \right\|^2 - \sum_i \alpha_i y_i \left(\sum_i \alpha_i y_i \boldsymbol{x}_i \bullet \boldsymbol{x}_i \right), \quad \alpha_i \geq 0 \quad \forall i \tag{43}$$

A simplification of the last remaining term is

$$\sum_i \alpha_i y_i \left(\sum_i \alpha_i y_i \boldsymbol{x}_i \bullet \boldsymbol{x}_i \right) = \sum_i \alpha_i y_i \boldsymbol{x}_i \left(\sum_j \alpha_j y_j \boldsymbol{x}_j \right) = \|\boldsymbol{V}\|^2 \tag{44}$$

The arbitrary assignment to vector **V** simplifies the remaining algebra. That is

$$V = \sum_i (\alpha_i y_i \boldsymbol{x}_i) \tag{45}$$

and

$$\|V\| = \sqrt{\sum_i (\alpha_i y_i \boldsymbol{x}_i)^2} \tag{46}$$

Hence, squaring $\|\boldsymbol{V}\|^2$ removes the square-root. This gives a simplified notation for the Lagrangian of Equation (43) as

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$L(\alpha, x, y) = \sum_i \alpha_i + \frac{1}{2}\|V\|^2 - \|V\|^2 = \sum_i \alpha_i - \frac{1}{2}\|V\|^2 \tag{47}$$

Expanding the norm-squared vector yields

$$\begin{aligned} \|V\|^2 &= \left\|\sum_i \alpha_i y_i \boldsymbol{x}_i\right\|^2 = \left\{\sum_i \alpha_i y_i \boldsymbol{x}_i\right\}^T \left\{\sum_j \alpha_j y_j \boldsymbol{x}_j\right\} = \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\ &= \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \bullet \boldsymbol{x}_j \end{aligned} \tag{48}$$

So, the final Lagrangian expression becomes

$$L(\alpha, x, y) = \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \bullet \boldsymbol{x}_j\,, \quad \alpha_i \geq 0 \quad \forall i \tag{49}$$

Finally, the optimization problem now becomes

$$\begin{aligned} &\underset{max}{L(\alpha)} = \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_i \bullet \boldsymbol{x}_j\right) \\ &\text{subject to:} \qquad \alpha_i \geq 0 \\ &\text{and} \qquad \sum_i \alpha_i y_i = 0 \end{aligned} \tag{50}$$

Substituting the solution for $\boldsymbol{w}$ into Equation (32) gives the classifier as

$$\boldsymbol{x} \mapsto signum\left(\sum_{i=1} \alpha_i y_i (\boldsymbol{x}_i \bullet \boldsymbol{x}) + w_0\right) \tag{51}$$

The dual form transforms the optimization problem to one of learning the α_i parameters, given the training data $\boldsymbol{x}_i$ and their classifications y_i. A key feature of Equation (40) is that $\alpha_i = 0$ for many of the vectors, except for the support vectors. The non-zero parameters are those of the support vectors.

2.2.2 Kernel Trick

Some data is not separable using linear hyperplanes. Fortunately, transforming non-linearly separable data to a higher dimension plane can allow the SVM to separate them with a linear hyperplane in the new space. The transformation function $\Phi(\boldsymbol{x})$ creates new vectors by mapping the input vectors to a higher dimensional feature space. The transform does so by operating on at least one or combinations of the lower or ambient dimension features. For example, the exponential transform can create infinite feature dimensions because the exponential is an infinite series that operates on a single dimension z such that

$$\Phi(z) = \exp(z) = \sum_{k=0}^{\infty} \frac{z^k}{k!}. \tag{52}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

Applying the transformation function changes the training problem to the optimization

$$\begin{aligned} \underset{max}{L(\alpha)} &= \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1} \alpha_i \alpha_j y_i y_j [\Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j)] \\ \text{subject to:} \quad & \alpha_i \geq 0 \\ \text{and} \quad & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{53}$$

and the classification problem to

$$\boldsymbol{x} \mapsto signum\left(\sum_{i=1} \alpha_i y_i [\Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x})] + w_0\right). \tag{54}$$

As with the non-transformed vectors, the dot product of the transformed vectors produces a scalar value called the SVM score, and the sign yields the class label.

