Linear Models

1. Definition: If x1 and x2 are two scalars or vectors of the same dimension and α and β are arbitrary scalars, then αx1+βx2 is called a linear combination of x1 and x2. If f is a linear function of x, then f (αx1 +βx2) = αf (x1)+βf (x2)

The function value of a linear combination of some inputs is a linear combination of their function values. As a special case, if $\beta = 1-\alpha$ we are taking a weighted average of $x1$ and $x2$, and the linearity of f then means that the function value of the weighted average is the weighted average of the function values. Linear functions take particular forms, depending on the domain and codomain of f. If x and $f(x)$ are scalars, it follows that f is of the form $f(x) = a+bx$ for some constants a and b; a is called the *intercept* and b the *slope*. If $\mathbf{x} = (x1, \ldots, xd)$ is a vector and $f(\mathbf{x})$ is a scalar, then f is of the form

$$f(\mathbf{x}) = a + b1x1 + \ldots + bd\,xd = a + \mathbf{b}\cdot\mathbf{x}$$

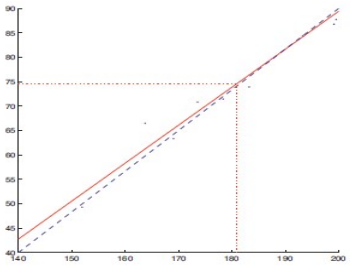
with $\mathbf{b} = (b1, \ldots, bd)$. The equation $f(\mathbf{x}) = 0$ defines a plane in Rd perpendicular to the *normal vector* **b**.

1.1 Characteristics of Linear Model:

a. Linear models are *parametric*, meaning that they have a fixed form with a small number of numeric parameters that need to be learned from data. This is different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.
b. Linear models are stable, which is to say that small variations in the training data have only limited impact on the learned model. Tree models tend to vary more with the training data, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well.
c. Linear models are less likely to overfit the training data than some other models, largely because they have relatively few parameters. The flipside of this is that they sometimes lead to underfitting: e.g., imagine you are learning where the border runs between two countries from labelled samples, then a linear model is unlikely to give a good approximation.

2. The least-squares method:

a. The method of least squares is about estimating parameters by minimizing the squared discrepancies between observed data and their expected values on the other.
b. Considering an arbitrary straight line, $y=b_0+b_1x$ is to be fitted through these data points.



c. The Least Squares (LS) criterion states that the sum of the squares of errors is minimum. The least squares yields y(x), whose elements sum to 1, but do not ensure the outputs to be in the range [0,1].
d. Draw a line based on the data points and assume the imaginary line of y=a+bx and imagine a vertical distance between the line and a data point and calculate E=Y-E(Y), this error is the deviation of the data point from the regression line.
e. Let us get a and b that can minimizes the sum of squared deviations. This method is called "Least Squares". The process of getting parameter estimators is called "estimations". Least Square method is the estimation method of Ordinary Least Squares (OLS)

2.1 Univariate Analysis:

1. The simplest form of regression analysis is a univariate regression or a model with one independent variable.
2. Univariate analysis is the simplest form of data analysis where the data being analyzed contains only one variable. Since it's a single variable it does not deal with causes or relationships.
3. The main purpose of univariate analysis is to describe the data and find patterns that exist with in it.
4. Assuming a linear relationship between the independent and dependent variables, the general equation can be written as:

$$W_i = \alpha + \beta_i + \epsilon$$

We can write univariate linear regression in matrix form as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} a + \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} b + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

5. The parameter α is called "intercept" or the value of W, when x=0
6. Since there is only one independent variable, regression analysis estimates parameters that provide the "best fitting" line when the dependent variable is graphed on the vertical axis and independent variable is on horizontal axis.

Ex: For a given data having 100 examples, if squared errors SE1,SE2, and SE3 are 13.33,3.33 and 4.00 respectively, calculate Mean Squared Error (MSE).

