SVMs, logistic regression and deep learning

Data Science 2 CS 209b

Javier Zazo Pavlos Protopapas

February 28, 2018

Abstract

In the previous data science course we introduced the use of support vector classifiers (SVCs) and support vector machines (SVMs). In this section we will further advance the material of these topics and relate their use to other classification techniques such as logistic regression. We will extend kernel methods to the logistic regression problem, and comment on the improvements and difficulties of this extension. Finally, we will conclude our exposition describing neural networks as a promising framework to exploit these difficulties, which will be further developed in the normal course.

1 Introduction

We will develop the problem formulation of SVMs. First, we introduce the maximal margin classifier, derive the SVCs and extend the methodology to SVMs using kernels.

The mathematical definition of a p-dimensional hyperplane is given by

$$\{x \,|\, f(x) = \beta_0 + \beta^T x = 0\}, \tag{1}$$

where $\beta = (\beta_1, \ldots, \beta_p)^T \in \mathbb{R}^p$ and $x \in \mathbb{R}^p$. A point x belongs to the hyperplane if it satisfies $f(x) = 0$.

Given N training data pairs $(x_1, y_1), \ldots, (x_N, y_N)$, with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, our goal is to develop a classifier based on this training data.

Assuming that the training data can be separated using a hyperplane, it follows that

$$y_i(\beta_0 + \beta^T x_i) \geq 0 \tag{2}$$

for every $i \in \{ 1, \ldots, N \}$. The result comes from the fact that the hyperplane divides the space into two halfspaces, and we assign the upper halfspace class $y_i = 1$ and the lower halspace class $y_i = -1$. The classification process for data point x is performed simply with $y \leftarrow \text{sign}[f(x)]$.

2 The maximal margin classifier

The maximal margin hyperplane is the separating hyperplane for which the margin is largest. It is the hyperplane that has the farthest minimum distance to the training observations. Figure 1 exemplifies this description by depicting the optimal maximum margin hyperplane in continuous line, and margins in dashed lines.

In order to maximize the margin, we can maximize the distance of the closest points to the hyperplane. The signed distance of any point x to a hyperplane is given by

$$D(x) = \frac{\beta_0 + \beta^T x}{\|\beta\|}, \tag{3}$$

where the formula can be derived from a geometric analysis. Figure 2 helps to understand this result. To determine the distance from x to the hyperplane, we calculate the projection of $x - x_0$ onto $\beta^* = \frac{\beta}{\|\beta\|}$. Point x_0 corresponds to any point in the hyperplane, satisfying $\frac{\beta^T}{\|\beta\|} x_0 = -\frac{-\beta_0}{\|\beta\|}$. Equation (3) follows from these two observations.

We can now formulate the problem and establish our goal as an optimization problem that maximizes the unsigned distance of every point to the hyperplane $|D(x_i)|$:

$$\begin{aligned} \max_{M,\beta_0,\beta} \quad & M \\ \text{s.t.} \quad & \frac{1}{\|\beta\|} y_i(\beta_0 + \beta^T x_i) \geq M, \quad \forall i \in \{ 1, \ldots, N \}. \end{aligned} \tag{4}$$

Since for any β and β_0 that satisfies $|D(x_i)| \geq M$ any positive multiple satisfies them too, we can assign $\|\beta\| = 1/M$ and transform problem (4) into

$$\begin{aligned} \min_{\beta_0,\beta} \quad & \|\beta\| \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1, \quad \forall i \in \{ 1, \ldots, N \}, \end{aligned} \tag{5}$$

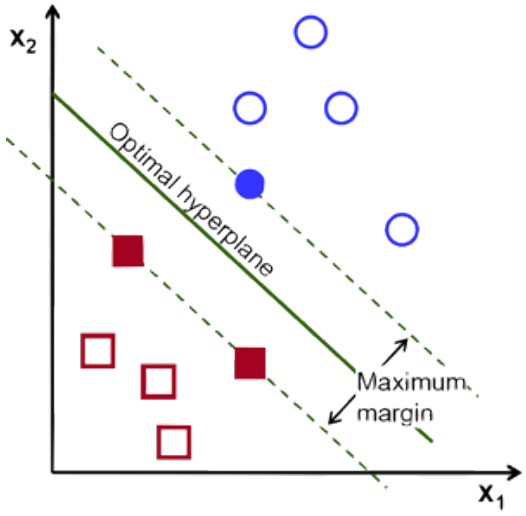

Figure 1: Separating hyperplane with maximum margin on separable data.

