Introduction to Machine Learning (for AI)

Machine Learning Methods

Dr. Matias Valdenegro & Dr. Andreea Sburlea

November 23, 2022

Today's Lecture

- Today we are covering a bunch of ML methods for classification and regression.
- Andreea will cover Non-Linear Methods, Decision Trees and Random Forests, and LDA.
- Matias will cover Linear Methods and SVMs.
- We will cover SVMs tomorrow before the Statistical Learning Theory lecture.

Outline

Introduction

- Linear regression is the simplest regression model.
- As reminder, regression means that the output variable is continuous.

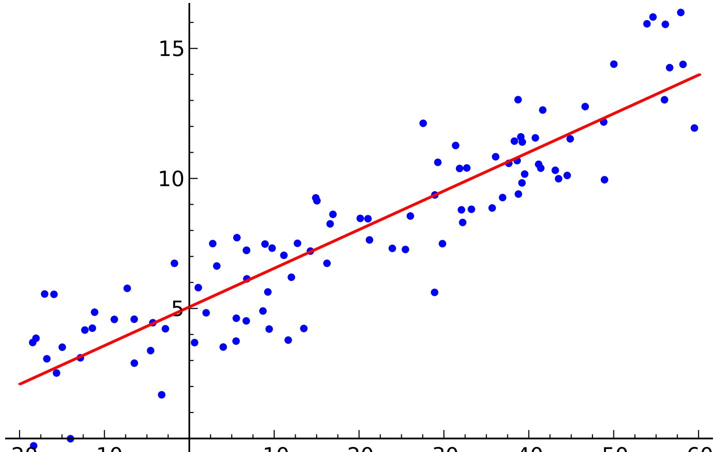
Linear Regression Model

The basic linear regression model is:

$$f(x) = \sum_i x_i w_i + b = \mathbf{x} \cdot \mathbf{w} + b \tag{1}$$

Here the parameter/weight vector **w** has the same length as the input features **x**, and the bias b is a scalar. The total number of parameters/weights is $P + 1$, where $P = \mathrm{dim}(x)$

Linear Regression Model



From https://en.wikipedia.org/wiki/File:Linear_regression.svg

Matrix Formulation

The previous equations can be re-written in a matrix form. First we define the input features X with an additional column of 1's concatenated/appended at the end:

$$X = \begin{pmatrix} \mathbf{x} & \mathbf{1} \end{pmatrix} \tag{2}$$

And the weight vector W is now defined as:

$$W = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} \tag{3}$$

Once these small transformations are done, we can re-write the linear regression model as:

$$f(x) = XW = \begin{pmatrix} \mathbf{x} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} = \mathbf{x} \cdot \mathbf{w} + b \tag{4}$$

Closed Form Solution

Linear regression models are trained using the mean squared error loss (with n data points):

$$L(y, \hat{y}) = n^{-1} \sum_i (WX_i - y_i)^2 \tag{5}$$

Because $\hat{y}_i = WX_i$. Then if we condense the whole training set as variables **y** and **X** (including the column of 1's), then we can rewrite the loss as:

$$L = (\mathbf{y} - W\mathbf{X})^T(\mathbf{y} - W\mathbf{X}) \tag{6}$$

To find the value of W that minimizes this loss, we can use the derivative of L and solve for this to be zero:

$$\frac{\partial L}{\partial W} = \frac{\partial}{\partial W}(\mathbf{y} - W\mathbf{X})^T(\mathbf{y} - W\mathbf{X}) = 0 \tag{7}$$

Closed Form Solution

After some algebraic manipulations, this has a closed form solution:

$$W = n^{-1}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{8}$$

Here $\mathbf{X}$ is a $n \times (d+1)$ matrix, and $\mathbf{y}$ is a $n \times 1$ vector, where n is the number of training samples, and d is their dimensionality.

Then we can note that $\mathbf{X}^T\mathbf{X}$ is a $(d+1) \times (d+1)$ matrix, while $\mathbf{X}^T\mathbf{y}$ is a $(d+1) \times 1$ vector, so the whole operation $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ produces a $(d+1) \times 1$ result.