The core idea behind using a Kernel is that the problem of first transforming each vector to a higher dimension and then taking the dot product $\Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x})$ becomes equivalent to applying a special Kernel function to the original vectors that achieves the same thing where

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j) \tag{55}$$

The ***trick*** aspect is that the Kernel achieves the same result as the dot product of the transformed vectors without computing the transforms. This "trick" saves computing time and lowers memory requirements.

As a simple example of how the Kernel works, consider $\Phi(\boldsymbol{x})$ to be a transform that maps from 2D to 3D space as

$$\Phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1 x_2, x_2^2\right) \tag{56}$$

The new hyperplane will be

$$\boldsymbol{w} \bullet \boldsymbol{x} = w_0 + w_1 x_1^2 + w_2\sqrt{2}x_1 x_2 + w_3 x_2^2 = 0 \tag{57}$$

which is an ellipse. Figure 3a is a plot of the transformation function in 3D space with $w_0 = 0$ and $\mathbf{w} = [1, 1, 1]$.

Figure 3b plots some possible hyperplanes in the 2D space. The possible hyperplanes are projections of the contour lines of the 3D transform function onto the 2D space. The contour lines are at the intersections of the function with linear 2D hyperplanes. Notice that the projected hyperplane in 2D space forms a non-linear decision boundary to separate the data in the ambient 2D feature space. To derive the Kernel function, let $\Phi(\boldsymbol{r})$ be the mapping function of vector $\boldsymbol{r}$ such that

$$\Phi(\boldsymbol{r}) = \left(r_1^2, \sqrt{2}r_1 r_2, r_2^2\right) \tag{58}$$

The dot product of the two transformed vectors is

$$\Phi(r_1, r_2) \bullet \Phi(x_1, x_2) = r_1^2 x_1^2 + 2r_1 r_2 x_1 x_2 + r_2^2 x_2^2 \tag{59}$$

and when reduced becomes

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$\Phi(r_1, r_2) \bullet \Phi(x_1, x_2) = (r_1 x_1)^2 + 2(r_1 x_1)(r_2 x_2) + (r_2 x_2)^2 = (r_1 x_1 + r_2 x_2)^2 = (\boldsymbol{r} \bullet \boldsymbol{x})^2 \quad (60)$$

Hence, the Kernel function is

$$K(\boldsymbol{r}, \boldsymbol{x}) = (\boldsymbol{r} \bullet \boldsymbol{x})^2 = \Phi(\boldsymbol{r}) \bullet \Phi(\boldsymbol{x}) \quad (61)$$

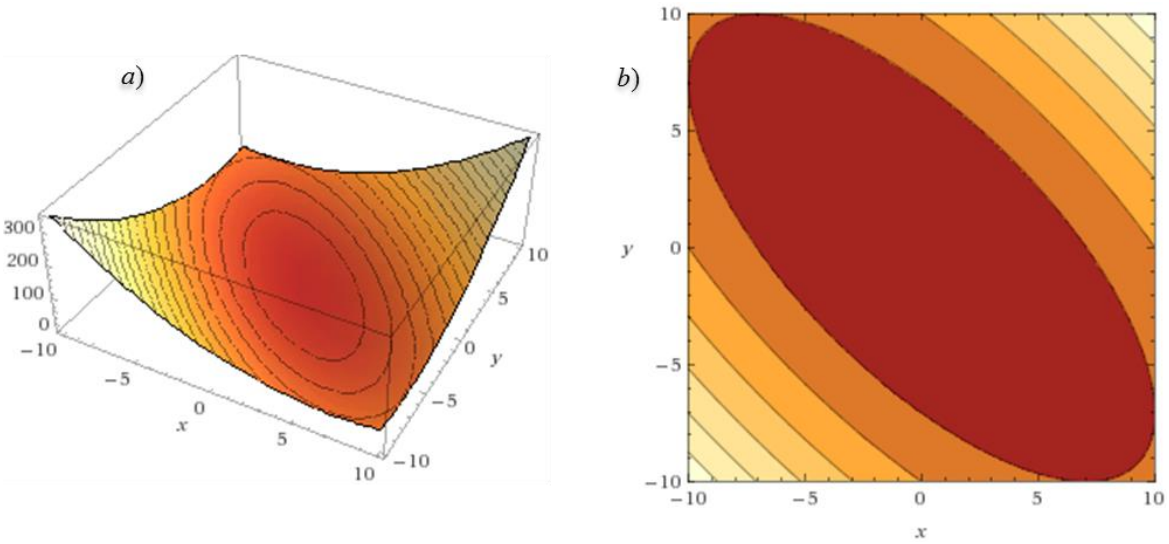

Figure 3: a) Function in 3D space and b) hyperplanes in 2D space.

