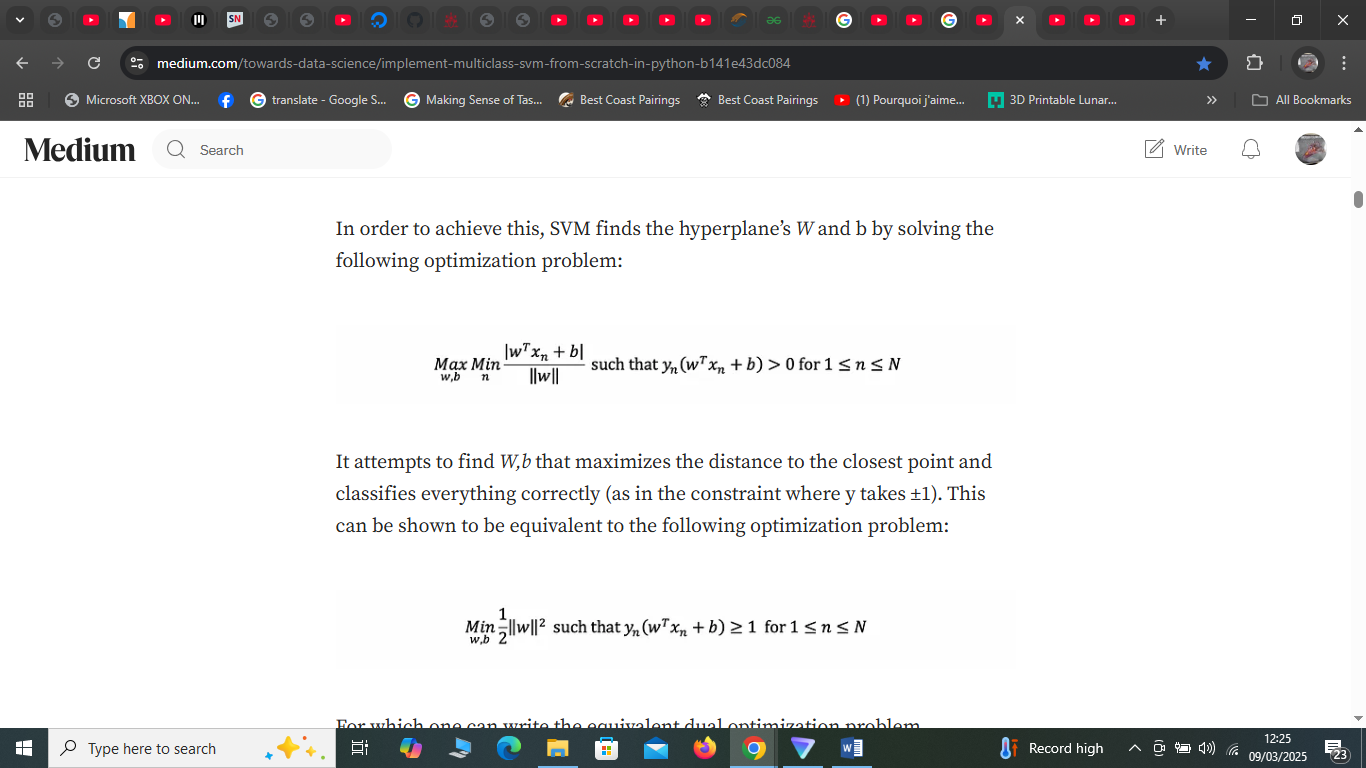
**Easy stuff**

https://medium.com/towards-data-science/implement-multiclass-svm-from-scratch-in-python-b141e43dc084