Closed Form Solution

This only works if the rows of $\mathbf{X}$ are *linearly independent*. It only makes sense to do this if the matrix inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ is tractable, and if the matrix multiplication $\mathbf{X}^T\mathbf{X}$ is computationally sane.

Linearly independence can be broken easily, for example, if two data samples are duplicated, or labels are inconsistent, or the data is far from being represented as a line/hyper-plane.

Modeling Assumptions

Residuals

The actual model considers errors or residuals ϵ:

$$f(x_i) = WX_i + \epsilon_i = y_i \tag{9}$$

This is because the y_i's can have measurement or other kinds of errors. These residuals are zero only if the model fits the data perfectly (zero loss).

Data Must Not Have Dependencies

As mentioned, the analytical solution assumes that the rows of $\mathbf{X}$ are *linearly independent*. This means that no variable x_i in the training set is linearly related to another variable x_j, and in simple words, variables in the training set are not correlated.

Now you see the importance of decorrelating your inputs?

Modeling Assumptions

Independence of Residuals

The residuals ϵ_i are assumed to be independent and not related to each other or to the input variables. This means that the errors do not depend on the input.

Linearity

In this model, the input features x_i are treated as fixed values, and the model is only linear with respect of its parameters. Input features can be transformed to produce other features and the model stays being linear, since the training data is treated as a constant, and optimization only happens in the parameters W.

Modeling Assumptions

Constant Variance

The variance of the residuals does not depend in the inputs or the outputs of the model. This is called Homoscedacity. The opposite (Heteroscedacity) is when the variance of the output or the errors can vary with the input or output of the model, for example, if larger outputs have larger variance then smaller outputs.

Modeling Assumptions

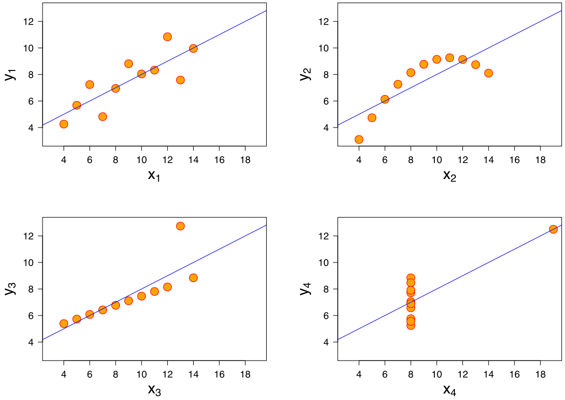


Figure: These are samples from Anscombe's Quartet, indicating training sets that have different data points, but the same linear regression line, indicating that a linear model should be used with care. Source from `https://en.wikipedia.org/wiki/Anscombe's_quartet`

Solution with Gradient Descent

In case that the analytic solution is intractable, gradient descent can be used as substitute. For this we need to compute the gradient of the loss with respect to parameters:

$$\frac{\partial L}{\partial W} = n^{-1} \frac{\partial}{\partial W} (\mathbf{y} - W\mathbf{X})^T (\mathbf{y} - W\mathbf{X}) \tag{10}$$

This has a well known closed form:

$$\frac{\partial L}{\partial W} = n^{-1} \mathbf{X}^T (W\mathbf{X} - \mathbf{y}) \tag{11}$$

Solution with Gradient Descent

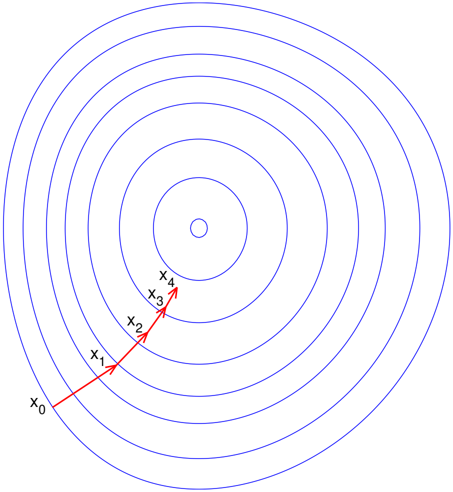
Which produces a vector $(d+1) \times 1$, and then parameters can be updated using gradient descent:

$$W_{m+1} = W_m - \alpha n^{-1} \mathbf{X}^T (W\mathbf{X} - \mathbf{y}) \tag{12}$$

Where W is initialized with a random vector (in a small range), and α is the learning range that has to be tuned manually.