This example shows that the Kernel function is equivalent to the square of a dot product of the feature vectors, which achieves the same function as computing the dot product of the transformed feature vectors in a higher dimensional space.

To generalize, the optimization problem in SVM training becomes

$$\begin{aligned} &\underset{max}{L(\alpha)} = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &s.t. \qquad \alpha_i \geq 0, \quad i = 1, \ldots, m \\ &\qquad \sum_i \alpha_i y_i = 0 \end{aligned} \quad (62)$$

and the classification becomes

$$\boldsymbol{x} \mapsto signum\left(\sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})\right). \quad (63)$$

Note that both the training and classification problems must use the same Kernel function. The general solution uses linear Kernel functions for data that is linearly separable. Theoreticians have worked out several types of Kernels. The following are a few examples:

Sigmoid

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

$$K(\boldsymbol{u},\boldsymbol{v}) = exp\left(-\frac{\|\boldsymbol{u}-\boldsymbol{v}\|}{2\sigma^2}\right) \tag{64}$$

Polynomial

$$K(\boldsymbol{u},\boldsymbol{v}) = (\gamma \cdot \boldsymbol{u} \bullet \boldsymbol{v} + r)^d \tag{65}$$

Radial basis function (RBF)

$$K(\boldsymbol{u},\boldsymbol{v}) = exp(-\gamma \cdot \|\boldsymbol{u}-\boldsymbol{v}\|^2) \tag{66}$$

2.2.3 Slack Variables

Data may not be completely separable in some cases, particularly if noisy. That is, some points will lie within the margin or they may not be correctly classified. Therefore, practical implementations of SVMs allow for some slackness in the classification by introducing an adjustable cost penalty hyperparameter *C*. The design is such that if $C = 0$, the SVM allows all errors. If $C \to \infty$ then the SVM does not tolerate the errors. Hence, SVM users must set the hyperparameter *C* to some finite value between 0 and infinity, with $C > 0$. The classifier with slackness is

$$\begin{aligned} \boldsymbol{w} \bullet \boldsymbol{x}_i &\geq +1 - \varepsilon_i \quad \text{for } y_i = +1 \\ \boldsymbol{w} \bullet \boldsymbol{x}_i &\leq -1 + \varepsilon_i \quad \text{for } y_i = -1 \end{aligned} \tag{67}$$

The cost function to optimize then becomes

$$J = \|\boldsymbol{w}\|^2 + C\sum_i \varepsilon_i^k \tag{68}$$

such that *C* determines the contributions of the slack variables relative to the influence of $\|\mathbf{w}\|$. The power *k* is an integer hyperparameter that SVM users normally set to $k = 1$. Inclusion of the slack variables requires forming the Lagrangian to include both the new objective, Equation (68), and new constraint functions, Equation (67). As done before, setting the partial derivatives of the modified Lagrangian to zero, solving for $\mathbf{w}$ and w_0, and substituting them back into the Lagrangian yields, the general SVM as

$$\begin{aligned} \underset{max}{L(\alpha)} &= \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ s.t. \quad & 0 \leq \alpha_i \leq C \\ & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{69}$$

Note that the only change is that the constraint involving the Lagrange multipliers has an upper bound of *C*. The generalization is that setting $C = \infty$ results in the original solution. Setting C

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

closer to zero results in a slack margin that trades off classifying more data by allowing for a higher rate of misclassification.

2.3 Classifier Evaluation

Classifiers often report a set of performance parameters such as accuracy, precision, and recall (Géron 2019). This section will define the various performance metrics by using the following abbreviations:

- *TP* = True Positive
- *TN* = True Negative
- FP = False Positive (Type I error or false alarm)
- FN = False Negative (Type II error or missed detection)

2.3.1 Confusion Matrix

A **confusion matrix** is a table that summarizes the performance of the classifier. Table 1 illustrates a confusion matrix with the relationships defined between each performance metric.