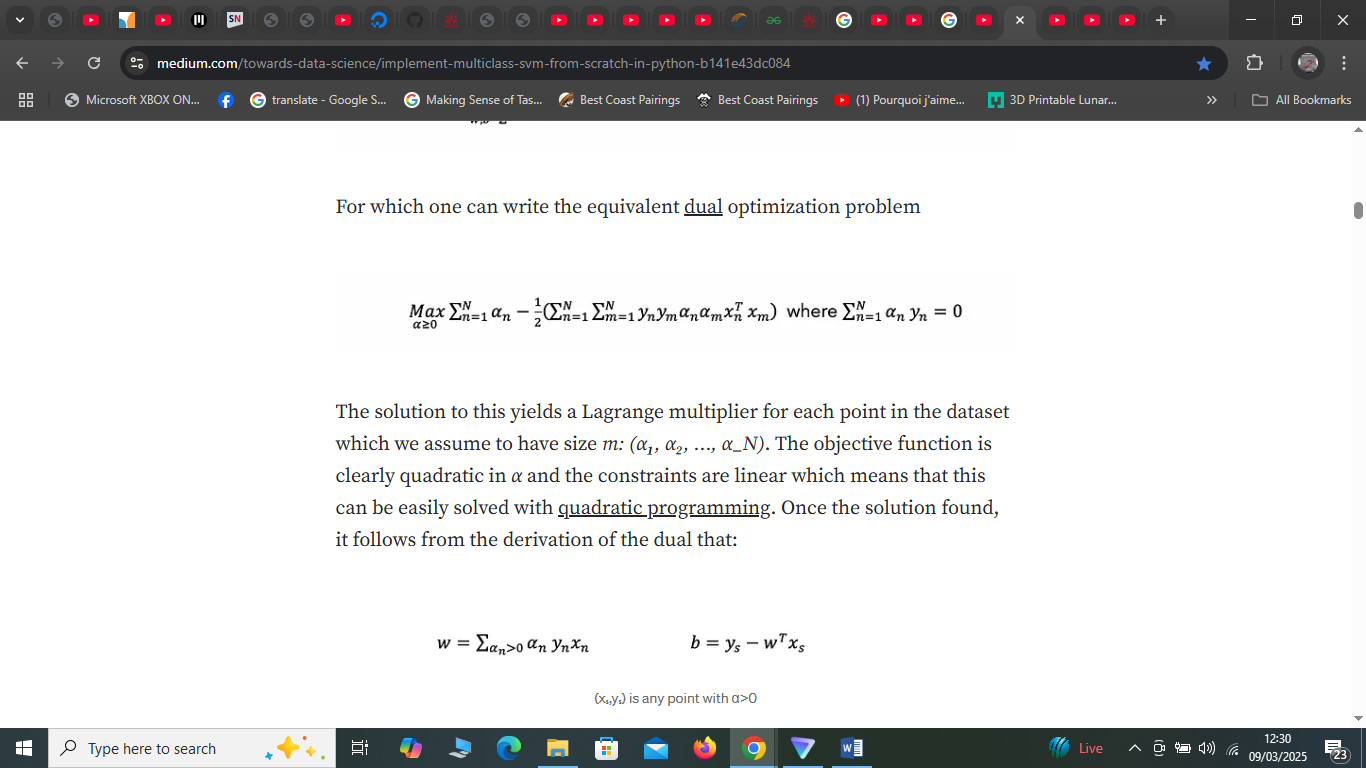
The goal in SVM is to fit the hyperplane that would attain the maximum margin (distance from the closest points from in the two classes).

In order to achieve this, SVM finds the hyperplane’s W and b by solving the following optimization problem:

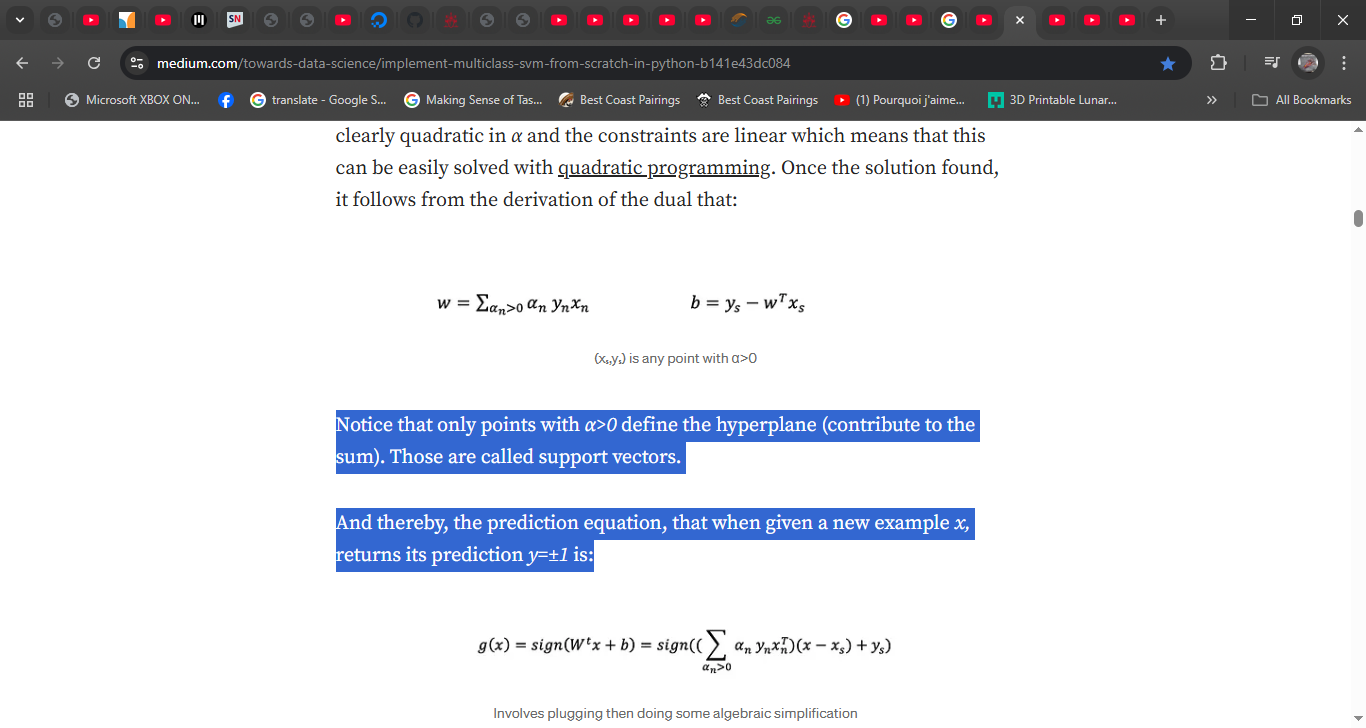
Ie the intercept and coefficient of the plane (or line)