This is iterated until M iterations have happened, and the loss value is monitored for convergence (loss is not decreasing and approximately constant).

Solution with Gradient Descent



From `https://en.wikipedia.org/wiki/Gradient_descent`

Multi-variable Linear Regression

What if the labels y are not scalars, but vectors of dimension m? We can still perform linear regression, but now the model outputs a vector instead of a scalar. This can be seen as performing m individual linear regression problems:

$$f(x) = [\mathbf{x}_j \cdot \mathbf{w}_j + b_j]_j = WX + b \tag{13}$$

Where now b is a $m \times 1$ vector instead of a scalar, and W is a $d \times m$ matrix instead of a vector.

In general all previous equations hold, but now there is an added dimension m, and in many cases multi-dimensional matrix multiplications are required. We will see more details later in multi-class logistic regression.

Interpretability of Weights

For linear models like LR, the weights/parameters can sometimes be interpreted if:

- Input features are normalized/scaled to be in the exact same range.
- For the case of multi-variable LR, then the output features should also be normalized/scaled to be at the same range.

The interpretation in this case is that each weight indicate the (loose) feature importance associated to each weight. The bias does not have a particular interpretation (other than the y intercept). For numerical features, an increase of that feature by one unit increases the value of output y by a factor of that feature's weight.

Polynomial Regression

A Polynomial of degree p is given by:

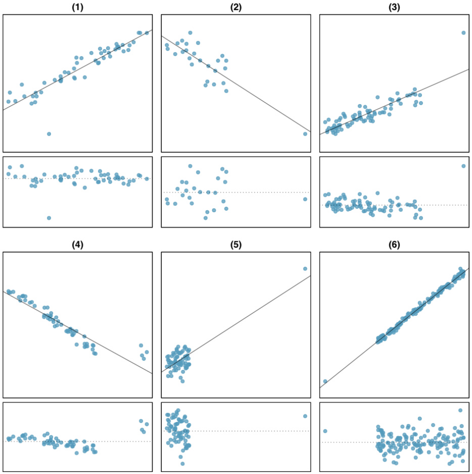
$$f(x) = \sum_{i=0}^{p} w_i x^i = w_0 + w_1 x + w_2 x^2 + w_3 w^3 + \ldots + w_p x^p \tag{14}$$

This is a linear model on the features $[1, x, x^2, x^3, \ldots, x^p]$.

This is called polynomial regression, and it is a way to make regression non-linear, by modifying the input features. You choose a value of p (using cross validation), transform each feature x_i into p polynomial values, and train a linear regression model on the new features.

This method increases the feature space dimensionality by a factor of p.

Outliers in Linear Regression



Robust Linear Regression

Linear Regression is overall not robust to outliers. There are many alternatives.

- There are many robust linear regression methods, generally making assumptions about the output y, for example that it follows a student's t-distribution or other heavy tailed distribution.
- The simple LR algorithm assumes that the output is Gaussian distributed, making it not robust to outliers.
- RANSAC (Random Sampling Consensus) can be used to fit multiple LR models and detect which ones are outliers.
- Exploratory data analysis can be used to identify and remove outliers.

Outline

Classification Reminder

- Classification is when the output variable and labels are discrete.
- In Classification, the model should separate or segregate the data points, while in Regression the model usually tightly fits the data points.
- Different losses are used for these tasks, but also they need changes in the model equations, mostly related on how a discrete output is drawn from continuous outputs produced by a model.

Probabilistic Classifiers

Most classifiers output a probability vector p of length C. The class integer class index c can be recovered by:

$$c = \arg\max_i p_i \tag{15}$$

Note that for C classes, their indices go from 0 to $C - 1$. For binary classification, only a single probability is required:

$$f(x) = P(y = 1) = 1 - P(y = 0) \tag{16}$$

In this case, the classifier outputs the probability of class 1 (usually the positive class), while the probability for class 0 (the negative class) can be recovered by subtracting with one.

