Machine Learning - Michaelmas Term 2016
Lectures 9, 10 : Support Vector Machines and Kernel Methods

Lecturer: Varun Kanade

In the previous lecture, we studied logistic regression, a discriminative model for classification. In this lecture, we'll study another model for classification problems, known as *support vector machines*. Unlike logistic regression, there is no natural probabilistic interpretation of this model; however, it is possible to construct one, if one really wishes.[1]

When studying logistic regression, we observed that the discriminating boundary (in the case of binary classification problems) was linear. This will be our starting point when studying support vector machines, or SVMs, for short. We'll explicitly try to separate the data using a linear separator and introduce the *maximum-margin* principle, which is one way to characterise optimal linear separators.

1 Support Vector Machines: Primal Formulation

In this lecture, we will adopt the notation that the class labels are given by $y = +1$ and $y = -1$, which we will refer to as positive examples and negative examples respectively (rather than $y \in \{0, 1\}$; recall that this is just for mathematical convenience). Let us start with a simple two-dimensional problem; suppose we are given some data that is labelled as positives and negatives. We wish to identify a linear separator, which in this case is simply a line, but in higher dimensions will be a hyperplane, that separates the positives from negatives. When the data admits such a linear separator (which in general it may not), we refer to the setting as the *linearly separable* setting. The data shown in Figure 1 is linearly separable, and in fact there are several possible linear separators Which of these should we pick?

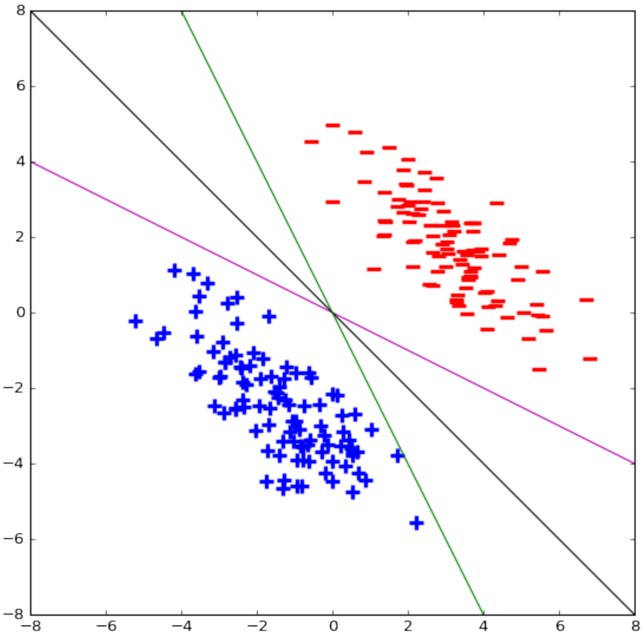


Figure 1: Positive and negative labelled data with possible linear separating boundaries.

[1] See (Murphy, 2012, Chap 14.5.5) for details.

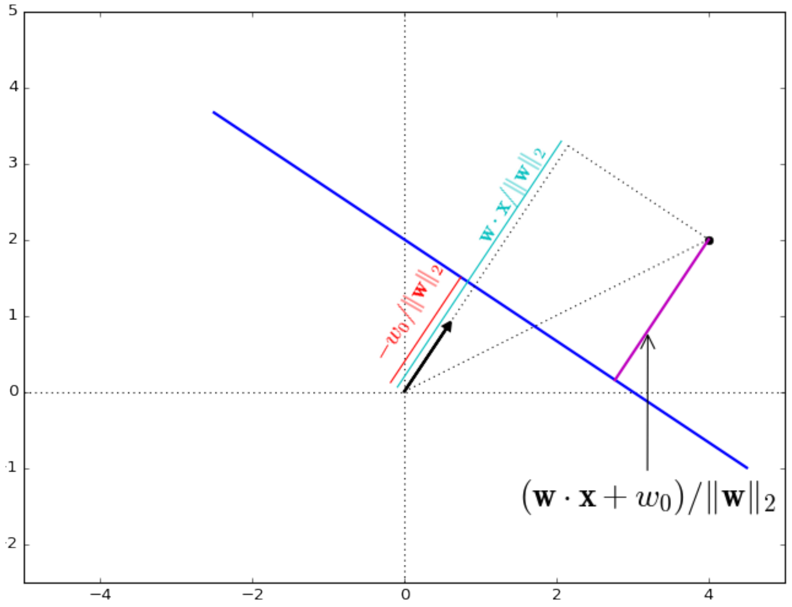


Figure 2: Figure depicting calculations of the distance of a point from a hyperplane.

Most likely, we would conclude that the black line (in the middle for those viewing without colour) is the most suitable, because somehow it separates the data by a larger *margin*. A precise mathematical formulation of this idea gives rise to the *maximum margin* principle and the support vector formulation. The margin for a datapoint is simply the distance of the point from the separating hyperplane. The *maximum margin* principle asserts that the separting boundary that maximises the smallest margin, *i.e.,* has the maxmimum least distance between the data and the boundary, should be preferred.

1.1 Geometry Review

Let us recall some notions from elementary geometry. A linear separator in $\mathbb{R}^D$ is constructed using $\mathbf{w} \in \mathbb{R}^D$ and $w_0 \in \mathbb{R}$; the set of all points $\mathbf{x} \in \mathbb{R}^D$ satisfying $\mathbf{w}\cdot\mathbf{x}+w_0 = 0$ is a hyperplane. A hyperplane divides $\mathbb{R}^D$ into two halfspaces; all points $\mathbf{x}$ on one side of the halfspace satisfy $\mathbf{w}\cdot\mathbf{x}+w_0 > 0$ and those on the other satisfy $\mathbf{w}\cdot\mathbf{x}+w_0 < 0$. The distance of a point $\mathbf{x}$ to the hyperplane is given by:

$$\frac{|\mathbf{w}\cdot\mathbf{x}+w_0|}{\|\mathbf{w}\|_2},$$

where $\|\mathbf{w}\|_2 = \sqrt{\sum_{i=1}^D w_i^2}$ is the Euclidean norm of the vector $\mathbf{w}$. Note that the vector $\mathbf{w}$ points in a direction orthogonal to the hyperplane defined by $\{\mathbf{x} \mid \mathbf{w}\cdot\mathbf{x}+w_0 = 0\}$. A pictorial explanation of this is show in Figure 2.