It attempts to find W,b that maximizes the distance to the closest point and classifies everything correctly (as in the constraint where y takes ±1). This can be shown to be equivalent to the following optimization problem:

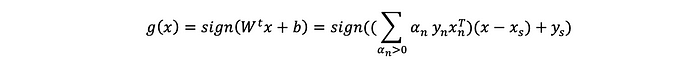


The solution to this yields a Lagrange multiplier for each point in the dataset which we assume to have size m: (α**₁**, α₂, …, α\_N). The objective function is clearly quadratic in α and the constraints are linear which means that this can be easily solved with [quadratic programming](https://en.wikipedia.org/wiki/Quadratic_programming). Once the solution found, it follows from the derivation of the dual that:



Notice that only points with *α>0*define the hyperplane (contribute to the sum). Those are called support vectors.

And thereby, the prediction equation, that when given a new example *x,*returns its prediction *y=±1* is:



soft margin SVM adapts the optimization problem by introducing a C constant (user given hyperparamer) that controls how “hard” it should be. In particular, it modifies the primal optimization problem to become the following:

https://miro.medium.com/v2/resize:fit:700/1*kdKoMPS7x3bU09TC6fErSA.png

modifications in blue

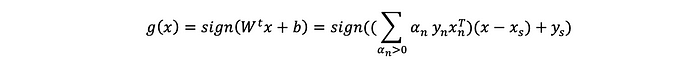
which allows each point to make some violation ϵₙ (e.g., be in the wrong side of the hyperplane) but still aims to reduce them overall by weighting their sum in the objective function by C. It becomes equivalent to hard margin as C approaches infinity (generally before it does). Meanwhile, a smaller C would allow more violations (in return for a wider margin; i.e., smaller wᵗw).

Quite surprisingly, it can be shown that the equivalent dual problem only changes by constraining *α*for each point to be *≤C.*

https://miro.medium.com/v2/resize:fit:700/1*kH5gGOgzNJ8FIlCAx87OpA.png

Since violations are allowed, support vectors (points with *α>0)*are no longer all on the margin’s edge. It can be shown that any support vector that has committed a violation will have *α=C*and that non-support vectors (*α=0)*cannot commit violations. We call support vectors that potentially committed violations (*α=C*) “non-margin support vectors” and other pure ones (that have not committed violations; lie on the edge) “margin support vectors” (0<*α<C*).

It can be shown that the inference equation doesn’t change:



However, now *(xₛ,yₛ)*must now be a support vector that has not committed violations because the equation assumes it’s on the margin’s edge (previously, any support vector could be used).

ernel Soft Margin SVM generalizes Soft Margin SVM to deal with situations where the data is naturally nonlinear.

It is possible, however, to map each point x in the dataset to a higher dimension via some transform function z=Φ(x)to make the data more linear (or perfectly linear) in the new higher dimensional space.

 reality, especially when Φ transforms into a very high-dimensional space, computing z can take a very long time. This is solved by the kernel trick which replaces the zᵗz with an **equivalent** computation of a mathematical function (called kernel function) and which is much faster (e.g., an algebraic simplification of zᵗz)

In practical scenarios, there may be some noise that prevents or limits the existence of a hyperplane that perfectly separates the data in which case the optimization problem would return no or a poor solution.