Sol: Mean Squared Error= $\frac{\sum SEi}{n}$

MSE= $\frac{13.33+3.33+4}{3}$ = 0.2066

2.2 Multivariate Regression:

1. In order to deal with an arbitrary number of features it will be useful to employ matrix Notation
2. The *multivariate regression* problem is

$$\hat{w} = S^{-1}X^{T}y$$

3. Assume that the features are uncorrelated (meaning the covariance between every pair of different features is 0) in addition to being zero-centred. The covariance matrix **Σ** is diagonal with entries σ_{jj} . Since $\mathbf{X}^{T}\mathbf{X} = n(\mathbf{\Sigma}+\mathbf{M})$, and since the entries of **M** are 0 because the columns of **X** are zero-centred, this matrix is also diagonal with entries $n\sigma_{jj}$ – in fact, it is the matrix **S** referred to above.
4. $(\mathbf{X}^{T}\mathbf{X})^{-1}$ *acts as a transformation that decorrelates, centres and normalises the features*.
5. If multiple independent variables affect the response variable, then the analysis calls for a model different from that used for the single predictor variable.

2.3 Bivariate linear regression in matrix notation:

1. we derive the basic expressions.

$$\mathbf{X}^{T}\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix}\begin{pmatrix} x_{11} & x_{12} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix} = n\begin{pmatrix} \sigma_{11}+\overline{x_1}^2 & \sigma_{12}+\overline{x_1}\,\overline{x_2} \\ \sigma_{12}+\overline{x_1}\,\overline{x_2} & \sigma_{22}+\overline{x_2}^2 \end{pmatrix}$$

2.
$$(\mathbf{X}^{T}\mathbf{X})^{-1} = \frac{1}{nD}\begin{pmatrix} \sigma_{22}+\overline{x_2}^2 & -\sigma_{12}-\overline{x_1}\,\overline{x_2} \\ -\sigma_{12}-\overline{x_1}\,\overline{x_2} & \sigma_{11}+\overline{x_1}^2 \end{pmatrix}$$

3.
$$\mathbf{X}^{T}\mathbf{y} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix}\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = n\begin{pmatrix} \sigma_{1y}+\overline{x_1}\,\overline{y} \\ \sigma_{2y}+\overline{x_2}\,\overline{y} \end{pmatrix}$$

2.4 Regularized Expression:

1. *Regularisation* is a general method to avoid such overfitting by applying additional constraints to the weight vector
2. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*.
3. Process to achieve:
   a. write down the least-squares regression problem as an optimisation problem:
   $$\mathbf{w}^{*} = \arg\min_{\mathbf{w}} (\mathbf{y}-\mathbf{X}\mathbf{w})^{T}(\mathbf{y}-\mathbf{X}\mathbf{w})$$
   b. write the sum of squared residuals as a dot product. The regularised version of this optimisation is then as follows:
   $$\mathbf{w}^{*} = \arg\min_{\mathbf{w}} (\mathbf{y}-\mathbf{X}\mathbf{w})^{T}(\mathbf{y}-\mathbf{X}\mathbf{w}) + \lambda||\mathbf{w}||^{2}$$
   where $||\mathbf{w}||^{2} = \sum_i w_i^2$ is the squared normof the vector w, or, equivalently, the dot product $w^{T}w$; λ is a scalar determining the amount of regularisation.
4. This regularised problem still has a closed-form solution:
   $$\hat{\mathbf{w}} = (\mathbf{X}^{T}\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^{T}\mathbf{y}$$

where **I** denotes the identity matrix with 1s on the diagonal and 0s everywhere else. regularisation amounts to adding λ to the diagonal of $\mathbf{X}^T\mathbf{X}$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-squares regression is known as *ridge regression*.

5. alternative form of regularised regression is provided by the lasso, which stands for 'least absolute shrinkage and selection operator'. It replaces the ridge regularisation term $\sum_i w_i^2$ with the sum of absolute weights $\sum_i |w_i|$.
6. The result is that some weights are shrunk, but others are set to 0, and so the lasso regression favours *sparse solutions.*
7. It should be added that lasso regression is quite sensitive to the regularisation parameter λ, which is usually set on hold-out data or in cross-validation.
8. Also, there is no closed-form solution and so some numerical optimisation technique must be applied.