1.2 SVM Primal Formulation: Linearly Separable Case

Let us now use these distance formulae to formulate the notion of maximum margin in mathematical terms. For now let us continue to assume that we are in the linearly separable setting.

First, let us simply try to write conditions that must be satisfied to find some separating hyperplane (not necessarily one that maximises the margin). Let $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$ be the training data available, with $y_i \in \{-1, +1\}$. We want to find $\mathbf{w} \in \mathbb{R}^D$, $w_0 \in \mathbb{R}$, satisfying:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \qquad\qquad \text{for } i = 1, \ldots, N \tag{1}$$

Let us take a moment to understand what the constraints represented by (1) demand. All points with $y_i = +1$, must have a positive value of $\mathbf{w} \cdot \mathbf{x} + w_0$, *i.e.,* lie in the positive halfspace created by the hyperplane $\mathbf{w} \cdot \mathbf{x} + w_0 = 0$; similarly all points with $y_i = -1$ should lie in the negative halfspace. However, we could have simply used 0 on the RHS of (1) and made the inequalities strict. Why did we not do this? First, because we're dealing with a finite dataset, we will always be able to find some value, say ϵ, such that the constraint in (1) is satisfied by replacing 1 on the RHS by ϵ. However, if some $(\mathbf{w}, w_0)$ satisfies constraints (1) with ϵ on the RHS, then $(\frac{\mathbf{w}}{\epsilon}, \frac{w_0}{\epsilon})$ satisfies (1) with 1 on the RHS. Thus, there is no loss of generality by using 1 on the RHS of (1).[2] So far, we have not talked about the margin; let us now address this. Notice that the margin for any datapoint is given exactly by $\frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)}{\|\mathbf{w}\|_2}$. This is because $|\mathbf{w} \cdot \mathbf{x}_i + w_0| = y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0)$ as $y_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + w_0)$. Thus, the margin for any datapoint must be at least $\frac{1}{\|\mathbf{w}\|_2}$, since they all satisfy (1). To obtain a maximum margin formulation, we could simply maximise $\frac{1}{\|\mathbf{w}\|_2}$ subject to the inequality constraints given in (1). Instead, it is easier to use the equivalent formulation that minimises $\|\mathbf{w}\|_2^2$.

$$\underset{\mathbf{w}, w_0}{\text{minimise}} \qquad \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{2}$$

$$\text{subject to :} \qquad y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \qquad\qquad \text{for } i = 1, \ldots, N \tag{3}$$

Remark: Problem Sheet 3 asks you to show that a more natural formulation involving the margin is equivalent to the formulation presented above.

Note that the above program (defined by (2)-(3)) is a *quadratic program*, as the objective function is quadratic and all the constraints case are linear. In fact, as the objective requires minimising a convex function (why?), and the feasible set in $\mathbb{R}^{D+1}$ ($D+1$ because $\mathbf{w} \in \mathbb{R}^D$ and $w_0 \in \mathbb{R}$) as defined by the constraints in (3) is convex (why?), this is a convex program. In principle, this means that it can be solved by generic convex optimization methods. However, some of these may be quite slow and we will discuss faster methods to solve this particular type of quadratic program.

Figure 3 shows the linear separator obtained by solving the above program (the black line). The figure also shows two parallel hyperplanes (lines) in dotted magenta that contain the points on either side of the separator that are closest to the separator. As these are the points that are closest to the separator, they have the *least* margin, and are called support vectors. (Incidentally, these are also the points for which the inequality (3) will be tight for the optimal solution. This point will be important when we consider the dual formulation. The reason for the name support vectors will also become a bit more clear when we consider the dual formulation of the SVM objective.)

1.3 SVM Primal Formulation: Non-separable Case

Let us now consider what would happen if our data looked like the one shown in Figure 4, where it is clear that there is no linear separator that separates the positive from the negatives.

[2] If we had infinite data, it is possible to envision scenarios where there exists $\mathbf{w}$, w_0 satisfying the inequality $y(\mathbf{w} \cdot \mathbf{x} + w_0) > 0$ for all $(\mathbf{x}, y)$, but not $y(\mathbf{w} \cdot \mathbf{x} + w_0) \geq 1$. For example, imagine the data in one dimension being $\{(x, +1) \mid x \in \mathbb{R}, x > 0\} \cup \{(x, -1) \mid x \in \mathbb{R}, x < 0\}$.

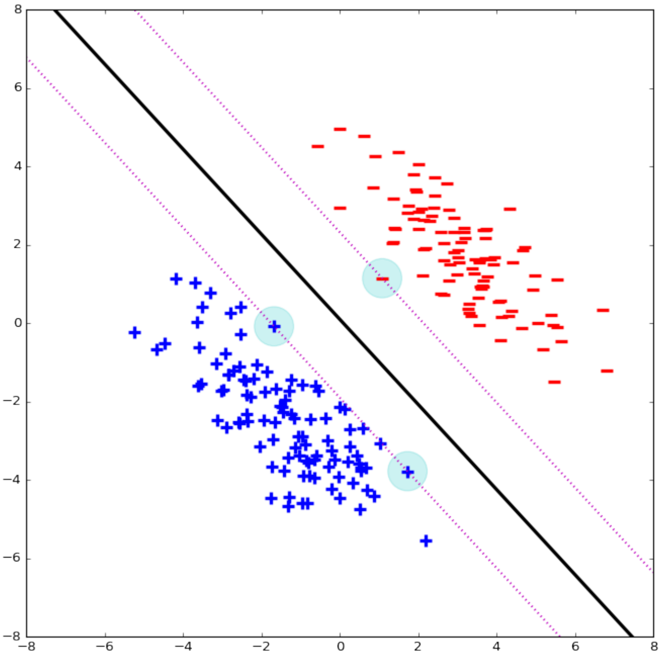


Figure 3: Figure showing the linear separator obtained by solving the SVM program (2, 3). The dotted magenta lines show hyperplanes (lines) parallel to the separator that contain points closest from the separator on either side. These points are called *support vectors* shown in light blue circles.