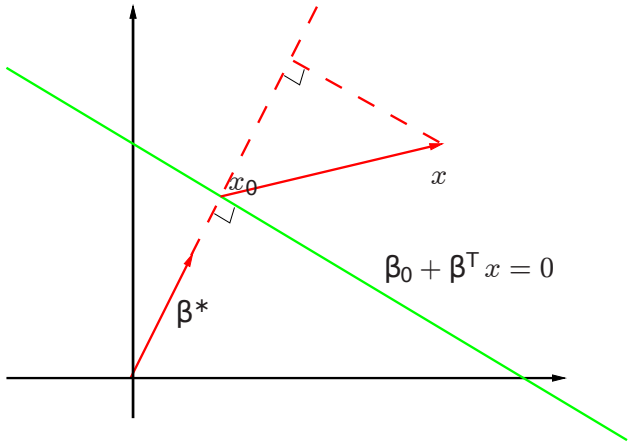

Figure 2: Graphical representation of hyperplane and related vectors.

which is a more common representation of the maximal margin classifier (sometimes also called hard margin SVC). Problems (4) and (5) are both convex, and can be solved using standard or specific solvers. We will discuss later how the dual problem is formulated, which is usually employed for the optimization task.

3 Support Vector Classifier (SVC)

Consider now the case in which the classes overlap in the feature space and there does not exist a positive margin that solves problem (4). One possibility is to relax the previous constraints and still maximize the margin while allowing some points violate the hard constraints. We can do that introducing some bounded non-negative slack variables $\xi = (\xi_1, \ldots, \xi_N)$:

$$\begin{aligned} \min_{\beta_0, \beta, \xi_i \geq 0} \quad & \|\beta\| \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i \in \{1, \ldots, N\} \\ & \sum_{i=1}^{N} \xi_i \leq C, \end{aligned} \tag{6}$$

for some positive constant C. The idea is that ξ_i represents the amount of violation that a given point x_i is allowed to exceed, while being on the wrong side of the margin. The higher the value of C, the higher amount of violation that is permitted and the margin widens. On the other hand, as C becomes smaller, the classifier becomes less tolerant of violations and the margin narrows. When $C = 0$ we recover the maximal margin classifier from Section 2. In practice, C constitutes a hyperparameter that needs to be established before the classification process. As such, it is normally chosen through cross-validation.

We are allowing two kind of 'errors' in (6): margin violation and misclassification. Margin violation refers to points that are on the correct side of the boundary but lie inside the margin. They have relative distance ξ_i to the margin boundary, with $0 < \xi_i < 1$. Misclassification points appear on the wrong side of the hyperplane. In this case they have relative distance $\xi_i > 1$. Figure 3 shows these point violations in a *soft margin classifier*.

Parameter C controls the bias-variance trade-off of the learning technique. When C has a small value, few points violate the margins and the distance of the margins to the hyperplane becomes small. Such classifier would try to fit