, the inference equation becomes (after algebraic manipulation):

https://miro.medium.com/v2/resize:fit:700/1*QLO83e5HuID4rODOUek_Nw.png

The SVM has three main hyperparameters, the kernel (we store the string given and the corresponding kernel function), the regularization parameter C and the kernel hyperparameter (to be passed to the kernel function); it represents Q for the polynomial kernel and γ for the RBF kernel.

Let **α**be a variable column vector(α**₁** α₂ … α\_N)ᵗ and let y be a constant column vector for the labels (y**₁** y₂ … y\_N)ᵗ and let K be a constant matrix where K[n,m] computes the kernel at (xₙ, xₘ).

We ensure that this is a binary problem and that the binary labels are set as assumed by SVM (±1) and that y is a column vector with dimensions (N,1). Then we solve the optimization problem to find (α**₁** α₂ … α\_N)ᵗ.

We use (α**₁** α₂ … α\_N)ᵗ to get an array of flags that is 1 at any index corresponding to a support vector so that we can later apply the prediction equation by only summing over support vectors and an index for a margin support vector for (xₛ,yₛ). Notice that in the checks we do assume that non-support vectors may not have α=0 exactly, if it’s α≤1e-3 then this is approximately zero (we know CVXOPT results may not be ultimately precise). Likewise, we assume that non-margin support vectors may not have α=C exactly.

To generalize the model to multiclass, over *k* classes. We train a binary SVM classifier for each class present where we loop on each class and relabel points belonging to it into +1 and points from all other classes into -1.

The result from training is *k* classifiers when given *k* classes where the *ith* classifier was trained on the data with the *ith* class being labeled as +1 and all others being labeled as -1.

**Generalizing Predict to Multiclass**

Then to perform prediction on a new example, we choose the class for which the corresponding classifier is most confident (has the highest score)

Thus a good classifier i.e. a good hyperplane will be one that will give a large positive value of *y*∗

(*wx*+

*b*)for all the points. It encourages the SVM to find a hyperplane that not only separates the two classes but also maximizes the margin between them.

Since many real practical dataset contains points that are not linealy seprable we need a plane that allows minimum misclassification of data points. For this we have Soft margin SVM. A soft margin SVM is an extension of hard marginv SVM that allows for some misclassification of data points withing the margin.

Links –

<http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>

<https://dmkothari.github.io/Machine-Learning-Projects/SVM_with_MNIST.html>

<file:///C:/Users/Probook/Downloads/guenther-schonlau-2016-support-vector-machines.pdf>

<https://github.com/christianversloot/machine-learning-articles/blob/main/creating-a-simple-binary-svm-classifier-with-python-and-scikit-learn.md>

<https://www.youtube.com/watch?v=s8q_OQBJpwU>

Lab and Thoery

<https://github.com/fastai/fastbook/blob/master/04_mnist_basics.ipynb>

A PyTorch tensor is nearly the same thing as a NumPy array, but with an additional restriction that unlocks some additional capabilities. It's the same in that it, too, is a multidimensional table of data, with all items of the same type. However, the restriction is that a tensor cannot use just any old type—it has to use a single basic numeric type for all components. For example, a PyTorch tensor cannot be jagged. It is always a regularly shaped multidimensional rectangular structure.

The vast majority of methods and operators supported by NumPy on these structures are also supported by PyTorch, but PyTorch tensors have additional capabilities. One major capability is that these structures can live on the GPU, in which case their computation will be optimized for the GPU and can run much faster (given lots of values to work on). In addition, PyTorch can automatically calculate derivatives of these operations, including combinations of operations. As you'll see, it would be impossible to do deep learning in practice without this capability.

nstead of trying to find the similarity between an image and an "ideal image," we could instead look at each individual pixel and come up with a set of weights for each one, such that the highest weights are associated with those pixels most likely to be black for a particular category. For instance, pixels toward the bottom right are not very likely to be activated for a 7, so they should have a low weight for a 7, but they are likely to be activated for an 8, so they should have a high weight for an 8. This can be represented as a function and set of weight values for each possible category—for instance the probability of being the number 8:

def pr\_eight(x,w): return (x\*w).sum()

Here we are assuming that x is the image, represented as a vector—in other words, with all of the rows stacked up end to end into a single long line. And we are assuming that the weights are a vector w. If we have this function, then we just need some way to update the weights to make them a little bit better. With such an approach, we can repeat that step a number of times, making the weights better and better, until they are as good as we can make them.

We want to find the specific values for the vector w that causes the result of our function to be high for those images that are actually 8s, and low for those images that are not. Searching for the best vector w is a way to search for the best function for recognising 8s.