Basic Linear Binary Classifier

The basic model is:

$$f(x) = \operatorname{sgn}(\sum_i w_i x_i + b) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \tag{17}$$

This model is called perceptron. The function sgn is the sign function defined by:

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{18}$$

Also the step function can be used:

$$\operatorname{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{19}$$

Perceptron Learning Rule

To learn the weights, this simple rule is applied iteratively:

1. Initialize all weights w_i with random values in a small range (like $[-0.5, 0.5]$).
2. For each data point (x_j, y_j) in the training set.
    2.1 Compute current output $\hat{y}_j = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$
    2.2 Update each weight w_i:

$$w_i(n+1) = w_i(n) + \alpha(\hat{y}_j - y_j)x_{ji} \tag{20}$$

3. Repeat step 1 up to n times, where n is the number of total iterations.

This assumes a representation where the bias is integrated into the weights (and a column of ones is added to the input features), and α is a learning rate.

Geometric Interpretation

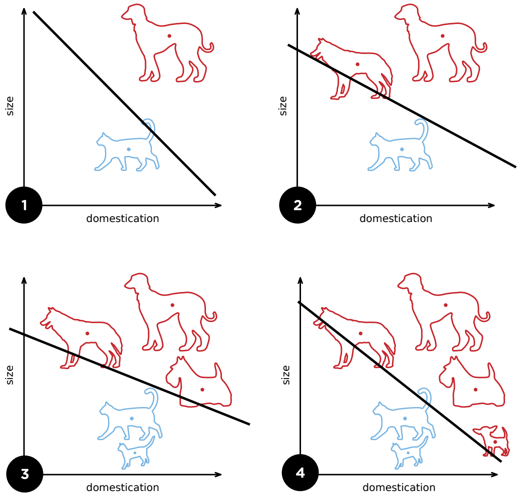


Figure from `https://en.wikipedia.org/wiki/Perceptron#/media/File:Perceptron_example.svg`

Decision Boundary

The equation $\mathbf{w} \cdot \mathbf{x} + b = 0$ defines a hyper-plane in the feature space x.

This hyper-plane is called the decision boundary. For a linear classifier is a line in 2D, a plane in 3D, and a hyper-plane in more than three dimensions. For a non-linear classifier, the decision boundary is a curve in feature space.

Usually the points close to the boundary are the most difficulty to classify, as due to noise and position of the boundary, they could fall in either side of the boundary.

Each side of the boundary defines a region for each class.

Decision Boundary

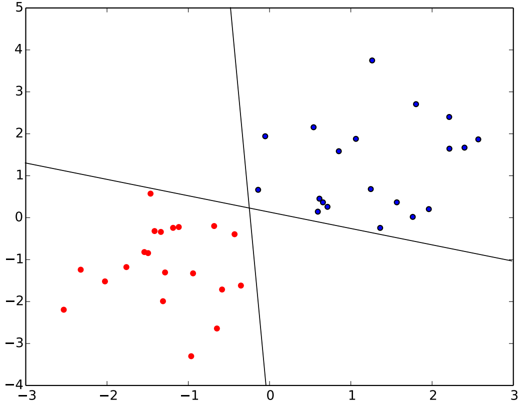


Figure: In this example there are multiple hyper-planes that would perfectly separate all data points into two classes.

Why is the Perceptron not used?

- The model is not differentiable due to the use of non-continuous activation functions.
- The model only produces binary outputs, with no probabilistic interpretations.
- The training process is quite slow (it processes one sample at a time) and cumbersome to tune.
- The learning rule only converges if the data is linearly separable.
- The decision boundary is not unique, and can be different in each training run.

Logistic Regression Model

Logistic Regression is *a classification model* (this is not a mistake, the name is misleading). The basic model is:

$$f(x) = \sigma(\sum_i w_i x_i + b) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{21}$$

Where the function $\sigma(x)$ is called the logistic or sigmoid function, given by:

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \tag{22}$$

This is basically linear regression with the logistic function applied to its output.

Probabilistic Interpretation

Logistic regression outputs a continuous value in the range $[0, 1]$, which is usually interpreted as:

$$P(y = 1|x) = f(x) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{23}$$

This means, the output of logistic regression is the probability that $y = 1$, meaning it is the probability of the positive class, given the input x.