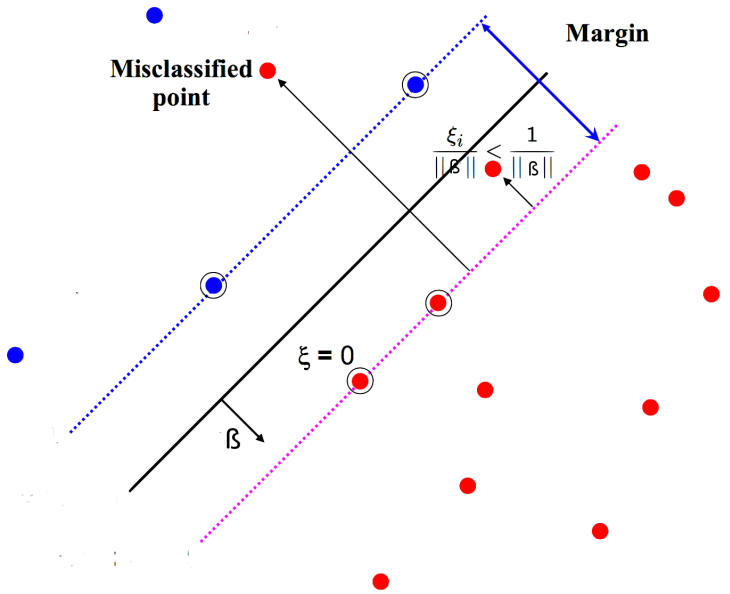

Figure 3: Soft margin classifier representation.

the data strongly, and would present low bias but high variance. On the other hand, when C is larger, the optimization problem allows more violations of points on the wrong side of the margin, and the distance between the margins becomes wider. This amounts to fitting the data less strongly and obtaining a classifier that is potentially more biased but may have lower variance.

A more convenient form of an SVC is obtained when using the constraint $\sum_i \xi_i \leq C$ as a penalization term:

$$\begin{aligned} \min_{\beta_0,\beta,\xi_i \geq 0} \quad & \frac{1}{2}\|\beta\|^2 + \lambda \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(\beta_0 + \beta^T x_i) \geq 1 - \xi_i, \quad \forall i \in \{1, \ldots, N\}. \end{aligned} \tag{7}$$

In this case λ is regarded as an intensity parameter, which affects the kind of violations allowed in the constraints. There is an implicit relation between C and λ through duality analysis, but a direct transformation between them is not straightforward. In practice it is usually easier to tune λ rather than C, although in terms of relative violations C is easier to interpret. For this reason, SVCs are normally presented in the form of (7).

Finally, small λ penalizes errors less and hence the classifier will have a large margin. A large λ penalizes errors more and then the classifier will accept narrow margins to improve classification. Setting $\lambda = \infty$ produces the hard margin solution. As we can see, this paremeter controls the bias-variance trade-off of the classifier.

3.1 The dual problem of the SVC

Studying the dual problem of a constrained optimization problem has several benefits. The more immediate one is that duality theory allows to determine if a candidate solution is optimal (under convexity and strong duality assumptions). Secondly, it also incorporates a set tools to find such optimal solutions. Additionally, occasionally, the algorithms based on the dual problem may be more efficient than those on the primal problem. Regarding SVCs, the sequential minimization optimization algorithm described in [4] is an example of an efficient algorithm. And finally, the dual problem may also include some relevant theoretical understanding of the original problem. In the case of SVCs, they help to visualize which constraints are active whenever their corresponding dual variable $\alpha_i \neq 0$.

We will now derive the dual problem of the SVC. If you are not familiar with constrained optimization techniques, please have a look at the "Intro to optimization" document that will introduce you into this matter.

In order to formulate the dual problem we first construct the Lagrangian:

$$L(\beta, \beta_0, \xi, \alpha, \mu) = \frac{1}{2}\|\beta\|^2 + \lambda \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\beta_0 + \beta^T x) - (1 - \xi_i))] - \sum_{i=1}^{N} \mu_i \xi_i. \tag{8}$$

To obtain the dual function, we need to minimize the Lagrangian with respect to the primal variables β, β_0 and ξ. Setting the respective partial derivatives to zero we obtain:

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{9a}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \tag{9b}$$

$$\alpha_i = \lambda - \mu_i, \quad \forall i. \tag{9c}$$

We also need to impose complementarity slackness and feasibility conditions:

$$\alpha_i[y_i(\beta_0 + \beta^T x) - (1 - \xi_i))] = 0 \tag{10a}$$
$$\mu_i \xi_i = 0 \tag{10b}$$
$$y_i(\beta_0 + \beta^T x) - (1 - \xi_i)) \geq 0. \tag{10c}$$
$$\alpha_i \geq 0, \quad \mu_i \geq 0, \quad \xi_i \geq 0 \quad \forall i \tag{10d}$$

Partial derivatives equal to zero and requirements derived in (10) constitute necessary and sufficient conditions for optimality.