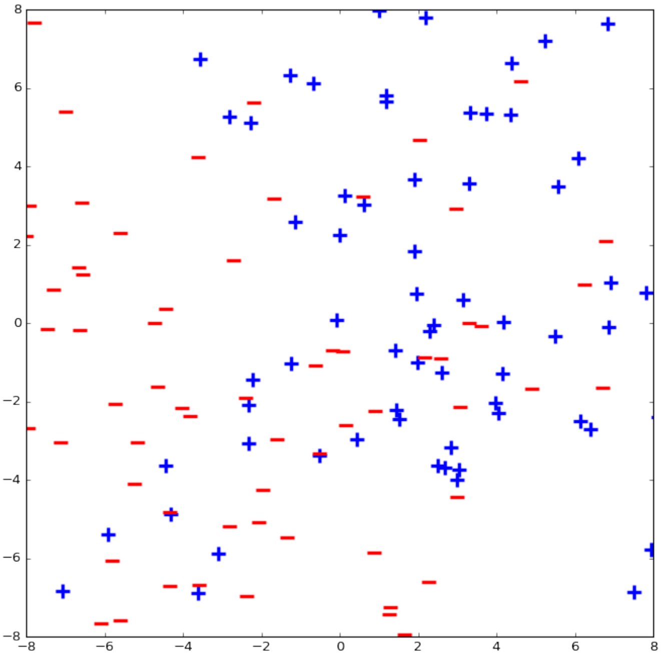


Figure 4: Data that is not linearly separable into positives and negatives.

When faced with such data, of course, we expect both our training error, and as a consequence generalisation (or test) error, to be non-zero. So far, the classifiers (and regression models) that we have studied have naturally incorporated imperfect data through minimising a loss function (or negative log-likelihood). However, the problem formulation as framed in (2)-(3) admits no solution when the data is not linearly separable. Thus, we have to change the problem formulation itself.

What would be a good relaxation? Since we are interested in classification, a natural goal is to look for a linear separator that makes the least mistakes (errors) on the training data. Unfortunately as formulated, this problem is NP-hard. Thus, we have no hope of solving the resulting optimisation problem within feasible computational resources.[3] While not exactly mathematically accurate, the idea of minimising misclassified points roughly corresponds to solving the optimisation problem defined by (2)-(3), where we ask that as many constraints given by (3) as possible should be satisfied. An alternative, and more feasible, approach is to allow all the constraints to be satisfied with some slack. This is done by introducing slack variables, ζ_i, one for every constraint. We then add a penalty for using the slack terms in the objective function. Thus, we get the quadratic program:

$$\begin{aligned}
&\underset{\mathbf{w}, w_0}{\text{minimise}} && \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\zeta_i && && (4)\\
&\text{subject to :} && y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \zeta_i && \text{for } i = 1, \ldots, N && (5)\\
& && \zeta_i \geq 0 && \text{for } i = 1, \ldots, N && (6)
\end{aligned}$$

The addition of slack variables ζ_i ensures that a feasible solution to the program always exists, by allowing ζ_i to be as large (and non-negative) as necessary. Of course, the objective function penalises these large ζ_i values. Let us briefly discuss why we have added the constraint $\zeta_i \geq 0$. If we allowed ζ_i to be negative, then for a point that is correctly classified and far from the separator, setting ζ_i to be large and negative would allow the program to compensate some of the penalty incurred on points that are misclassified. However, we don't necessarily think that a point that is correctly classified by a *huge* margin as opposed to simply a *large* margin should help reducing the overall penalty for misclassified points. Thus, forcing ζ_i to be non-negative ensures that there is a penalty incurred for classifying points incorrectly (or being correct but by a very small margin), but there is no bonus for classifying points correctly by a really large margin!

Loss Function View of the SVM Primal Formulation

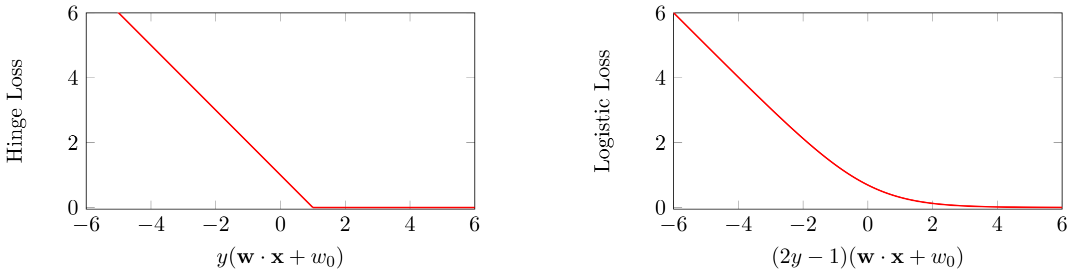
Let us now consider an optimal solution $(\mathbf{w}, w_0, \boldsymbol{\zeta})$ obtained by solving the program (4)-(6), where $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_N)$. Note that the optimal solution must satisfy the following for every $i = 1, \ldots, N$: either $\zeta_i = 0$ or $\zeta_i = (1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0)) \geq 0$. Alternatively, we can write this as:

$$\zeta_i = \max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0)\}$$

Thus, let $\ell_{\text{hinge}}(\mathbf{w}, w_0; \mathbf{x}_i, y_i) = \max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0)\}$ denote the *hinge loss* function, so called because of its shape, shown in Figure 5(a). Then we could alternatively view the SVM formulation as simply minimising the following objective function:

$$\mathcal{L}_{\text{SVM}}(\mathbf{w}, w_0 \mid \mathbf{X}, \mathbf{y}) = \underbrace{\frac{1}{2}\|\mathbf{w}\|_2^2}_{\text{Regularizer}} + \underbrace{C\sum_{i=1}^{N}\ell_{\text{hinge}}(\mathbf{w}, w_0; \mathbf{x}_i, y_i)}_{\text{Loss}}$$

[3] Unless of course P = NP!