1. *Initialize* the weights.
2. For each image, use these weights to *predict* whether it appears to be a 3 or a 7.
3. Based on these predictions, calculate how good the model is (its *loss*).
4. Calculate the *gradient*, which measures for each weight, how changing that weight would change the loss
5. *Step* (that is, change) all the weights based on that calculation.
6. Go back to the step 2, and *repeat* the process.
7. Iterate until you decide to *stop* the training process (for instance, because the model is good enough or you don't want to wait any longer).

Python is slow compared to many languages. Anything fast in Python, NumPy, or PyTorch is likely to be a wrapper for a compiled object written (and optimized) in another language—specifically C. In fact, **NumPy arrays and PyTorch tensors can finish computations many thousands of times faster than using pure Python.**

A NumPy array is a multidimensional table of data, with all items of the same type. Since that can be any type at all, they can even be arrays of arrays, with the innermost arrays potentially being different sizes—this is called a "jagged array.

**Classification and analysis of the MNIST dataset using PCA and SVM algorithms**

**Классификация и анализ набора данных MNIST с использованием алгоритмов PCA и SVM**

**Класификција и анализа скупа података МНИСТ помоћу алгоритама ПЦА и СВМ**

**Mokhaled N. A.** **Al-Hamadani** [a](https://www.redalyc.org/journal/6617/661774773001/html/#fn6) mokhaled\_hwj@ntu.edu.iq

University of Debrecen, Hungary

Classification and analysis of the MNIST dataset using PCA and SVM algorithms

Vojnotehnicki glasnik/Military Technical Courier, vol. 71, no. 2, pp. 221-238, 2023

University of Defence

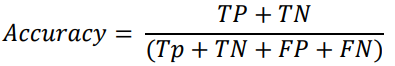
Since the SVM is a binary classification method,it is used to support classification for two classes. In our case, we used the SVM algorithm for multiclass classification. In multiclass classification, the SVM approach mainly involves splitting the dataset into several binary classification sub-datasets and then fitting a separate binary classifier for each one. There are two different types of multiclass classification for this SVM approach which are one versus one (OvO) and one versus the rest or all (OvR).

The OvO method is splitting the Multiclass dataset the data into Multiple binary classification problems. This approach splits the dataset into one dataset for each class versus every other class. The formula for calculating the number of binary datasets is: (Num\_Classes \* (Num\_Classes – 1)) / 2. For the MNIST dataset is (10 \*(10-1)/2)=45 binary classification problems.

The OvR method is also splitting the multiclass dataset into multiple binary classification problems. This approach trains a binary SVM classifier for each class, where the class is treated as a positive class and all other classes are combined into a single negative class. For example, class No (0) versus all the other classes combined as one class.

In this paper, we used the SVC (Support Vector Classification) method which implements the “one-versus-one” approach for multi-class classification.

For our MNIST dataset, we build linear and nonlinear SVM models to check the accuracy of these models. For the linear SVM model, we build it with its default hyperparameters to check the accuracy using the confusion matrix. The confusion matrix provides a decision that has been collected in training and testing which contains the actual labels with the predicted ones ([Saputra et al, 2022](https://www.redalyc.org/journal/6617/661774773001/html/" \l "redalyc_661774773001_ref12)). Accuracy can be calculated based on this equation:



where true positive (TP) refers to the data that has been correctly classified as positive, true negative (TN) refers to the data that has been correctly classified as negative while false positive (FP) refers to the data that has been incorrectly classified as positive and false negative (FN) refers to the data that has been incorrectly classified as negative.

We build the nonlinear SVM with its randomly chosen hyperparameters using the RBF kernel (Specifying the kernel type since it has several types such as “linear”, RBF, which is the radial basis function, “poly”, which is polynomial kernel function, “Sigmoid”) and it is used to take the data as an input and transform it into a required form of processing data. We used the RBF kernel because it is a commonly used kernel function due to its ability to perform well in classification tasks, as well as due to its flexibility since it does not require prior information about the dataset to be used effectively. When C=1, it is the regularization parameter which controls the trade-off between maximizing the margin and minimizing the training error in the model. When the value of the regularization parameter C is large, the model prioritizes correctly classifying all training points over having a larger margin. On the other hand, when C is small, the model will prioritize having a larger margin, even if it leads to misclassifying more training points

**Fucking hard shit!**

**Sources are hyperlinks - important points are highlighted and very important points are underlined.**

<https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf>

However, there is a class of pattern recognition techniques, in which the training data points, or a subset of them, are kept and used also during the prediction phase. Section 2.5.1 For instance, the Parzen probability density model comprised a linear combination of ‘kernel’ functions each one centred on one of the training data points - f memory-based methods - fast at training - slow at predictions