Substituting relations from (9) into (8) we obtain the dual problem:

$$\begin{aligned} \max_{0 \leq \alpha_i \leq \lambda} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{11}$$

Solving (11) we can obtain the optimal α_i. Using equation (9a) we recover β. From (9c) we recover μ_i and if $\mu_i \neq 0$ we obtain a support vector i such that $\xi_i = 0$ because of (10b). We can then use (10a) to obtain β_0. Finally, we perform classification of a point x with $y \leftarrow \text{sign}[\beta_0 + \beta^T x]$.

We call support vectors all points for which α_i is non-zero, and they constitute the set of points which are within the wrong side of the margin or even the wrong side of the hyperplane. From an optimization point of view $\alpha_i \neq 0$ indicates that the constraint is active, and that the point is affecting the result. Whenever $\alpha_i = 0$, the constraint is inactive, and such point is not affecting the optimization process.

4 Support Vector Machines (SVMs)

Given a training set $\{ (x_1, y_1), \ldots, (x_N, y_N) \}$ with a single real-valued predictor, we can view fitting a 2nd degree polynomial model $\beta_0 + \beta_1 x + \beta_2 x^2$ on the data as the process of finding the best quadratic curve that fits the data. But in practice, we first expand the feature dimension of the training set

$$x_i \mapsto (x_i^0, x_i^1, x_i^2, x_i^3) \tag{12}$$

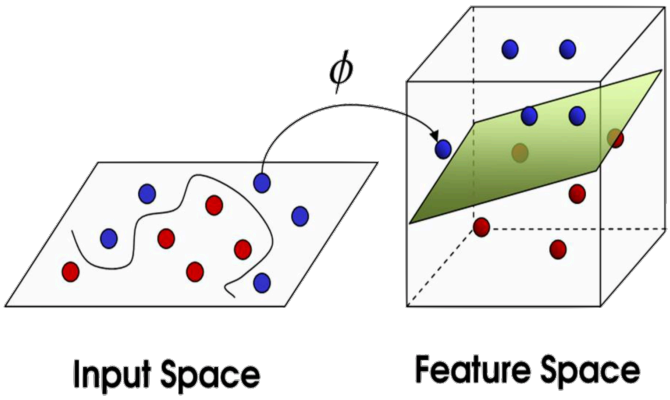

Figure 4: Transformation of input space into feature space.

and train a linear model on the expanded data

$$\{ (x_0^0, x_0^1, x_0^2, x_0^3, y_1), \ldots, (x_N^0, x_N^1, x_N^2, x_N^3, y_N) \} \tag{13}$$

The key observation is that training a polynomial model is just training a linear model on data with transformed predictors. In our previous example, transforming the data to fit a 3rd degree polynomial model requires a map

$$\begin{aligned} &\phi : \mathbb{R} \to \mathbb{R}^4 \\ &\phi(x) = (x^0, x^1, x^2, x^3) \end{aligned} \tag{14}$$

where $\mathbb{R}$ is called the input space, and $\mathbb{R}^4$ is called the feature space. Feature spaces can be used to compare objects which have much more complex structure., generalizing the polynomial regression we just described.

The same insight applies to classification: while the data may not be linear separable in the input space, it may be in a feature space after a fancy transformation, as in Figure 4. Going into higher dimensions has the motivation that we will hopefully be able to separate classes linearly in such domain.