(a) Hinge Loss with $y \in \{-1, 1\}$

(b) Logistic Loss with $y \in \{0, 1\}$

Figure 5: Comparison of loss functions for SVM and logistic regression.

As a comparison, let us view the negative log-likelihood of logistic regression on a single datapoint $(\mathbf{x}_i, y_i)$ as a loss function. As $y_i \in \{0, 1\}$ for logistic regression, to compare more effectively, let $z_i = (2y_i - 1)$, so that $z_i = 1$ if $y_i = 1$ and $z_i = 0$ when $y_i = -1$. Then the negative log likelihood for a single point $(\mathbf{x}_i, y_i)$ is given by:

$$\begin{aligned}\mathrm{NLL}(y_i; \mathbf{w}, \mathbf{x}_i) &= -\left(y_i \log\left(\frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}_i}}\right) + (1 - y_i) \log\left(\frac{1}{1 + e^{\mathbf{w} \cdot \mathbf{x}_i}}\right)\right) \\ &= \log\left(1 + e^{-z_i(\mathbf{w} \cdot \mathbf{x}_i)}\right) = \log\left(1 + e^{-(2y_i - 1)(\mathbf{w} \cdot \mathbf{x}_i)}\right)\end{aligned}$$

We can view this as a loss function, called logistic loss, and shown in Figure 5(b).

2 Support Vector Machines: Dual Formulation

Let us now consider the dual form of the Optimization Problem (4)-(6). Let us begin by looking at constrained optimization problems a bit more generally before we study the specific dual SVM problem.

2.1 Constrained Minimisation Problems

By necessity our discussion in this section will be rather terse and brief. For further details please refer to a textbook on optimisation, *e.g.,* (Boyd and Vandenberghe, 2004). Let us consider the primal form of a constrained minimisation problem.

$$\text{minimise} \quad F(\mathbf{z}) \tag{7}$$

subject to:

$$g_i(\mathbf{z}) \geq 0 \qquad i = 1, \ldots, m \tag{8}$$

$$h_j(\mathbf{z}) = 0 \qquad j = 1, \ldots, l \tag{9}$$

We can look at the Lagrange form of the above constrained minimisation. In order to do so, we need to introduce dual variables α_i, $i = 1, \ldots, m$ for each of the inequality constraints (8) and μ_j, $j = 1 \ldots, m$ for each of the equality constraints (9). Then, the Lagrange form is expressed as:

$$\Lambda(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\mu}) = F(z) - \sum_{i=1}^{m} \alpha_i g_i(\mathbf{z}) - \sum_{j=1}^{l} \mu_j h_j(\mathbf{z}) \tag{10}$$

Karush-Kuhn-Tucker (or KKT) conditions are necessary conditions for a criticial point $(\mathbf{z}^*, \boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)$ of Λ to be be a minimum of the constrained minimisation problem (7)-(8). These conditions are:

1. **Primal Feasibility**:

$$g_i(\mathbf{z}^*) \geq 0 \qquad i = 1, \ldots, m \tag{11}$$

$$h_j(\mathbf{z}^*) = 0 \qquad j = 1, \ldots, l \tag{12}$$

2. **Dual Feasibility**:

$$\alpha_i^* \geq 0 \qquad i = 1, \ldots, m \tag{13}$$

3. **Complementary Slackness**:

$$\alpha_i^* g_i(\mathbf{z}^*) = 0 \qquad i = 1 \ldots, m \tag{14}$$

For convex minimisation problems, *i.e.,* when F and $g_1, \ldots, g_m$ are convex and $h_1, \ldots, h_l$ are affine,[4] the KKT conditions are necessary and sufficient. This is the case for the SVM Optimisation Problem (4)-(6).

2.2 SVM Dual Optimisation Problem

Let us now write the Lagrange form for the SVM optimisation problem (4)-(6) by introducing the variables α_i and μ_i for the constraints given in (5) and (6) respectively. Then the Lagrange form is:

$$\Lambda(\mathbf{w}, w_0, \boldsymbol{\zeta}; \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{i=1}^{N} \zeta_i - \sum_{i=1}^{N} \alpha_i \left(y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0 - 1 + \zeta_i)\right) - \sum_{i=1}^{N} \mu_i \zeta_i \tag{15}$$

In order to find critical points of Λ, we need to differentiate with respect to all the variables. Let us first focus on the primal variables, $\mathbf{w}, w_0, \zeta_1, \ldots, \zeta_N$. We can write the following:

$$\frac{\partial \Lambda}{\partial w_0} = -\sum_{i=1}^{N} \alpha_i y_i \tag{16}$$

$$\frac{\partial \Lambda}{\partial \zeta_i} = C - \alpha_i - \mu_i \tag{17}$$

$$\nabla_{\mathbf{w}} \Lambda = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{18}$$

Let us now substitute the expressions for the primal variables obtained by setting the derivatives in (16)-(18) to 0. The constraints $\mu_i \geq 0$ in (13) and $C - \alpha_i - \mu_i = 0$ obtained by setting $\frac{\partial \Lambda}{\partial \zeta_i} = 0$ in (17), can be replaced by the constraints $0 \leq \alpha_i \leq C$ and as a result the variables μ_i can be eliminated entirely. Substitiuting $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$ in (15) and using the constraint $C - \alpha_i - \mu_i = 0$, we get a reduced expression for the Lagrange function.

$$\begin{aligned} g(\boldsymbol{\alpha}) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \cdot \left(\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j\right) \\ &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \end{aligned}$$

[4] An affine function is of the form $w_0 + w_1 z_1 + \cdots w_D z_D$.