The kernel concept was introduced into the field of pattern recognition by Aizerman et al. (1964) in the context of the method of potential functions, so-called because of an analogy with electrostatics. Although neglected for many years, it was re-introduced into machine learning in the context of largemargin classifiers by Boser et al. (1992) giving rise to the technique of support Chapter 7 vector machines. Since then, there has been considerable interest in this topic

The simplest example of a kernel function is obtained by considering the identity mapping for the feature space in (6.1) so that φ(x) = x, in which case k(x, x ) = xTx . We shall refer to this as the linear kernel. The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the kernel trick, also known as kernel substitution. The general idea is that, if we have an algorithm formulated in such a way that the input vector x enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel.

A further specialization involves homogeneous kernels, also known as raSection 6.3 dial basis functions, which depend only on the magnitude of the distance (typically Euclidean) between the arguments so that k(x, x ) = k(x − x )

and is often called a ‘Gaussian’ kernel. Note, however, that in this context it is not interpreted as a probability density, and hence the normalization coefficient is 6.2. Constructing Kernels 297 omitted. We can see that this is a valid kernel by expanding the square x − x 2 = xTx + (x ) Tx − 2xTx (6.24) to give

 k(x, x ) = exp −xTx/2σ2 exp xTx /σ2 exp −(x ) Tx /2σ2

f radial basis functions, which have the property that each basis function depends only on the radial distance (typically Euclidean) from a centre µj , so that φj (x) = h(x − µj).

Historically, radial basis functions were introduced for the purpose of exact function interpolation (Powell, 1987). Given a set of input vectors {x1,..., xN } along with corresponding target values {t1,...,tN }, the goal is to find a smooth function f(x) that fits every target value exactly

Expansions in radial basis functions also arise from regularization theory (Poggio and Girosi, 1990; Bishop, 1995a). For a sum-of-squares error function with a regularizer defined in terms of a differential operator, the optimal solution is given by an expansion in the Green’s functions of the operator (which are analogous to the eigenvectors of a discrete matrix), again with one basis function centred on each data 300 6. KERNEL METHODS point. If the differential operator is isotropic then the Green’s functions depend only on the radial distance from the corresponding data point. Due to the presence of the regularizer, the solution no longer interpolates the training data exactl

e noise on the input variable x is described by a variable ξ

The effect of such normalization is shown in Figure 6.2. Normalization is sometimes used in practice as it avoids having regions of input space where all of the basis functions take small values, which would necessarily lead to predictions in such regions that are either small or controlled purely by the bias parameter.

The training data set comprises N input vectors x1,..., xN , with corresponding target values t1,...,tN where tn ∈ {−1, 1}, and new data points x are classified according to the sign of y(x).

the training data set exactly, then we should try to find the one that will give the smallest generalization error

**The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles**

**The intuition behind this result is that as σ2 is reduced, the hyperplane is increasingly dominated by nearby data points relative to more distant ones. In the limit, the hyperplane becomes independent of data points that are not support vectors**

**leads to a decision boundary that lies in the middle of the region separating the data points. The large margin solution has similar behaviour**

**. Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve**

**canonical representation of the decision hyperplane**

**The optimization problem then simply requires that we maximize w−1, which is equivalent to minimizing w2, and so we have to solve the optimization problem arg min w,b 1 2 w2**

**It appears that the bias parameter b has disappeared from the optimization. However, it is determined implicitly via the constraints, because these require that changes to w be compensated by changes to b. We shall see how this works shortly. In order to solve this constrained optimization problem, we introduce Lagrange Appendix E multipliers an 0, with one multiplier an for each of the constraints in (7.5), giving the Lagrangian function**

**L(w, b, a) = 1 2 w2 − N n=1 an tn(wTφ(xn) + b) − 1**

**Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to w and b, and maximizing with respect to a. Setting the derivatives of L(w, b, a) with respect to w and b equal to zero, we obtain the following two conditions w = N n=1 antnφ(xn) (7.8) 0 = N n=1 antn.**

**Eliminating w and b from L(w, b, a) using these conditions then gives the dual representation of the maximum margin problem in which we maximize L(a) = N n=1 an − 1 2 N n=1 N m=1 anamtntmk(xn, xm) (7.10) with respect to a subject to the constraints an 0, n = 1,...,N, (7.11) N n=1 antn = 0.**

However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces. The kernel formulation also makes clear the role of the constraint that the kernel function k(x, x ) be positive definite, because this ensures that the Lagrangian function L(a) is bounded below, giving rise to a welldefined optimization problem