We can have a concrete example of how this works in practice. Consider the XOR example, where we have variables from $\mathbb{R}^2$ arranged in a XOR pattern and would like to classify them using a separating hyperplane. This is clearly not possible in the original space, but through the following transformation

$$\phi(x) = (x_1, x_2, x_1 x_2), \tag{15}$$

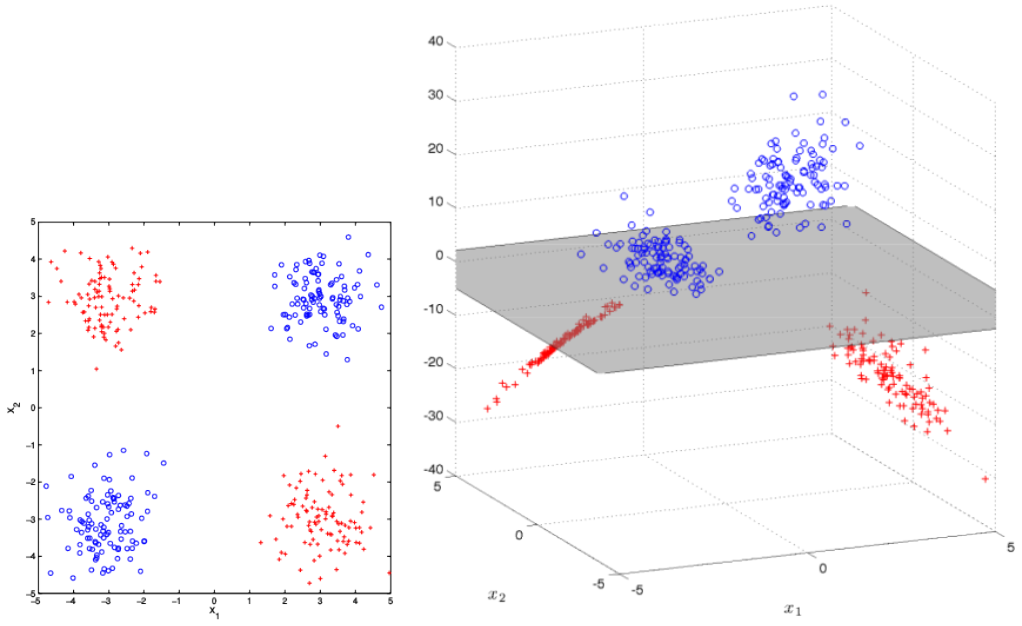

Figure 5: Transformation of input space into feature space.

we can map the input space into a feature space and perform classification in the transformed space. Figure 5 shows the result.

As we have seen, whenever we want to apply non-linear boundaries, a possible solution is the following: map the data into a feature space in which the classes are linearly separable, and then train an SVC in the new feature space. This approach has two main drawbacks. The first one is that coming up with a separable feature space can be difficult, and the second one, if the feature space is high dimensional computing ϕ can be costly. Instead, we note that computing ϕ is unnecessary. If we check on the dual problem given in (11), we observe that we are only interested in computing the inner products $\phi(x_i)^T\phi(x_i)$ in the feature space, and not the quantities $\phi(x_i)$ themselves.

The inner product between two vectors is a measure of the similarity of the two vectors. We have the following definition of a kernel:

Definition 1. *Given a transformation* $\phi : \mathbb{R}^J \to \mathbb{R}^{J'}$*, from input space* $\mathbb{R}^J$ *to feature space* $\mathbb{R}^{J'}$*, the function* $K : \mathbb{R}^J \times \mathbb{R}^{J'} \to \mathbb{R}$ *defined by*

$$K(x_i, x_j) = \phi(x_i)^T\phi(x_j), \quad x_i, x_j \in \mathbb{R}^J \tag{16}$$

is called the kernel function of ϕ*.*

Generally, a kernel function may refer to any positive function $K : \mathbb{R}^J \times \mathbb{R}^{J'} \to \mathbb{R}$ that measures the similarity of vectors in $\mathbb{R}^J$, without explicitly designing a transformation ϕ.

For a choice of kernel K, we train an SVC by solving

$$\begin{aligned} \max_{0 \le \alpha_i \le \lambda} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0, \end{aligned} \tag{17}$$

which is taken from the dual of the SVC with expanded feature maps.

Computing $K(x_i, x_j)$ can be done without computing the mappings $\phi(x_i)$, $\phi(x_j)$. This way of training a SVC in feature space without explicitly working with the mapping ϕ is called the kernel trick. The extension of SVCs using kernels as in (17) is called a *support vector machine* (SVM).