Using least-squares regression for classification

1. we can also use linear regression to learn a binary classifier by encoding the two classes as real numbers.
2. For instance, we can label the *Pos* positive examples with $y_{\oplus} = +1$ and the *Neg* negative examples with $y_{\ominus} = -1$. It then follows that $\mathbf{X}^T\mathbf{y} = Pos\ \boldsymbol{\mu}_{\oplus} - Neg\ \boldsymbol{\mu}_{\ominus}$, where $\boldsymbol{\mu}_{\oplus}$ and $\boldsymbol{\mu}_{\ominus}$ are d-vectors containing each feature's mean values for the positive and negative examples, respectively.
3. In the general case, the *least-squares classifier* learns the decision boundary $\mathbf{w}\cdot\mathbf{x} = t$ with

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}(Pos\,\mu^{\oplus} - Neg\,\mu^{\ominus})$$

4. We would hence assign class $\hat{y} = \text{sign}(w\cdot x - t)$ to instance x, where

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Various simplifying assumptions can be made, including zero-centred features, equal variance features, uncorrelated features and equal class prevalences.

5. A general way of constructing a linear classifier with decision boundary $w\cdot x = t$ is by constructing w as $M^{-1}(n_{\oplus}\mu_{\oplus} - n_{\ominus}\mu_{\ominus})$, with different possible choices of M, $n_{\oplus}$ and $n_{\ominus}$.
6. The full covariance approach with $M = \mathbf{X}^T\mathbf{X}$ has time complexity $O(n^2d)$ for construction of M and $O(d^3)$ for inverting it,1 so this approach becomes unfeasible with large numbers of features.

3. The Perceptron:

1. A linear classifier that will achieve perfect separation on linearly separable data is the *perceptron*, originally proposed as a simple neural network.
2. The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

3. For example, let $\mathbf{x}i$ be a misclassified positive example, then we have yi =+1and$\mathbf{w}\cdot\mathbf{x}i < t$. We therefore want to find $\mathbf{w}$ such that $\mathbf{w}\cdot\mathbf{x}i > \mathbf{w}\cdot\mathbf{x}i$, which moves the decision boundary towards and hopefully past xi . This can be achieved by calculating the new weight vector as $\mathbf{w} = \mathbf{w}+\eta\mathbf{x}i$, where $0 < \eta \leq 1$ is the *learning rate*.
4. It iterates through the training examples until all examples are correctly classified. The algorithm can easily be turned into an *online* algorithm that processes a stream of examples, updating the weight vector only if the last received example is misclassified.
5. The perceptron is guaranteed to converge to a solution if the training data is linearly separable, but it won't converge otherwise.
6. The key point of the perceptron algorithm is that, every time an example $\mathbf{x}i$ is misclassified, we add $yi\ \mathbf{x}i$ to the weight vector.
7. After training has completed, each example has been misclassified zero or more times – denote this number αi for example $\mathbf{x}i$.

Algorithm: Perceptron(D,η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates; learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w}\cdot\mathbf{x})$.

```
w←0 ; // Other initialisations of the weight vector are possible
converged←false;
while converged = false do
    converged←true;
    for i = 1 to |D| do
        if yiw· xi ≤ 0 // i.e., ŷi = yi
        then
            w←w+η yi xi ;
            converged←false; //We changed w so haven't converged yet
        end
    end
end
```

4. Support vector machines:

- The training examples nearest to the decision boundary are called support vectors: the decision boundary of a support vector machine (SVM) is defined as a linear combination of the support vectors.
- The margin is thus defined as m/||w||, where m is the distance between the decision boundary and the nearest training instances (at least one of each class) as measured along w. Since we are free to rescale t , ||w|| and m, it is customary to choose m = 1.
- Maximising the margin then corresponds to minimising ||w|| or, more conveniently, ½ $||w||^2$, provided of course that none of the training points fall inside the margin. This leads to a quadratic, constrained optimisation problem:

$$\mathbf{w}^*, t^* = \arg\min_{\mathbf{w},t} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