**In Appendix E, we show that a constrained optimization of this form satisfies the Karush-Kuhn-Tucker (KKT) conditions, which in this case require that the following three properties hold an 0 (7.14) tny(xn) − 1 0 (7.15) an {tny(xn) − 1} = 0. (7.16) Thus for every data point, either an = 0 or tny(xn)=1. Any data point for which an = 0 will not appear in the sum in (7.13) and hence plays no role in making predictions for new data points. The remaining data points are called support vectors, and because they satisfy tny(xn)=1, they correspond to points that lie on the maximum margin hyperplanes in feature space, as illustrated in Figure 7.1. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained**

For later comparison with alternative models, we can express the maximummargin classifier in terms of the minimization of an error function, with a simple quadratic regularizer, in the form N n=1 E∞(y(xn)tn − 1) + λw2 (7.19) where E∞(z) is a function that is zero if z 0 and ∞ otherwise and ensures that the constraints (7.5) are satisfied. Note that as long as the regularization parameter satisfies λ > 0, its precise value plays no role.

Although the data set is not linearly separable in the two-dimensional data space x, it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus the training data points are perfectly separated in the original data space. This example also provides a geometrical insight into the origin of sparsity in the SVM. The maximum margin hyperplane is defined by the location of the support vectors. Other data points can be moved around freely (so long as they remain outside the margin region) without changing the decision boundary, and so the solution will be independent of such data points.

r. In practice, however, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization. We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified. From (7.19) we see that in the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was classified correctly, and then optimized the model parameters to maximize the margin. We now modify this approach so that data points are allowed to be on the ‘wrong side’ of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance. To do this, we introduce slack variables, ξn 0 where n = 1,...,N, with one slack variable for each training data point (Bennett, 1992; Cortes and Vapnik, 1995). These are defined by ξn = 0 for data points that are on or inside the correct margin boundary and ξn = |tn − y(xn)| for other points. Thus a data point that is on the decision boundary y(xn)=0 will have ξn = 1, and points

in which the slack variables are constrained to satisfy ξn 0. Data points for which ξn = 0 are correctly classified and are either on the margin or on the correct side of the margin. Points for which 0 < ξn 1 lie inside the margin, but on the correct side of the decision boundary, and those data points for which ξn > 1 lie on the wrong side of the decision boundary and are misclassified, as illustrated in Figure 7.3. This is sometimes described as relaxing the hard margin constraint to give a soft margin and allows some of the training set data points to be misclassified. Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ. Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize

**C N n=1 ξn + 1 2 w2**

**where the parameter C > 0 controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has ξn > 1, it follows that n ξn is an upper bound on the number of misclassified points. The parameter C is therefore analogous to (the inverse of) a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. In the limit C → ∞, we will recover the earlier support vector machine for separable data**

e {an 0} and {µn 0} are Lagrange multipliers. The corresponding set of Appendix E KKT conditions are given by an 0 (7.23) tny(xn) − 1 + ξn 0 (7.24) an (tny(xn) − 1 + ξn)=0 (7.25) µn 0 (7.26) ξn 0 (7.27) µnξn = 0

∂L ∂w = 0 ⇒ w = N n=1 antnφ(xn) (7.29) ∂L ∂b = 0 ⇒ N n=1 antn = 0 (7.30) ∂L ∂ξn = 0 ⇒ an = C − µn. (7.31) Using these results to eliminate w, b, and {ξn} from the Lagrangian, we obtain the dual Lagrangian in the form

0 an 1/N (7.39) N n=1 antn = 0 (7.40) N n=1 an ν. (7.41) This approach has the advantage that the parameter ν, which replaces C, can be interpreted as both an upper bound on the fraction of margin errors (points for which ξn > 0 and hence which lie on the wrong side of the margin boundary and which may or may not be misclassified) and a lower bound on the fraction of support vectors. An example of the ν-SVM applied to a synthetic data set is shown in Figure 7.4. Here Gaussian kernels of the form exp (−γx − x 2) have been used, with γ = 0.45.

Although predictions for new inputs are made using only the support vectors, the training phase (i.e., the determination of the parameters a and b) makes use of the whole data set, and so it is important to have efficient algorithms for solving

By working directly in terms of the kernel function, without introducing the feature space explicitly, it might therefore seem that support vector machines somehow manage to avoid the curse of di- 336 7. SPARSE KERNEL MACHINES Section 1.4 mensionality. This is not the case, however, because there are constraints amongst the feature values that restrict the effective dimensionality of feature space

***We have already highlighted the fact that the support vector machine does not provide probabilistic outputs but instead makes classification decisions for new input vectors. Veropoulos et al. (1999) discuss modifications to the SVM to allow the trade-off between false positive and false negative errors to be controlled.***

The data used to fit the sigmoid needs to be independent of that used to train the original SVM in order to avoid severe over-fitting. This twostage approach is equivalent to assuming that the output y(x) of the support vector machine represents the log-odds of x belonging to class t = 1. Because the SVM training procedure is not specifically intended to encourage this, the SVM can give a poor approximation to the posterior probabilities (Tipping, 2001).