Table 1: Confusion Matrix Example

n	Predicted Positive	Predicted Negative	
Actual Positive	TP	FN	TP + FN
Actual Negative	FP	TN	FP + TN
	TP + FP	FN + TN	

The **accuracy** of a classifier is

$$a = \frac{TP + TN}{n} = \frac{TP + TN}{(TP + FN) + (FP + TN)} \tag{70}$$

The **error rate** of misclassification rate is 1 - *a*. The **precision** *p* of classification for each class is a measure of its prediction accuracy for that class. Precision is the proportion of positive predictions that are correct where

$$p = \frac{TP}{TP + FP}. \tag{71}$$

Precision, therefore, provides a relative comparison of performance for each class to gauge unbalanced performers—those that predict better for one class than another. Practitioners sometimes refer to precision as **specificity**. Intuitively, once can think of specificity as the ability of a classifier to identify a *specific* class without errors. Specificity is also 1 - *FP*, which is the true negative rate.

The **recall** *r* is the proportion of the positive classes that the classifier correctly predicts where

$$r = \frac{TP}{TP + FN}. \tag{72}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

Intuitively, one can think of recall as the proportion of the true positives that a classifier can *recall* from the sample by its predictions. Therefore, practitioners sometimes refer to recall as **sensitivity**.

The **F1 score** is the harmonic mean of precision and recall such that

$$F_1 = 2 \times \frac{p \cdot r}{p + r} \tag{73}$$

Hence, the harmonic mean is a weighted average of the precision and recall where 1 is best and 0 is worst. Note that F_1 is 1 if $p = r = 1$ and zero if either p or r is zero. Rewriting the F1 score as

$$F1 = \frac{TP}{TP + \left(\frac{FP + FN}{2}\right)} \tag{74}$$

makes it clear that the score is equivalent to either a precision or a recall based on the mean of the total errors, which is (FP + FN)/2. In practice, there is always a tradeoff between sensitivity and specificity. Hence, the F1 score reflects the amount of the tradeoff involved.

2.3.2 ROC and AUC

ROC is often confused with the term region of convergence. However, it stands for Receiver Operating Characteristic in the context of evaluating classifier performance. The ROC is a graphical plot of the TP rate versus the FP rate (Type I error), as a function of a classifier's decision threshold (Fawcett 2006). The y-axis and x-axis are ranges for the TP and FP rates, respectively.

Figure 4 provides an illustration for deriving a ROC based on hypothetical probability distributions of two classes. In this scenario, a radio receiver predicts the transmission of a zero binary digit or a one binary digit based on a received feature. In this scenario, the feature is a voltage level derived from the antenna. The plot spans the feature value on the x-axis and the probability of reception for each class on the y-axis. The feature threshold *T* is set such that the classifier, which is a binary receiver in this scenario, predicts the transmission as a zero and a one digit for voltages received below and above *T*, respectively.

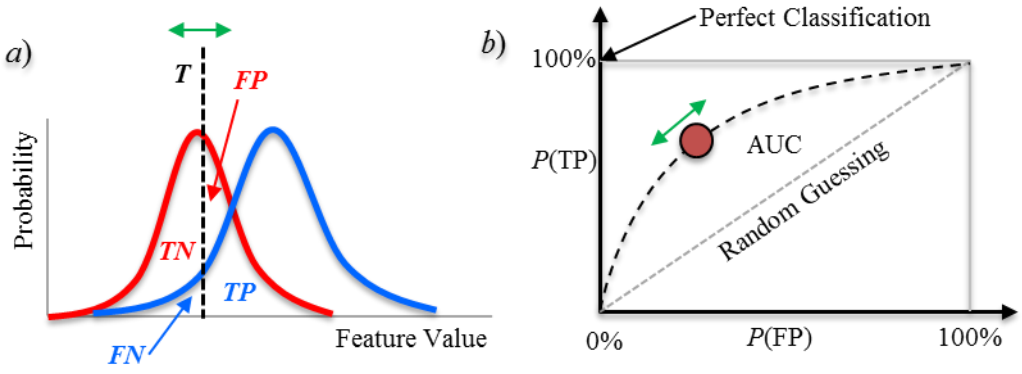

Figure 4: Determination of the ROC curve for a classifier.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