Let's take an example: define $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ by

$$\phi([x_1, x_2]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2) \tag{18}$$

The inner product in the feature space is $\phi([x_{11}, x_{12}])^\top \phi([x_{21}, x_{22}]) = (1 + x_{11}x_{21} + x_{12}x_{22})^2$. Thus, we can directly define a kernel function $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ by

$$K(x_1, x_2) = (1 + x_{11}x_{21} + x_{12}x_{22})^2. \tag{19}$$

Notice that we need not compute $\phi([x_{11}, x_{12}])$, $\phi([x_{21}, x_{22}])$ to determine $K(x_1, x_2)$. Using a kernel implicitly uses a feature mapping that we do not necessarily need to know nor calculate. In fact, kernel methods can (and often do) use infinitely many features, whenever the kernel originates from an infinite dimensional Hilbert space. As long as the learning algorithm is defined using dot products between features, we can use this kind of representation. The term "kernel" simply refers to a dot product between features.

These are some common kernels used in the literature:

- **Polynomial Kernel**

$$K(x_1, x_2) = (x_1^\top x_2 + 1)^d \tag{20}$$

  where d is a hyperparameter.

- **Radial Basis Function Kernel**

$$K(x_1, x_2) = \exp\left\{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right\} \tag{21}$$

  where σ is a hyperparameter.

- **Sigmoid Kernel**

$$K(x_1, x_2) = \tanh(\kappa x_1^\top x_2 + \theta) \tag{22}$$

  where κ and θ are hyperparameters.

5 Relationship to Logistic Regression

SVMs introduced a novel approach on how to classify data using a separating hyperplane, in combination with the powerful use of kernels to extend the feature space to accommodate non-linear class boundaries. In that moment this appeared to be a unique and valuable characteristic of SVMs, but this is not the case. Kernel methods can be extended to any process in which inner products between features arise, and further connections between SVMs and other procedures have been established as well.

We want to present SVMs with an alternative motivation based on the classical view

$$\min_{\beta} \quad J(X, y, \beta) + \lambda P(\beta), \tag{23}$$

where $J(X, y, \beta)$ corresponds to a general loss function and $P(\beta)$ to a regularizer or penalization function on the values of β. Common functions of J correspond to the least squares error function, or to the logistic loss (binary cross-entropy in some contexts). Penalization functions P correspond to $\|\beta\|^2$ in the Ridge case, or to $\|\beta\|_1$ in the LASSO example.

We can transform our SVC formulation given in (7) to

$$\min_{\beta,\beta_0} \quad \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \beta^T x_i)] + \lambda\|\beta\|^2, \tag{24}$$

where we can figure out a straightforward correspondence to (23) for loss function J and regularizer P. Function $J_i = \max[0, 1 - y_i(\beta_0 + \beta^T x_i)]$ is

sometimes called *hinge* loss function. The transformation is easy:

$$y_i(\beta_0 + \beta^T x) = 1 - \xi_i \;\rightarrow\; \xi_i = 1 - y_i(\beta_0 + \beta^T x) \geq 0 \tag{25a}$$

$$\rightarrow\; \xi_i = \max[0, 1 - y_i(\beta_0 + \beta^T x)]. \tag{25b}$$

We can substitute (25b) in the objective of (7) and obtain (24).

Finally, we want that (24) may incorporate kernels. The first modification is to use extended feature vectors $\phi(x)$:

$$\min_{\beta,\beta_0} \quad \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \beta^T \phi(x_i))] + \lambda \|\beta\|^2.$$

Then, we can substitute β derived from the KKT condition (9a), i.e., $\beta = \sum_{j=1}^{N} y_j \alpha_j \phi(x_j)$, obtaining:

$$\min_{\beta,\beta_0} \quad \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \sum_{j=1}^{N} y_j \alpha_j \phi(x_i)^T \phi(x_j))] + \lambda \|\beta\|^2. \tag{26}$$

Finally, we only need to substitute the inner product of feature maps with an appropriate kernel:

$$\min_{\beta,\beta_0} \quad \sum_{i=1}^{N} \max[0, 1 - y_i(\beta_0 + \sum_{j=1}^{N} y_j \alpha_j K(x_i, x_j))] + \lambda \|\mathbf{K}^{1/2} \alpha\|^2.$$

This generalizes the SVC to incorporate kernel methods. The loss function becomes $\max[0, 1 - yf(x)]$ with $f(x) = \beta_0 + \sum_{j=1}^{N} \alpha_j K(x, x_j)$.