- Adding the constraints with multipliersαi for each training example gives the Lagrange Function

$$\begin{aligned}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n) &= \frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i-t)-1)\\ &= \frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{n}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i)+\sum_{i=1}^{n}\alpha_i y_i t+\sum_{i=1}^{n}\alpha_i\\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w}-\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)+t\left(\sum_{i=1}^{n}\alpha_i y_i\right)+\sum_{i=1}^{n}\alpha_i\end{aligned}$$

- By taking the partial derivative of the Lagrange function with respect to t and setting it to 0 we find that for the optimal threshold t we have $\sum_{i=1}^{n}\alpha_i y_i = 0.$.
- Similarly, by taking the partial derivative of the Lagrange function with respect to **w** we see that the Lagrange multipliers define the weight vector as a linear combination of the training examples:

$$\frac{\partial}{\partial\mathbf{w}}\Lambda(\mathbf{w},t,\alpha_1,\ldots,\alpha_n)=\frac{\partial}{\partial\mathbf{w}}\frac{1}{2}\mathbf{w}\cdot\mathbf{w}-\frac{\partial}{\partial\mathbf{w}}\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)=\mathbf{w}-\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i$$

- Now, by plugging the expressions $\sum_{i=1}^{n}\alpha_i y_i = 0$ and $\mathbf{w}=\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i$ back into the Lagrangian we are able to eliminate **w** and t , and hence obtain the dual optimisation problem, which is entirely formulated in terms of the Lagrange multipliers:

$$\begin{aligned}\Lambda(\alpha_1,\ldots,\alpha_n) &= -\frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)\cdot\left(\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i\right)+\sum_{i=1}^{n}\alpha_i\\ &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j+\sum_{i=1}^{n}\alpha_i\end{aligned}$$

- The dual problem is to maximise this function under positivity constraints and one equality constraint:

$$\alpha_1^*,\ldots,\alpha_n^*=\underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}}-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j+\sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } \alpha_i\geq 0, 1\leq i\leq n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

The following example makes these issues a bit more concrete by showing detailed calculations on some toy data.

4.1Two maximum-margin classifiers and their support vectors: Let the data points and labels be as follows:

$$\mathbf{X}=\begin{pmatrix}1 & 2\\ -1 & 2\\ -1 & -2\end{pmatrix}\quad \mathbf{y}=\begin{pmatrix}-1\\ -1\\ +1\end{pmatrix}\quad \mathbf{X}'=\begin{pmatrix}-1 & -2\\ 1 & -2\\ -1 & -2\end{pmatrix}$$

The matrix **X** on the right incorporates the class labels; i.e., the rows are *yi* **x***i* . The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}=\begin{pmatrix}5 & 3 & -5\\ 3 & 5 & -3\\ -5 & -3 & 5\end{pmatrix}\quad \mathbf{X}'\mathbf{X}'^{\mathrm{T}}=\begin{pmatrix}5 & 3 & 5\\ 3 & 5 & 3\\ 5 & 3 & 5\end{pmatrix}$$

The dual optimisation problem is thus

$$\begin{aligned}&\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+3\alpha_1\alpha_2+5\alpha_1\alpha_3+3\alpha_2\alpha_1+5\alpha_2^2+3\alpha_2\alpha_3+5\alpha_3\alpha_1\right.\\&\qquad\left.+3\alpha_3\alpha_2+5\alpha_3^2\right)+\alpha_1+\alpha_2+\alpha_3\\&=\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+6\alpha_1\alpha_2+10\alpha_1\alpha_3+5\alpha_2^2+6\alpha_2\alpha_3+5\alpha_3^2\right)+\alpha_1+\alpha_2+\alpha_3\end{aligned}$$

subject to $\alpha 1 \geq 0, \alpha 2 \geq 0, \alpha 3 \geq 0$ and $-\alpha 1 - \alpha 2 + \alpha 3 = 0$.