The amount of overlap between the two distributions is proportional to the amount of noise in the transmission channel, which causes confusion about the binary digits received. That is, some of the voltages representing zeros that exceed the threshold will cause the classifier to predict them as ones instead, so those would be FP predictions. Similarly, some of the voltages representing ones that fall below the threshold would cause the classifier to predict them as zeros instead, so those would be FN predictions. A low noise channel would yield distributions for zeros and ones that do not overlap, in which case the classifier can find a clear threshold to separate the classes with FP or FN errors.

The amount of overlap in the distributions affect the shape of the ROC curve. No overlap moves the curve towards the coordinate of perfect classification (0, 1). Full overlap moves the ROC curve towards the diagonal, which represents the results for random guessing. When the ROC curve falls below the diagonal, the classifier performs worse than random guessing. Given some amount of overlap in the distribution of feature values for each class (Figure 4a), moving the threshold is equivalent to moving the point along the ROC curve (Figure 4b). The goal of classifier design is to maximize the area under the ROC curve (AUC), and to move the detection threshold towards the point of perfect classification.

The tradeoff between sensitivity and specificity becomes evident by imagining moving the threshold line *T*. Increasing the threshold level by moving it towards the right will *decrease* the FP rate because the *sensitivity* will be lowered, thus reducing the probability of predicting noise as a positive. However, the higher threshold (lower sensitivity) will also *increase* the number of missed detections by not detecting true positives that fall below the threshold, thus *increasing* the FN rate. In summary, decreasing the sensitivity of the receiver will decrease the FP rate but increase the FN rate. The reverse occurs when increasing the sensitivity by lowering the threshold. That is, the FP rate will increase (decreased specificity) because the receiver may detect noise peaks as positives whereas the FN rate will decrease (increased sensitivity) because the receiver will be less likely to miss a weak signal that is a true positive.

2.3.3 Cross Validation

The goal of validating a classifier is to determine how it performs on unseen data. This is a mechanism to help prevent setting hyperparameters that cause the model to *overfit* the training data, which can lead to poor *generalization* in performance on new data. The simplest method of classifier performance validation splits the dataset into a training and a validation subset. Typically, the classifier trains on 70% of the data and uses the remaining 30% to validate its performance using one or more of the performance measures such as F1 score and AUC. The literature also refers to the two-part data split as **leave-p-out** validation. That is, the approach leaves *p* observations in a validation dataset, and uses the rest for training. Some techniques use leave-one-out validation (p = 1) for relatively small datasets.

Another technique called **cross-validation** evaluates the performance of a classifier by segmenting the dataset into several near equal parts so that the evaluation cycle uses each part for validation at least once and the union of the remaining parts to train a new model. Thereafter, average score of each model becomes the reported performance of the classifier. **K-fold** cross-validation is a generalization that partitions the data into one validation set, and *k*-1 training sets. The algorithm creates the partitions (subsets or folds) by randomly selecting data points for each subset. The user can also modify the randomization to yield stratification that balances the proportion of classes across each subset. The key advantage of k-fold cross validation is that it uses all the data for both training and validation to prevent bias towards any portion of the data. The disadvantage is increased processing time for larger datasets.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

3 Discussion

This section expands on some of the intuition behind the Lagrangian formulation, the Kernel Trick, and tuning hyperparameters. A few remarks also highlight some of the main advantages and limitations of SVMs relative to other types of machine learning classifier models.

3.1 The Lagrangian

The Lagrangian formulation is a mathematical tool that packages an optimization problem with constraints so that the problem becomes more practical to solve using a computer. The formulation comes from a realization that at each solution point in feature space, the gradient vector of the optimization function f is a scaled version of the gradient vector of the combined constraint function g such that

$$\nabla f(\boldsymbol{x}) = \lambda \nabla g(\boldsymbol{x}) \tag{75}$$

where λ is the scaling factor or the Lagrangian multiplier. Rewriting Equation (75) and assigning it to a function (the Lagrangian) gives

$$\nabla L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) - \lambda \nabla g(\boldsymbol{x}) = 0. \tag{76}$$