In order to the critical points of Λ satisfying the KKT conditions (11)-(14), it is sufficient to find the critical points of g that satisfy the constraints, $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$ (obtained by using (16) and setting $\frac{\partial \Lambda}{\partial w_0} = 0$). However, we observe that g is a concave quadratic function and hence has a unique critical point which is also a global maximum of g. This gives rise to the *dual problem* which we now state formally below.

$$\text{maximise} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{19}$$

subject to :

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{20}$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, m \tag{21}$$

Let us compare the form of dual to that of the primal. In both cases the objective function is quadratic, in the primal we are minimising a convex function and in the dual we are maximising a concave function. However, the constraints in the dual form are extremely simple, most of them are *box constraints* simply stipulating that each α_i lie in the interval $[0, C]$; there is also one additional linear equality constraint. On the other hand the primal constraints may define a rather complex polytope. Also the number of variables in the primal formulation is $N+D+1$, whereas in the dual there are only N variables. This makes it particularly useful if D is very large, either to begin with, or because we perform basis expansion to capture non-linear discriminating boundaries. However, using the dual form does come at some cost: the dual objective function requires N^2 terms to express, whereas the primal objective only requires $N + D$. Thus, when D is much smaller than N, it may still be more efficient to solve the primal form directly.

Let us also look at the implications of the complementary slackness conditions. What these conditions say is that for any i,

$$\alpha_i \cdot y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0 - 1 + \zeta_i) = 0$$

This means that either $\alpha_i = 0$ or $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 1 - \zeta_i$, *i.e.,* the corresponding primal constraint is satisfied with equality. This implies that $\mathbf{x}_i$ is a *support vector* as it must be one of the points with the least *margin* (see Fig. 3). The name *support vectors* becomes clearer when we see that the solution $\mathbf{w}$ is of the form:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

Only the points $\mathbf{x}_i$ for which the corresponding $\alpha_i > 0$ determine the solution, *i.e.,* they form the support of the solution.

Exercise: We have seen how using the dual problem and obtaining the solution $\boldsymbol{\alpha}$, we can express $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$. There remains the problem of finding w_0. Outline how you would find the value of w_0 given the optimal solution to the dual $\boldsymbol{\alpha}$. (*Hint*: Once $\mathbf{w}$ is known you are essentially solving a one-dimensional problem to find w_0; you can find the optimal w_0 in linear time given $(\mathbf{w} \cdot \mathbf{x}_i, y_i)$ for $i = 1, \ldots, N$.)

3 Kernels

Let us observe the form of the dual objective (19). Notice that an input $\mathbf{x}_i$ only appears in the objective as inner products with other inputs. Likewise, using (18), we know that the optimal solution $\mathbf{w}$ is of the form:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

And hence for any new point $\mathbf{x}_{\text{new}}$,

$$\mathbf{w} \cdot \mathbf{x}_{\text{new}} = \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}_{\text{new}}) \tag{22}$$

Thus, to make a prediction on a new point, $\mathbf{x}_{\text{new}}$, we only require the ability to compute inner products with the datapoints, in fact, only with the support vectors. Thus, the SVM objective could be solved using blackbox access to a function $\kappa(\cdot, \cdot)$ that given two inputs $\mathbf{x}$, $\mathbf{x}'$, returns $\mathbf{x} \cdot \mathbf{x}'$. This is essentially a *kernel* function.

3.1 Polynomial Basis Expansion

Let us consider the case where we perform a polynomial basis expansion up to degree d. As an illustrative example, let us consider the case when $D = 2$ and $d = 2$. Thus, the original inputs are points in $\mathbb{R}^2$ and we consider the map, $\mathbf{x} \mapsto \psi_2(\mathbf{x})$, where,

$$\psi_2([x_1, x_2]^{\mathsf{T}}) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\right]^{\mathsf{T}} \tag{23}$$

Note that the dual formulation still has N variables, the only difference is that the inner products, $\mathbf{x}_i \cdot \mathbf{x}_j$, need to be replaced by $\psi_2(\mathbf{x}_i) \cdot \psi_2(\mathbf{x}_j)$. (Likewise, when making predictions.) On the other hand, the primal problem, if using degree d basis expansion with D input features, would have roughly D^d variables. Thus, the dual form allows us to fit increasingly complex classifiers without significant increase in computational cost. Of course, there is sill the problem of *generalization*, *i.e.,* if you fit increasingly complex classifiers with the same amount of data and without regularization, you will *overfit*.

Let us address the computational question is greater detail. While it is true that the number of inner products required to express the dual problem is always roughly $N^2/2$, computing each inner product when using degree d polynomial expansion may take time roughly D^d. The *kernel trick* allows us to avoid this. Let us consider a slightly different polynomial basis expansion than the one in (23), a map $\mathbf{x} \mapsto \phi_2(\mathbf{x})$, given by:

$$\phi_2([x_1, x_2]^{\mathsf{T}}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right]^{\mathsf{T}} \tag{24}$$

And for two inputs, $\mathbf{x}$, $\mathbf{x}'$, consider,

$$\begin{aligned} \phi_2(\mathbf{x}) \cdot \phi_2(\mathbf{x}') &= 1 + 2x_1 x_1' + 2x_2 x_2' + x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2x_1 x_1' x_2 x_2' \\ &= (1 + \mathbf{x} \cdot \mathbf{x}')^2 \end{aligned} \tag{25}$$

In general, if we have a degree d expansion, we can use a suitable expansion $\mathbf{x} \mapsto \phi_d(\mathbf{x})$, so that

$$\phi_d(\mathbf{x}) \cdot \phi_d(\mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^d \tag{26}$$

Note that ϕ_d still contains all the monomials of total degree up to d, and hence a linear classifer using $\phi_d(\mathbf{x})$ as features can represent arbitrary degree d separating surfaces. However, if we let $\kappa_d(\mathbf{x}, \mathbf{x}') = \phi_d(\mathbf{x}) \cdot \phi_d(\mathbf{x}')$, then κ_d can be computed in time only $O(n \log d)$! The function κ_d is known as the degree-d polynomial kernel.