Using the equality constraint we can eliminate one of the variables, say $\alpha 3$, and simplify the objective function to

$$\begin{aligned}&\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(5\alpha_1^2+6\alpha_1\alpha_2+10\alpha_1(\alpha_1+\alpha_2)+5\alpha_2^2+6\alpha_2(\alpha_1+\alpha_2)+5(\alpha_1+\alpha_2)^2\right)\\&\qquad+2\alpha_1+2\alpha_2\\&=\underset{\alpha_1,\alpha_2,\alpha_3}{\operatorname{argmax}} -\frac{1}{2}\left(20\alpha_1^2+32\alpha_1\alpha_2+16\alpha_2^2\right)+2\alpha_1+2\alpha_2\end{aligned}$$

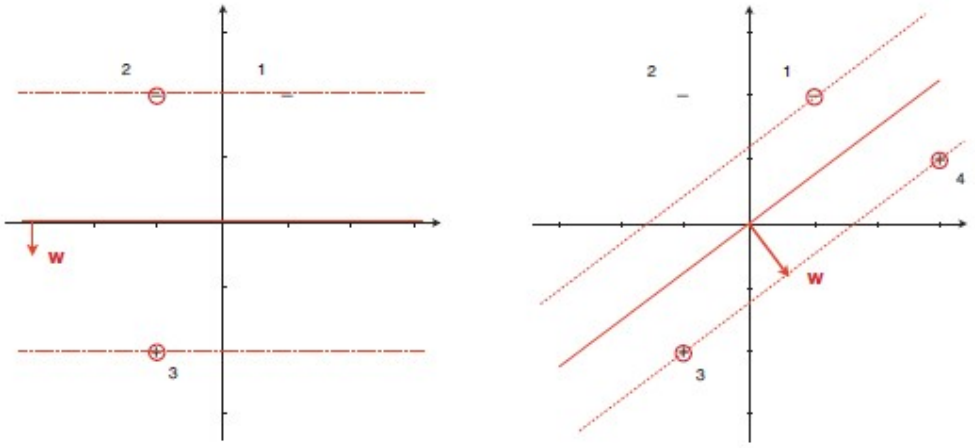


Figure 7.8. **(left)** A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2. The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary. **(right)** By adding a second positive the decision boundary is rotated to $\mathbf{w} = (3/5, -4/5)$ and the margin decreases to 1.

Algorithm 7.3: PerceptronRegression(D, T) – train a perceptron for regression.

Input : labelled training data D in homogeneous coordinates;
maximum number of training epochs T.
Output : weight vector $\mathbf{w}$ defining function approximator $\hat{y} = \mathbf{w} \cdot \mathbf{x}$.

```
w ← 0; t ← 0;
while t < T do
    for i = 1 to |D| do
        w ← w + (y_i − ŷ_i)² x_i;
    end
    t ← t + 1;
end
```

5. Soft margin SVM

1. If the data is not linearly separable, then the constraints $\mathbf{w} \cdot \mathbf{x}i - t \geq 1$ posed by the examples are not jointly satisfiable. However, there is a very elegant way of adapting the optimisation problem such that it admits a solution even in this case
2. The idea is to introduce slack variables ξi , one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary – we will call these margin errors. Thus, we change the constraints to $\mathbf{w} \cdot \mathrm{xi} - t \geq 1 - \xi i$ and add the sum of all slack variables to the objective function to be minimised, resulting in the following *soft margin* optimisation problem:

$$\mathbf{w}^*, t^*, \xi_i^* = \arg\min_{\mathbf{w}, t, \xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.

3. If we allow more margin errors we need fewer support vectors, hence C controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*. It can be seen as a form of regularisation similar to that discussed in the context of least-squares regression.
4. The Lagrange function is then as follows:

$$\begin{aligned}
\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i - t) - (1 - \xi_i)) - \sum_{i=1}^{n} \beta_i \xi_i \\
&= \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \mathbf{w} \cdot \left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) + t\left(\sum_{i=1}^{n} \alpha_i y_i\right) + \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} (C - \alpha_i - \beta_i)\xi_i \\
&= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^{n} (C - \alpha_i - \beta_i)\xi_i
\end{aligned}$$