5.1 Review and comparison with logistic regression

The logistic regression problem consists of determining the class to which some observable belongs to. Given training data $X = (x_1, \ldots, x_N), y = (y_1, \ldots, y_N)$ with $x_i \in \mathbb{R}^p$, $y_i \in \{1, 0\}$, we can assign a probability based on the logistic function to each data point for belonging to a specific class:

$$p = P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}}, \tag{27}$$

where $f(x) = \beta_0 + \beta^T x$ and $P(y = 0|x) = 1 - P(y = 1|x)$.

For binary classification we use a Bernoulli random variable with probability mass function given by:

$$J_l(X, y, \beta) = \prod_{i=1}^{N} P(y = y_i|x) = \prod_{i=1}^{N} p^{y_i}(1-p)^{1-y_i} \tag{28}$$

where p is taken from (27). To find the parameters that *best* classify the training data, we consider a maximum likelihood estimator. To simplify the process we maximize the log-likelihood:

$$\begin{aligned} &\max_{\beta_0,\beta} \quad \log(J_l(X, y, \beta)) \Longleftrightarrow \\ &\min_{\beta_0,\beta} \quad -\sum_{i=1}^{N} \left[y_i \log(1 + e^{-(\beta_0+\beta^T x_i)}) + (1-y_i) \log(1 + e^{(\beta_0+\beta^T x_i)} \right], \end{aligned} \tag{29}$$

where (29) is sometimes called *binary cross-entropy*.

If we depict the *binary cross-entropy* function $\log(1 + e^{y_i(\beta_0+\beta^T x)})$ vs. the *hinge loss* function $\max[0, 1 - y_i(\beta_0 + \beta^T x)]$ we obtain figure Figure 6. This figure strongly suggests that SVCs and logistic regression are strongly related, as there is a very slight difference in penalization values for both problems. Due to the similarities between their loss functions, logistic regression and the support vector classifier often give very similar results. When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred.

We have discussed the relation that exists between SVCs and logistic regression, but we have not considered the use of kernels within logistic regression. The extension is obtained using the following classification criteria:

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i K(x, x_i), \tag{30}$$

where $f(x)$ has the same form as in the SVM classification. The previous classifier is motivated by function estimation in reproducing kernel Hilbert spaces, where the reproductive kernel property is derived.

The regularized kernel logistic regression (KLR) problem becomes:

$$\min_{\beta_0,\alpha_i} \quad -\sum_{i=1}^{N} \left[y_i \log(1 + e^{-f(x_i)}) + (1-y_i) \log(1 + e^{f(x_i)} \right] + P(\alpha), \tag{31}$$

where $P(\alpha)$ constitutes a regularization term.

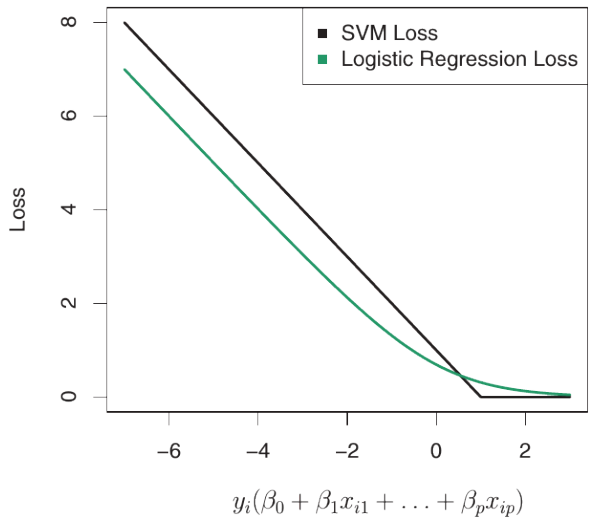

Figure 6: Hinge loss function vs logistic regression penalty function.