3.2 Mercer Kernels

The idea of kernels is considerably more general than outlined above. Let $\mathcal{X}$ denote the set of inputs (not even necessarily some subset of $\mathbb{R}^D$); a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *kernel.* We'll focus on kernels that are symmetric and non-negative, though kernels can be much more general. The *Gram matrix* is defined as

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \kappa(\mathbf{x}_N, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \tag{27}$$

for inputs $\langle \mathbf{x}_i \rangle_{i=1}^N$, each $\mathbf{x}_i \in \mathcal{X}$. A kernel is a *Mercer kernel*, or a *positive definite kernel* if the Gram matrix is always positive semi-definite. Any kernel that can be viewed as arising from an inner product (possibly in an expanded basis), is a Mercer kernel. However, there are other kernels that can be shown to be Mercer kernels. In fact the ability to define kernels directly without explicitly providing a feature expansion is one of their very appealing properties. As we shall see in the examples in the next section, kernels may be defined directly on discrete objects; constructing a kernel that captures similarity between discrete objects is often easier than designing a map from a discrete space to a real vector space. However, the task of proving that kernels are Mercer kernels is not always easy. These are a few rules that allow us to easily compose kernels that are known to be Mercer kernels: Supose that κ_1 and κ_2 are Mercer kernels and $\alpha \in \mathbb{R}^+$, then $\kappa_1 + \kappa_2$, $\alpha\kappa_1$ and $\kappa_1 \cdot \kappa_2$ are all Mercer kernels. Furthermore, if $\mathcal{X} \subseteq \mathbb{R}^D$, then for any $D \times D$ positive definite matrix $\mathbf{B}$, $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\mathsf{T}\mathbf{B}\mathbf{x}'$ is a Mercer kernel.

Why Mercer Kernels?

A natural question is why the emphasis on Mercer kernels? Let us look at the SVM dual objective in terms of the kernel.

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Or more succintly, if $\mathbf{K}$ is the Gram matrix, $\mathbf{y} \odot \boldsymbol{\alpha}$ is the elment-wise product of the two vectors, also known as the *Hadamard* product, and $\mathbf{1}$ is the vector of all ones,

$$g(\boldsymbol{\alpha}) = \mathbf{1}^\mathsf{T}\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha} \odot \mathbf{y})^\mathsf{T}\mathbf{K}(\boldsymbol{\alpha} \odot \mathbf{y}) \tag{28}$$

If $\mathbf{K}$ is positive semi-definite, then $g(\boldsymbol{\alpha})$ is concave and hence attains a unique maximum value; if $\mathbf{K}$ is positive definite then the maximum value is achieved at a unique point. Thus, the principal reason for choosing Mercer kernels is that the resulting SVM dual optimisation problem can be solved using convex programming techniques. Otherwise, the KKT conditions are no longer sufficient and the optimisation may have to be performed using non-linear programming methods; this means is that we either have to settle for local optima or run in time that is not necessarily polynomial in the input size, which is typically infeasible for all but very small problems.

Theory of Kernels

We have alluded to the fact that if a kernel can be expressed as an inner product then it is a Mercer kernel. In fact the converse is also true, however, the resulting inner product space is not necessarily finite dimensional. There is a rich and deep theory of kernel methods which is beyond the scope of this course. However, the interested student may begin with the books by Shawe-Taylor and Cristianini (2004) and Schölkopf and Smola (2002).

3.3 Examples of Mercer Kernels

We've already seen an example of a kernel, the degree d polynomial kernel in Section 3.1. Let us now consider a few more examples. Further examples are provided in (Murphy, 2012, Chap 14.2) and the books by Shawe-Taylor and Cristianini (2004) and Schölkopf and Smola (2002).

3.3.1 RBF Kernels

The spherical Gaussian kernel is defined by,

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right) \tag{29}$$

Here σ is known as the bandwidth. We encountered these kernels when studying kernel ridge regression, where we used the width parameter $\gamma = \frac{1}{2\sigma^2}$. The spherical Gaussian kernel is indeed a Mercer kernel. The category of kernels that only depend on $\|\mathbf{x} - \mathbf{x}'\|_2$ are known as RBF or *radial basis function* kernels.

It is possible to use more general Gaussian kernels with a 'covariance' matrix $\mathbf{\Sigma}$, however, these kernels are no longer RBF kernels (though they are Mercer kernels).

3.3.2 String Kernels

The power of kernel methods is that kernels can be defined directly on discrete objects, *e.g.,* strings. Let us consider an example (taken from (Murphy, 2012, Chap 14.2.6)). Proteins can be viewed as sequences of amino acids represented as strings. There are 20 relevant amino acids denoted by letters in the alphabet $\mathcal{A} = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$. If we have two sequences say $\mathbf{x}$ and $\mathbf{x}'$ of lengths 107 and 150 respectively:

$$\begin{aligned}
\mathbf{x} &= \texttt{IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLEQTVQGGTV} \\
&\ \texttt{ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDYLQEFLGVMNTEWI} \\
\mathbf{x}' &= \texttt{PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAERLQENLQAYRTFHVLLA} \\
&\ \texttt{RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK} \\
&\ \texttt{LWGLKVLQELSQWTVRSIHDLRFISSHQTGIP}
\end{aligned}$$

These strings have the string `LQE` in common. In general for every string $s \in \mathcal{A}^*$ (the set of all finite strings over the alphabet $\mathcal{A}$), we can have a weight w_s and define the kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(\mathbf{x}) \phi_s(\mathbf{x}') \tag{30}$$

where $\phi_s(\mathbf{x})$ and $\phi_s(\mathbf{x}')$ denote the number of occurrences of the string s as a substring in $\mathbf{x}$ and $\mathbf{x}'$ respectively. This is a Mercer kernel and can be computed efficiently in time $O(|\mathbf{x}| + |\mathbf{x}'|)$, where $|\mathbf{x}|$ denotes the length of the string $\mathbf{x}$.