This is advantageous since now the model outputs a probability that can represent uncertainty. This also requires some changes for training. Note that for class 0:

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{24}$$

Training Logistic Regression

Logistic regression uses the binary cross-entropy loss, which can be used to estimate the maximum likelihood solution given the data.

Binary Cross-Entropy

Used for binary classification problems with labels $y_i \in \{0, 1\}$

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Where now $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$.

Logistic Regression Concepts

Logit

Logits are the input to the logistic/sigmoid function:

$$l = \mathbf{w} \cdot \mathbf{x} + b \tag{25}$$
$$\hat{y} = \sigma(l) \tag{26}$$

Here l is a logit. Logits range in the real numbers $(\mathbb{R})$, while the output of the logistic/sigmoid function is $[0, 1]$. The expanded range is useful in some applications, for example if you want to do regression of the logits.

Training with Gradient Descent

Using the matrix representation, the gradient of the cross-entropy loss has a well known closed form:

$$\frac{\partial L}{\partial W} = n^{-1}\mathbf{X}^T(\sigma(W\mathbf{X}) - \mathbf{y}) \tag{27}$$

Which produces a vector $(d+1) \times 1$, and then parameters can be updated using gradient descent:

$$W_{n+1} = W_n - \alpha n^{-1}\mathbf{X}^T(\sigma(W\mathbf{X}) - \mathbf{y}) \tag{28}$$

Where W is initialized with a random vector (in a small range), and α is the learning range that has to be tuned manually. Note that the gradient is very similar to the one in linear regression, except for the application of logistic/sigmoid function $\sigma(x)$.

Multi-class Logistic Regression

The current logistic regression formulation only works for binary classification, and there are some strategies to extend to a multi-class setting.

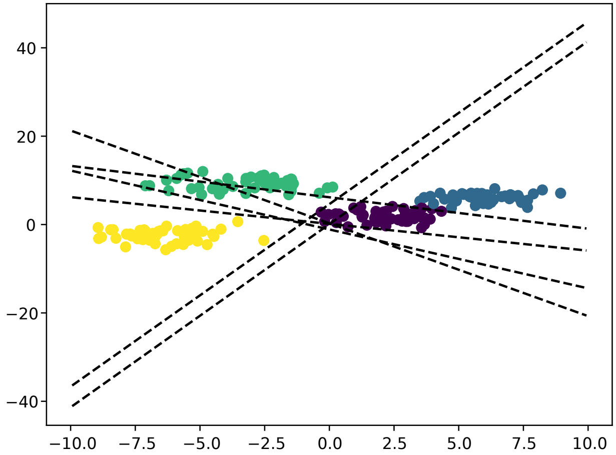
One vs One

For each pair of classes, train one classifier. To decide the output class at inference, select the class with the most votes. This way each classifier pair works as a vote for one class. Requires $0.5c(c-1)$ classifiers.

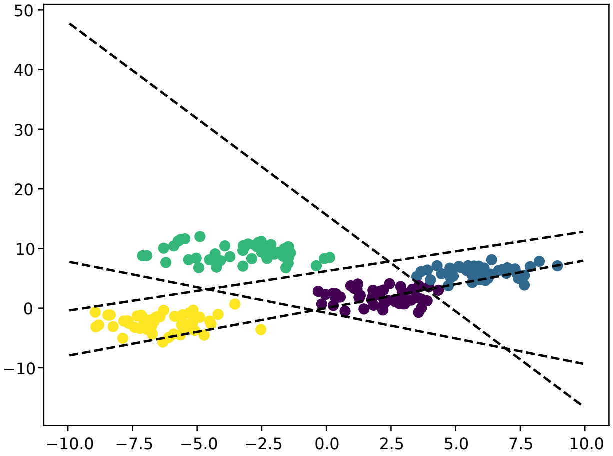
One vs All

For each class, train a classifier for that class vs all other data points, and at inference, make a prediction with all classifiers and select the class with highest probability. Requires only c classifiers.

One vs One



One vs Rest



Multinomial Logistic Regression

A simpler way to formulate a multi-class logistic regression model is with:

$$f(\mathbf{x}) = \text{softmax}(\mathbf{wx} + \mathbf{b}) \tag{29}$$

Where now $\mathbf{W}$ is a $c \times d$ matrix and b is a $c \times 1$ vector, where c is the number of classes. This model now outputs a probability vector instead of a single probability.