5.2 Advantages and disadvantages of SVMs vs KLR

- Classification performance is very similar.
- KLR provides estimates of class probabilities.
- KLR generalizes naturally to M-class classification, using a multiclass cross-entropy classifier.
- KLR converges to the maximum margin classifier as $\lambda \to 0$.
- KLR is computationally more expensive, in the order of $O(N^3)$ vs. $O(N^2 m)$, where m is the number of support points. In noisy problems, m can be large, approx $N/2$.
- In SVMs many α_i are actually zero. This allows some data compression and faster lookup. In KLR all α_i are typically non-zero.

6 Introduction to neural networks (NN)

One of the main open problems in statistical learning is the curse of dimensionality, i.e., the fact that learning becomes more difficult in higher dimensions because the training data do not fill the space densely and it is hard

to generalize and learn intricate relations. Kernel methods within SVMs or logistic regression may have a hard time to solve this problem. The reason is that when we calculate the α coefficients that multiply the kernel, all features generated by the kernel are weighted equally. This makes the task of learning a specific subspace within a larger space impossible with most used kernels, and as a consequence the performance of these methods degrade on higher dimensions.

It turns out that deep neural networks have shown remarkable capacity to obtain statistical knowledge in high dimensional spaces. In fact, NN will be trained to uncover a feature space that hopefully allows to separate the classes with a linear classifier. This uncovered space will be high dimensional, which we have seen is a beneficial characteristic for linear separability. Notice however, that for every mapping we can come up with (and has finite length), there is actually a straightforward kernel $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ that will be able to perform in such feature map.

For the rest of this course, we will focus on the techniques involved in NN that disentangle these mappings... Connecting to what you have learned in this lecture, we present a shallow neural network (of only 1 layer) reproducing a logistic regression.

As before, we have N training data pairs $(x_1, y_1), \ldots, (x_N, y_N)$, with $x_i \in \mathbb{R}^p$ and $y_i \in \{1, 0\}$. Values $x \in \mathbb{R}^p$ constitute the input layer to the NN, and we will use a single sigmoid function as the output layer of the NN. In this case, the parameters of the neural network are the β weights at the input and β_0 offset. We forward this result to the sigmoid function and obtain our estimate. Figure 7 depicts these interactions where $\sigma(z) = 1/(1 + \exp(-z))$. In the next lecture we will learn how to train β and β_0 using the back-propagation algorithm. Logistic regression does not have any hidden layers, as it only has a single output layer. Note that in general the linear computation and the activation function will be represented by a single neuron, but here we wanted to emphasize both computations separately.

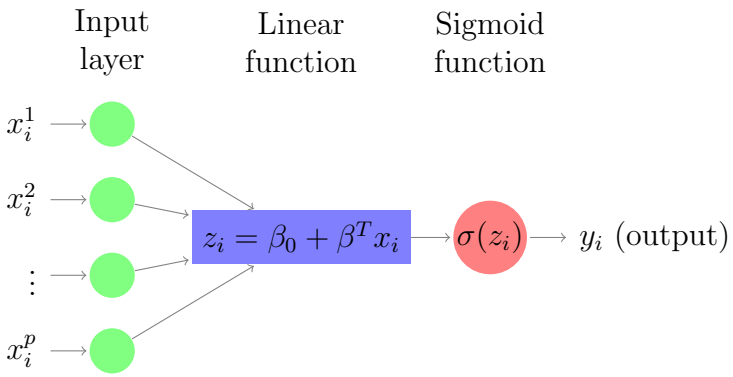

Figure 7: Neural network representing a logistic regression problem.

References

[1] A. Gretton. Introduction to RKHS, and some simple kernel algorithms. Technical report, University College London, 2017.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer Series in Statistics. Springer New York, 2009.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R.* Springer Texts in Statistics. Springer New York, 2014.

[4] Andrew Ng. Support vector machines. Technical report, Stanford, 2017.

Acknowledgments

Figure 5 has been borrowed from [1]. Figures 1, 2 and 6 from [3]. Figures 3 and 4 from an unknown source.