3.4 Discussion

Kernel methods are widely applicable in machine learning and not just restricted to support vector machines. For instance, linear regression using kernel basis expansion, is nothing but a *kernelised* form of linear regression. Let us briefly revisit the Ridge regression objective:

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2$$

We observe that the optimal solution $\mathbf{w}$ can always be written as $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$. Why? Suppose not. Let us write $\mathbf{w} = \mathbf{w}_s + \mathbf{w}_\perp$, where $\mathbf{w}_s$ is the component of $\mathbf{w}$ that lies in the span$\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\mathbf{w}_\perp$ is orthogoal to this linear subspace spanned by the inputs. Then, since $\mathbf{w}_\perp \cdot \mathbf{x}_i = 0$ for all $i = 1, \ldots, N$, $\mathbf{w}_\perp$ cannot affect $(\mathbf{w} \cdot \mathbf{x}_i - y_i)^2$ at all. However, $\|\mathbf{w}\|_2^2 = \|\mathbf{w}_s\|_2^2 + \|\mathbf{w}_\perp\|_2^2$. Thus, setting $\mathbf{w}_\perp = \mathbf{0}$ can only decrease $\mathcal{L}_{\text{ridge}}$. Thus, equivalently we can try to solve for α_i rather than $\mathbf{w}$, and we get a *kernelised* form of linear regression. This is exactly what you did on Problem 3 on Sheet 1.

4 Multiclass Classification

Let us briefly discuss some generic approaches to convert binary classifiers to multiclass classifiers. We've already seen that logistic regression yields itself to a rather natural multiclass formulation. In the case of the *maximum-margin* principle, it is possible, though mathematically a lot more involved to formulate a multiclass framework. However, in practice one of the two following approaches is far more common.

4.1 One-vs-One (ovo)

Suppose our data has K different classes, *i.e.,* $y \in \{1, \ldots, K\}$. The idea is simply to train $\binom{K}{2}$ different binary classifiers for every possible pair of classes.

Now suppose we have these classifiers trained, at test time when presented with a new input $\mathbf{x}_{\text{new}}$, what class do we output? Each of the $\binom{K}{2}$ classifiers give us one possible class label. In the end we can simply choose the most commonly occuring, or *plurality*, label as our ouptut.

4.2 One-vs-Rest (ovr)

In this case, instead of training $\binom{K}{2}$ classifiers only K classifiers are trained. In each case data from one class is considered as positive and all of the remaining data is considered negative. For this reason it is called the one-vs-rest or one-vs-all multiclass classification.

Again, when we have a new input $\mathbf{x}_{\text{new}}$ at test time, how do we choose the label? In this case, if we are lucky, exactly one of the K classifiers will classify this as 'positive' or or on the 'one' side, and the remaining will classify it as 'negative' or on the 'rest' side. However, this is a rather optimistic view, we may find ourselves in a situation where all classifers assign the new point to the 'rest' side or more than one classifier assigns it to the 'one' side. In this case, we rely on the fact that many classifiers, such as SVMs or logistic regression, actually output $\mathbf{w} \cdot \mathbf{x}_{\text{new}}$ and then threshold it. Thus, we may use the value of $\mathbf{w} \cdot \mathbf{x}_{\text{new}}$ to assign the label.

Discussion

The one-vs-one (ovo) and one-vs-rest (ovr) methods for multiclass classification can be considered as somewhat hacky. In general, there is no principled way to choose the correct class label, however, it is observed that these methods work rather well in practice.

There is no clear answer as to which one of the two is to be preferred. Let us discuss some tradeoffs here. On the one hand in the ovo approach, we are required to train $\binom{K}{2}$ classifiers as opposed to just K classifiers in ovr. Thus, there may be a clear computational advantage to using ovr. However, note that the problem size in the case of ovo may be substantially smaller, since we are discarding all but two classes to train the classifier. Thus, if the running time of the learning algorithm scales poorly with data size, say N^2 or N^3, then even though we are training fewer classifiers, the fact that we are training on smaller sample sizes may be helpful.

Also, one may argue that ovr depends on rather unnatural assumptions regarding decision boundaries. Suppose you were using support vector machines for digit classification. In the ovr approach, the classifier training 0 vs 1-9, is looking for a separator that puts 0s on one side and all of the remaining digits on the other. However, the assumption that there is a linear separating surface (or a non-linear one, but of relatively low-complexity if using basis expansion) separating 0s from all the other digits may be much stronger than assuming that there is such a separating surface between 0s and each of the other digits considered individually. Thus, the ovo approach may create more natural 'learning problems'.

It is possible to use more sophisticated approaches; one possible approach is to use some sort of error-correcting-codes (ECC). The basic idea is to divide the classes into several pairs of disjoint subsets. (Typically, all classes are included in at least some of the pairs, though not all classes are included in every pair.) Then for each pair, a binary classifier is trained to separate these two subsets. If this is done with enough pairs of subsets, an error correcting approach can be used to determine the class label. While it is beyond the scope of the course to provide the detailed history of this approach, it is well worth reading the paper by Allwein et al. (2000) which received the ICML test of time award in 2010.

5 Measuring Performance

Let us briefly discuss how to measure performance of the classifiers we've trained. For regression problems, our learning algorithms minimised some objective function on the training data, *e.g.,* sum of squared residuals. Since the goal was to predict some real number as output, the value of this same loss function on the test set itself served as a reasonable measure of performance. In principle we could measure performance using some alternative criterion.