This Lagrangian formulation includes an extra variable λ into the solution set. Hence, solving $\nabla L(\boldsymbol{x}, \lambda) = 0$ finds all the solution points where the optimization and combined constraint function share the same tangent. At each solution point, the two gradient vectors are perpendicular to the tangent. After substituting the solutions into the original Lagrangian expression, a subsequent optimization problem finds the maximum or minimum in the solution set. Some of the advantages of the Lagrangian formulation are:

1. There exists a global minimum or maximum (none are local), which is unlike the situation for other optimization problems such as using gradient descent to train artificial neural networks. Therefore, the solution set for a given training dataset is global and not local.
2. There are relatively few support vectors that define the class decision boundary, which reduces the computational complexity over other methods that use all the training data.
3. During training, the dot product of the feature vectors can facilitate rapid computation by leveraging the efficient multiply-accumulate (MAC) operation of digital signal processors.
4. The dot product reduces complexity by facilitating the direct substitution of any Kernel function that can generalize the optimization problem of finding either linear or non-linear hyperplanes in the ambient space.

3.2 The Kernel Trick

As demonstrated earlier, a Kernel achieves the same effect as computing a dot product of the vectors transformed to a higher dimensional feature space by performing a dot product, along with some other manipulations, only in the ambient space. The type of Kernel selected determines the nature of the higher dimension space and its effectiveness in separating the classes so that an optimal bisecting hyperplane becomes possible. Given that a dot product is a measure of similarity between feature vectors, a Kernel retains the *relative* relationship among the feature vectors in the higher dimension space. Hence, feature clusters in ambient space will

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 January 2022 doi:10.20944/preprints202201.0232.v1

similarly cluster in the higher dimension feature space. Data objects that are more alike tend to cluster together and dissimilar data objects tend to be further apart from each other.

Figure 5 illustrates the dot product of two vectors ***x*** and ***y*** to highlight their relationship based on the mathematical definition. Vectors that point in the same direction are maximally similar because the angle between them is zero degrees and cos(0) = 1. Therefore, the measure of similarity for parallel vectors is just the product of their lengths. In contrast, perpendicular vectors will be dissimilar regardless of their lengths because the angle between them is 90 degrees and cos(90) = 0.

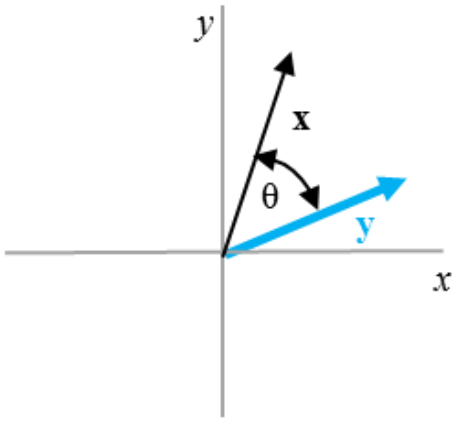

Figure 5: Dot product between two vectors.

In summary, although a Kernel function adds the extra dimensions of feature space to enable the SVM to separate the classes with a linear hyperplane, the Kernel preserves the relative within class and across class relationships in the ambient space.

3.3 Hyperparameter Tuning

As noted, the value of the slack hyperparameter *C* can range from zero to infinity and accommodates the tradeoff of classifying more data by allowing for a higher rate of misclassification. Other hyperparameters include the type of Kernel function and their parameters. Setting the best value for the hyperparameters requires some intuition about the nature of the data, some domain knowledge about the processes that generated the data, and some idea about the relationship among the vectors in feature space. Unsupervised machine learning methods such as principle component analysis (PCA) (Jolliffe and Cadima 2016), k-means clustering, and non-linear dimensionality reduction (Becht, et al. 2019) can help to visualize relationships in a lower dimension space such as 2D or 3D. Such visualizations can reveal how sparse, clustered, noisy, or separable the data might be.

A blind approach to finding the best hyperparameters for a classifier is to conduct several cross-validation cycles and plot the performance trends with different hyperparameter settings (Bridgelall and Tolliver 2021). Of course, doing so also requires selecting the best cross-validation type, the cross-validation parameters, the performance scores to monitor, and the number cycles to average those scores.