N n=1 ESV(yntn) + λw2

where λ = (2C)−1, and ESV(·) is the hinge error function defined by ESV(yntn) = [1 − yntn] +

. **The hinge error function, so-called because of its shape, is plotted in Figure 7.5. It can be viewed as an approximation to the misclassification error, i.e., the error function that ideally we would like to minimize, which is also shown in Figure 7.5. When we considered the logistic regression model in Section 4.3.2, we found it convenient to work with target variable t ∈ {0, 1}. For comparison with the support vector machine, we first reformulate maximum likelihood logistic regression using the target variable t ∈ {−1, 1**}. To do this, we note that p(t = 1|y) = σ(y) where y(x) is given by (7.1), and σ(y) is the logistic sigmoid function defined by (4.59). It follows that p(t = −1|y)=1 − σ(y) = σ(−y), where we have used the properties of the logistic sigmoid function, and so we can write p(t|y) = σ(yt).

Both the logistic error and the hinge loss can be viewed as continuous approximations to the misclassification error. Another continuous error function that has sometimes been used to solve classification problems is the squared error, which is again plotted in Figure 7.5. It has the property, however, of placing increasing emphasis on data points that are correctly classified but that are a long way from the decision boundary on the correct side. Such points will be strongly weighted at the expense of misclassified points, and so if the objective is to minimize the misclassification rate, then a monotonically decreasing error function would be a better choice

7.1.3 Multiclass SVMs The support vector machine is fundamentally a two-class classifier. In practice, however, we often have to tackle problems involving K > 2 classes. Various methods have therefore been proposed for combining multiple two-class SVMs in order to build a multiclass classifier. One commonly used approach (Vapnik, 1998) is to construct K separate SVMs, in which the kth model yk(x) is trained using the data from class Ck as the positive examples and the data from the remaining K − 1 classes as the negative examples. This is known as the one-versus-the-rest approach. However, in Figure 4.2 we saw that using the decisions of the individual classifiers can lead to inconsistent results in which an input is assigned to multiple classes simultaneously. This problem is sometimes addressed by making predictions for new inputs x using y(x) = maxk yk(x). (7.49) Unfortunately, this heuristic approach suffers from the problem that the different classifiers were trained on different tasks, and there is no guarantee that the realvalued quantities yk(x) for different classifiers will have appropriate scales. Another problem with the one-versus-the-rest approach is that the training sets are imbalanced. For instance, if we have ten classes each with equal numbers of training data points, then the individual classifiers are trained on data sets comprising 90% negative examples and only 10% positive examples, and the symmetry of the original problem is lost. A variant of the one-versus-the-rest scheme was proposed by Lee et al. (2001) who modify the target values so that the positive class has target +1 and the negative class has target −1/(K − 1). Weston and Watkins (1999) define a single objective function for training all K SVMs simultaneously, based on maximizing the margin from each to remaining classes. However, this can result in much slower training because, instead of solving K separate optimization problems each over N data points with an overall cost of O(KN2), a single optimization problem of size (K −1)N must be solved giving an overall cost of O(K2N2

Another approach is to train K(K −1)/2 different 2-class SVMs on all possible pairs of classes, and then to classify test points according to which class has the highest number of ‘votes’, an approach that is sometimes called one-versus-one. Again, we saw in Figure 4.2 that this can lead to ambiguities in the resulting classification. Also, for large K this approach requires significantly more training time than the one-versus-the-rest approach

A different approach to multiclass classification, based on error-correcting output codes, was developed by Dietterich and Bakiri (1995) and applied to support vector machines by Allwein et al. (2000). This can be viewed as a generalization of the voting scheme of the one-versus-one approach in which more general partitions of the classes are used to train the individual classifiers. The K classes themselves are represented as particular sets of responses from the two-class classifiers chosen, and together with a suitable decoding scheme, this gives robustness to errors and to ambiguity in the outputs of the individual classifiers. Although the application of SVMs to multiclass classification problems remains an open issue, in practice the one-versus-the-rest approach is the most widely used in spite of its ad-hoc formulation and its practical limitations.