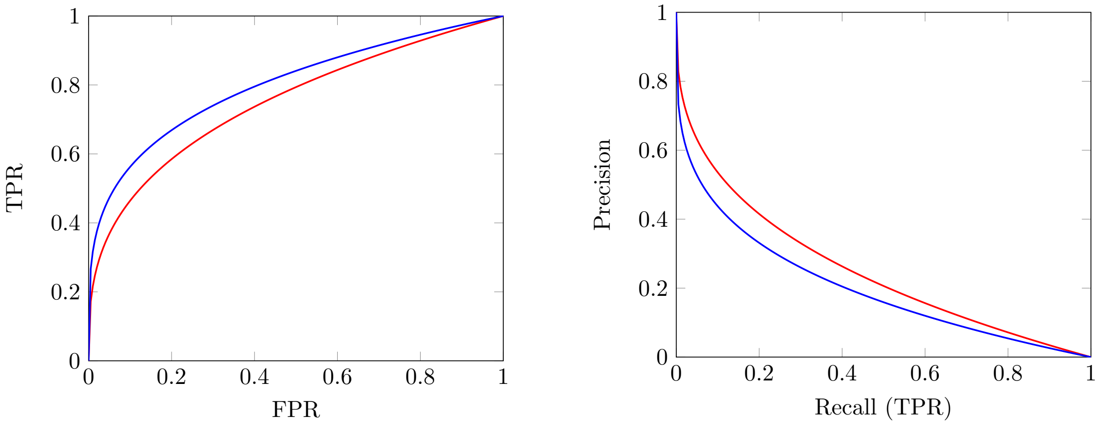
In the case of classification this need is more obvious. At the end of the day, we are less interested in what value the likelihood or average hingeloss had on some training/test data, but more interested in what the misclassification rate was. Thus, although the learning algorithms (a.k.a. training procedures) may seek to minimise some objective such as the *hinge loss* in the case of SVM, or negative log-likelihood in the case of logistic regression, we can (and perhaps should) actually measure the test error as simply the number of misclassified points.

In certain cases, not all mistakes are equally problematic. Suppose you are training a classifier to determine if a comment by a newspaper reader counts as abuse or not, depending on what you value more, free expression or preventing online abuse, you may want to treat false positive detection of abuse as more or less important. Similarly for medical diagnoses, it may be preferable to err on one side or the other. Let us introduce names for some of these notions in the case of binary classification. We can draw a 2×2 table with actual labels (as columns) and model predictions (as rows).

	Actual Labels	
Prediction	yes	no
yes	true positive	false positive
no	false negative	true negative

Due to their origins in statistics, false positive errors are also called Type I errors, and false negative errors as Type II errors. For some classifier and its prediction on a dataset, let TP, TN, FP and FN denote the number of datapoints that are true positives, true negatives, false positives and false negatives respectively. Let us also introduce a few other terms that are commonly used.

- **True Positive Rate**: The true positive rate (TPR), a.k.a. sensitivity or recall is the ratio of the number of true positives to the number of actual positives, *i.e.,* $\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}}$.
- **False Positive Rate**: The false positive rate (FPR), is the ratio of the number of false positives to the number of actual negatives, *i.e.,* $\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}$.



(a) ROC Curve

(b) Precision-Recall Curve

Figure 6: Curves showing tradeoff between (a) TPR and FPR (ROC curve), and (b) Precision and Recall (FPR)

- **Precision**: The precision is the ratio of the number of true positives to the number of predicted positives, *i.e.,* $\text{FPR} = \frac{\text{TP}}{\text{FP+TP}}$.

For classifiers such as logistic regression, the classifier actually models the conditional distribution on the labels, $p(y = 1 \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w} \cdot \mathbf{x})$. Ordinarily, we simply use the threshold of $1/2$ to label a point positive, however, if we value two types of mistakes differently, we may wish to lower or raise the threshold. For example, if we truly want very few false positives, we could raise the threshold to 0.9, so that the classifier would predict something as positive only if it were 90% sure (according to the model) that the point were positive. Similarly, for generative models, the actual decision is made using the ratio $p(y = 1 \; |\mathbf{x}, \boldsymbol{\theta})/p(y = 0 \mid \mathbf{x}, \boldsymbol{\theta})$; when treating all errors equally, it is natural to threshold this ratio at 1 to decide whether to predict 1 or 0, but when one type of errors are more costly we may lower or raise the threshold. Finally, for classifiers like SVM, where there is no natural probabilistic interpretation, we can simply modify the objective function to penalise false positive errors more than false negative errors, or the other way round. In general, as we vary some parameter controlling our desired tradeoff between the false positive rate (FPR) and the true positive rate (TPR), we can plot what is called the *Receiver Operating Characteristic* or ROC curve. Typically, this curve will go from the point $(0, 0)$ where all predictions are negative, so there are no true or false positive predictions, to the point $(1, 1)$ where all predictions are positive, so both FPR and TPR are 1. Depending on our desired tradeoff, we may wish to pick one particular point on the curve; the area under the curve (AUC) for the ROC curve is typically used as a measure to determine which classifiers allow a good tradeoff. Obviously, the maximum (and the best) possible value for AUC is 1; and the worst is 0. (See Fig. 6(a).)

In certain cases, such as when searching for documents in a large corpus about a certain topic, most examples will be labelled negative. Thus, in this case the more interesting tradeoff is between precision and recall. Note that precision indicates what fraction of returned data (documents predicted as positive) were actually true positives (relevant documents), whereas recall indicates what fraction of positive data (truly relevant documents) were returned (true positives). As in the case of the ROC curve, there is a natural tradeoff between precision and recall which is called the precision-recall tradeoff curve. (See Fig. 6(b).)

Confusion Matrix

When there are more than two classes, it is common to output a *confusion* matrix. The rows indicated the true labels of the data whereas the columns indicate the predicted labels. Thus, the number N_{ij} indicates the number of items of class j in the dataset that were predicted to be of class i. Obviously, a good classifier will have large entries on the diagonal and smaller entries off diagonal. But the confusion matrix may be useful in analysing the discrepancies in greater detail.

	Actual Labels			
Prediction	1	2	$\cdots$	K
1	N_{11}	N_{12}	$\cdots$	N_{1K}
2	N_{21}	N_{22}	$\cdots$	N_{2K}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
K	N_{K1}	N_{K2}	$\cdots$	N_{KK}

References

Erin L Allwein, Robert E Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Kevin P. Murphy. *Machine Learning : A Probabilistic Perspective*. MIT Press, 2012.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.