3.4 Limitations of SVM

Some advantages of SVM classifiers are:

1. They can more efficiently accommodate problems with feature dimensions that are larger than the number of data samples (Géron 2019).
2. They use only a few samples of the training data to form the decision boundary for classification. So unlike probabilistic classifiers that use all the training data, they require fewer computational resources for both training and classification.
3. The relatively small number of support vectors reduce the tendency for overfitting.
4. The flexibility to evaluate many different Kernel functions within a cross-validation cycle increase the likelihood that SVMs can find optimal hyperplanes that provide good generalized performance.

Despite their advantages above, there are several SVM limitations. When the data is noisy, such as containing many erroneous, missing, or irrelevant values, it can become more difficult to identify a hyperplane that is void of data objects. By contrast, algorithms that create a voting ensemble of diverse classifiers may perform better because the randomization that they incorporate may improve their chance of finding better locations in feature space that can accommodate non-linear decision boundaries.

Other types of classifiers natively produce a class membership probability as part of their classification decision. A distribution of this probability for each class can become the basis for generating the ROC curve by varying the class decision threshold to compute the TP and FP rates. One way around this deficiency is to use Platt scaling, which applies logistic regression to the SVM scores for estimating the probabilities (Lin, Lin and Weng 2007). However, the Platt scaling can be a computationally expensive for large datasets, and the probability estimates may be inconsistent with the scoring.

4 Conclusions

The aim of this tutorial on support vector machines (SVMs) was to help beginning students of data science to grasp the key concepts behind the theory and to gain some intuition on the meaning of the hyperparameters that require manual selection in practice. The tutorial begins by introducing the main concept of using a linear hyperplane with a maximum margin to separate the training data and to predict the classes of new data. Next, the method highlighted how to recast the margin optimization problem from its primal quadratic form to an equivalent Lagrangian dual form that is more computationally efficient.

A description of the Kernel Trick highlighted how it achieves the same effect of implementing a linear decision boundary in a higher dimension space by operating on the feature vectors in their ambient space. Hence, for non-linearly separable data, the Kernel Trick creates a non-linear decision boundary in the ambient feature space that optimally separates the classes.

The performance evaluation concepts of a confusion matric, the AUC, and cross-validation brough into perspective techniques that users can employ to select the best values for the hyperparameters. The intuitive explanations of these important concepts will help to enhance learning for beginning and seasoned students of machine learning. Future work will develop tutorials that provide an intuitive understanding of the similarities and differences among the most popular ensemble machine learning methods, including the concepts of *boosting* and *bagging*. In the meanwhile, the reference section contains a small selection of resources to help students focus on those as recommended reading.

5 Conflicts of Interest

The author declares no conflict of interest. No funding was received for this work.

References

Aggarwal, Charu C. 2015. *Data Mining.* New York, New York: Springer International Publishing.

Becht, Etienne, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel WH Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W. Newell. 2019. "Dimensionality Reduction for Visualizing Single-Cell Data using UMAP." *Nature Biotechnology* 37 (1): 38-44. doi:10.1038/nbt.4314.

Bhavsar, Hetal, and Amit Ganatra. 2012. "Variations of Support Vector Machine classification Technique: A survey." *International Journal of Advanced Computer Research* 2 (6): 230-236.

Bridgelall, Raj, and Denver D. Tolliver. 2021. "Railroad accident analysis using extreme gradient boosting." *Accident Analysis & Prevention* 156: 106126-106126. doi:10.1016/J.AAP.2021.106126.

Fawcett, Tom. 2006. "An introduction to ROC analysis." *Pattern Recognition Letters* (Elsevier) 27 (8): 861-874. http://people.inf.elte.hu/kiss/11dwhdm/roc.pdf.

Géron, Aurélien. 2019. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2nd. Sebastopol, California: O'Reilly Media.

Jolliffe, Ian T., and Jorge Cadima. 2016. "Principal Component Analysis: A Review and Recent Developments." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2065): 20150202. doi:10.1098/rsta.2015.0202.

Lin, Hsuan-Tien, Chih-Jen Lin, and Ruby C. Weng. 2007. "A note on Platt's probabilistic outputs for support vector machines." *Machine Learning* 68 (3): 267-276. doi:10.1007/S10994-007-5018-6.