5. For an optimal solution every partial derivative with respect to ξi should be 0, from which it follows that $C - \alpha i - \beta i = 0$ for all i , and hence the added term vanishes from the dual problem

6. Obtaining probabilities from linear classifiers:

1. A linear classifier produces scores $\hat{s}(\mathbf{x}i) = \mathbf{w}\cdot\mathbf{x}i - t$ that are thresholded at 0 in order to classify examples. Owing to the geometric nature of linear classifiers, such scores can be used to obtain the (signed) distance of $\mathbf{x}i$ from the decision boundary
2. The length of the projection of $\mathbf{x}i$ onto $\mathbf{w}$ is $||\mathbf{x}i||\cos\theta$, where θ is the angle between $\mathbf{x}i$ and $\mathbf{w}$. Since $\mathbf{w}\cdot\mathbf{x}i = ||\mathbf{w}||\,||\mathbf{x}i||\cos\theta$, we can write this length as $(\mathbf{w}\cdot\mathbf{x}i)/||\mathbf{w}||$. This gives the following signed distance:

$$d(\mathbf{x}_i) = \frac{\hat{s}(\mathbf{x}_i)}{||\mathbf{w}||} = \frac{\mathbf{w}\cdot\mathbf{x}_i - t}{||\mathbf{w}||} = \mathbf{w}'\cdot\mathbf{x}_i - t'$$

   with $\mathbf{w} = \mathbf{w}/||\mathbf{w}||$ rescaled to unit length and $t = t/||\mathbf{w}||$ the corresponding rescaled intercept. The sign of this quantity tells us which side of the decision boundary we are on: positive distances for points on the 'positive' side of the decision boundary
3. This geometric interpretation of the scores produced by linear classifiers offers an interesting possibility for turning them into probabilities, a process that was called *calibration*

Example: Suppose we now observe a point $\mathbf{x}$ with distance $d(\mathbf{x})$. We classify this point as positive if $d(\mathbf{x}) > 0$ and as negative if $d(\mathbf{x}) < 0$, but we want to attach a probability $\hat{p}(\mathbf{x}) = P(\square|d(\mathbf{x}))$ to these predictions. Using Bayes' rule we obtain

$$P(\oplus|d(\mathbf{x})) = \frac{P(d(\mathbf{x})|\oplus)P(\oplus)}{P(d(\mathbf{x})|\oplus)P(\oplus) + P(d(\mathbf{x})|\ominus)P(\ominus)} = \frac{LR}{LR + 1/clr}$$

where LR is the likelihood ratio obtained from the normal score distributions, and clr is the class ratio. We will assume for simplicity that $clr = 1$ in the derivation below. Furthermore, assume for now that $\sigma^2 = 1$ and $\overline{d}^{\oplus} = -\overline{d}^{\ominus} = 1/2$ (we will relax this in a moment). We then have

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \frac{\exp\left(-(d(\mathbf{x}) - 1/2)^2/2\right)}{\exp\left(-(d(\mathbf{x}) + 1/2)^2/2\right)}$$
$$= \exp\left(-(d(\mathbf{x}) - 1/2)^2/2 + (d(\mathbf{x}) + 1/2)^2/2\right) = \exp\left(d(\mathbf{x})\right)$$

and so

$$P(\oplus|d(\mathbf{x})) = \frac{\exp(d(\mathbf{x}))}{\exp(d(\mathbf{x})) + 1} = \frac{\exp(\mathbf{w}\cdot\mathbf{x} - t)}{\exp(\mathbf{w}\cdot\mathbf{x} - t) + 1}$$

So, in order to obtain probability estimates from a linear classifier outputting distance scores d, we convert d into a probability by means of the mapping d $\mapsto$ exp(d)/ exp(d)+1 (or, equivalently, d $\mapsto$ 1/1+exp(−d)). This S-shaped or sigmoid function is called the logistic function; it finds applications in a wide range of areas

Suppose now that $\overline{d}^{\oplus} = -\overline{d}^{\ominus}$ as before, but we do not assume anything about the magnitude of these mean distances or of $\sigma 2$. In this case we have

$$LR = \exp\left(\frac{-(d(\mathbf{x}) - \overline{d}^{\oplus})^2 + (d(\mathbf{x}) - \overline{d}^{\ominus})^2}{2\sigma^2}\right)$$
$$= \exp\left(\frac{2\overline{d}^{\oplus} d(\mathbf{x}) - \left(\overline{d}^{\oplus}\right)^2 - 2\overline{d}^{\ominus} d(\mathbf{x}) + \left(\overline{d}^{\ominus}\right)^2}{2\sigma^2}\right) = \exp\left(\gamma d(\mathbf{x})\right)$$

with $a = (\overline{d}^{\oplus} - \overline{d}^{\ominus})/\sigma^2$ a scaling factor that rescales the weight vector so that the mean distances per class are one unit of variance apart. In other words, by taking the scaling factor γ into account, we can drop our assumption that $\mathbf{w}$ is a unit vector.

If we also drop the assumption $\overline{d}^{\oplus}$ and $\overline{d}^{\ominus}$ that are symmetric around the decision boundary, then we obtain the most general form

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \exp\left(\gamma(d(\mathbf{x}) - d_0)\right)$$
$$\gamma = \frac{\overline{d}^{\oplus} - \overline{d}^{\ominus}}{\sigma^2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\sigma^2}, \quad d_0 = \frac{\overline{d}^{\oplus} + \overline{d}^{\ominus}}{2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})}{2} - t$$

d_0 has the effect of moving the decision boundary from $\mathbf{w}\cdot\mathbf{x} = t$ to $\mathbf{x} = (\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$, that is, halfway between the two class means. The logistic mapping thus becomes $d \mapsto \frac{1}{1+\exp(-\gamma(d-d_0))}$, and the effect of the two parameters is visualised in Figure 7.11.

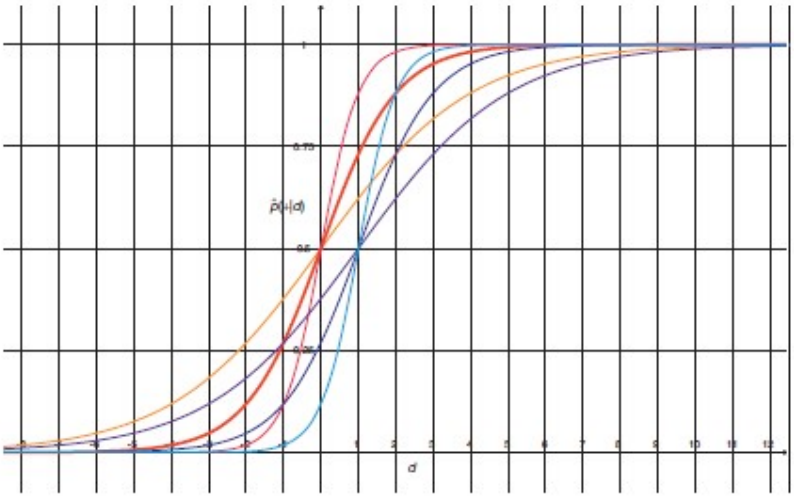


Figure 7.11. The logistic function, a useful function for mapping distances from a linear decision boundary into an estimate of the positive posterior probability. The fat red line indicates the standard logistic function $\hat{p}(d) = \frac{1}{1+\exp(-d)}$; this function can be used to obtain probability estimates if the two classes are equally prevalent and the class means are equidistant from the decision boundary and one unit of variance apart. The steeper and flatter red lines show how the function changes if the class means are 2 and 1/2 units of variance apart, respectively. The three blue lines show how these curves change if $d_0 = 1$, which means that the positives are on average further away from the decision boundary.

5.1 Example: Logistic calibration of a linear classifier:

Logistic calibration has a particularly simple form for the basic linear classifier, which has $\mathbf{w} = \boldsymbol{\mu}\square - \boldsymbol{\mu}\_$. It follows that

$$\overline{d}^{\oplus} - \overline{d}^{\ominus} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\|\mathbf{w}\|} = \frac{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|^2}{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|} = \|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|$$

and hence $\gamma = \|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|/\sigma^2$. Furthermore, $d_0 = 0$ as $(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$ is already on the decision boundary. So in this case logistic calibration does not move the

decision boundary, and only adjusts the steepness of the sigmoid according to the separation of the classes.

7. Going beyond linearity with kernel methods:

It is customary to call the transformed space the *feature space* and the original space the *input space*. The approach thus appears to be to transform the training data to feature space and learn a model there. In order to classify new data we transform that to feature space as well and apply the model.

Process: Take the perceptron algorithm in dual form. The algorithm is a simple counting algorithm – the only operation that is somewhat involved is testing whether example $\mathbf{x}i$ is correctly classified by evaluating $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot$

$\mathbf{x}_j$. The key component of this calculation is the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$. Assuming bivariate examples $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}_j = (x_j, y_j)$ for notational simplicity, the dot product can be written as $\mathbf{x}_i \cdot \mathbf{x}_j = x_i x_j + y_i y_j$. The corresponding instances in the quadratic feature space are (x_i^2, y_i^2) and $\left(x_j^2, y_j^2\right)$, and their dot product is

$$(x_i^2, y_i^2) \cdot \left(x_j^2, y_j^2\right) = x_i^2 x_j^2 + y_i^2 y_j^2$$

This is almost equal to

$$(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = (x_i x_j + y_i y_j)^2 = (x_i x_j)^2 + (y_i y_j)^2 + 2 x_i x_j y_i y_j$$

but not quite because of the third term of cross-products. We can capture this term by extending the feature vector with a third feature $\sqrt{2}xy$. This gives the following feature space:

$$\phi(\mathbf{x}_i) = \left(x_i^2, y_i^2, \sqrt{2} x_i y_i\right) \qquad \phi(\mathbf{x}_j) = \left(x_j^2, y_j^2, \sqrt{2} x_j y_j\right)$$
$$\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = x_i^2 x_j^2 + y_i^2 y_j^2 + 2 x_i x_j y_i y_j = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

We now define $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, and replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ in the dual perceptron algorithm to obtain the *kernel perceptron* (Algorithm 7.4), which is able to learn the kind of non-linear decision boundaries illustrated in Example 7.8.

The introduction of kernels opens up a whole range of possibilities. Clearly we can define a polynomial kernel of any degree p as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$. This transforms

a *d*-dimensional input space into a high-dimensional feature space, such that each new feature is a product of *p* terms (possibly repeated). If we include a constant, say $\kappa(\mathbf{x}i, \mathbf{x}j) = (\mathbf{x}i \cdot \mathbf{x}j + 1)p$, we would get all lower-order terms as well. So, for example, in a bivariate input space and setting $p = 2$ the resulting feature space is

$$\phi(\mathbf{x}) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right)$$

with linear as well as quadratic features. But we are not restricted to polynomial kernels. An often-used kernel is the *Gaussian kernel*, defined as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right)$$

where σ is a parameter known as the *bandwidth*. To understand the Gaussian kernel a bit better, notice that $\kappa(\mathbf{x},\mathbf{x}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}) = ||\varphi(\mathbf{x})||2$ for any kernel obeying a number of standard properties referred to as 'positive semi-definiteness'.

Algorithm 7.4: KernelPerceptron(D, κ) – perceptron training algorithm using a kernel.

Input : labelled training data D in homogeneous coordinates;
kernel function κ.
Output : coefficients α_i defining non-linear decision boundary.

```
1  α_i ← 0 for 1 ≤ i ≤ |D|;
2  converged←false;
3  while converged = false do
4      converged←true;
5      for i = 1 to |D| do
6          if y_i Σ_{j=1}^{|D|} α_j y_j κ(x_i, x_j) ≤ 0 then
7              α_i ← α_i + 1;
8              converged←false;
9          end
10     end
11 end
```

Kernel methods are best known in combination with support vector machines. Notice that the soft margin optimisation problem is defined in terms of dot products between training instances and hence the 'kernel trick' can be applied:

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \le \alpha_i \le C \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$