This formulation is equivalent to a one-layer neural network.

Softmax Function

The softmax is a function $s : R^n \rightarrow [0, 1]^n$ defined as:

$$\text{softmax}(\mathbf{x}) = \left[\frac{e^{x_i}}{\sum_j e^{x_j}}\right]_i = \left[\frac{e^{x_0}}{\sum_j e^{x_j}}, \frac{e^{x_1}}{\sum_j e^{x_j}}, \ldots, \frac{e^{x_{n-1}}}{\sum_j e^{x_j}}\right] \tag{30}$$

This function transforms a vector of logits into a discrete probability distribution, where the elements of the output vector sum to 1.

It is usually used to transform the output of a linear classifier (producing logits) into probabilities.

Training Multinomial LR

This model is trained using the categorical cross-entropy loss.

Categorical Cross-Entropy

For this loss, labels y^c should be one-hot encoded. Used for multi-class classification problems, where the model predictions are $\hat{y}_i{}^c$ are class probabilities that sum to 1.

$$L(y, \hat{y}) = -\sum_i \sum_c y_i^c \log(\hat{y}_i{}^c)$$

The gradient has the same form as in binary LR:

$$\frac{\partial L}{\partial W} = n^{-1}\mathbf{X}^T(\text{softmax}(W\mathbf{X}) - \mathbf{y}) \tag{31}$$

Multi-Label Logistic Classification

The multi-label setting is where you have multiple classes but more than one class is possible at the same time.

This can also be modeled using logistic regression, where the model is:

$$f(\mathbf{x}) = \sigma(\mathbf{wx} + \mathbf{b}) \tag{32}$$

The difference is the use of the logistic/sigmoid activation for each class, as **W** is a $d \times c$ matrix and **b** is a $d \times 1$ vector.

This model is trained using the binary-cross entropy loss, but applied for each class separately:

$$L(y, \hat{y}) = -\sum_{c} \sum_{i} y_i^c \log(\hat{y}_i^c) + (1 - y_i^c) \log(1 - \hat{y}_i^c)$$

This means the target vector is not one-hot encoded, but set to 1 if the class is present, and to 0 otherwise.

Multi-Label Logistic Classification

Predictions can be made normally, but classes need to be decided in a slightly different way:

$$\hat{y} = \sigma(\mathbf{wx} + \mathbf{b}) \tag{33}$$

$$\text{class}(\hat{y}^c) = \begin{cases} \text{Class c is present} & \text{if } \hat{y}^c < T^c \\ \text{Class c is absent} & \text{if } \hat{y}^c \geq T^c \end{cases} \tag{34}$$

Where T^c is a threshold that can be tuned for each class, for example, by using an ROC curve.

Outline

Motivation

- One large theoretical issue with linear models (for classification) is that the separating hyper-plane is not unique.
- Other issues is that the decision boundary is a line or hyperplane,
- A Support Vector Machine (SVM) is a linear model that has some theoretical advantages over other models, such as having a unique solution, can be easily transformed into a non-linear model, and integrates interpretable regularization.

Motivation - Hyperplanes

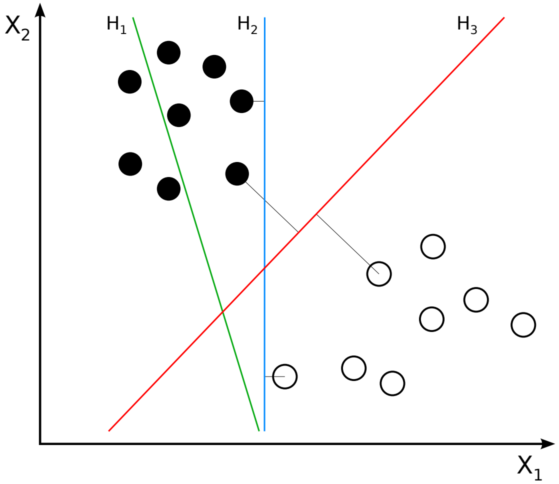


Figure from `https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_(SVG).svg`

Maximum Margin Formulation

People came up with a simple idea to solve this issue.

What if instead of a hyper-plane separating the data, we learn a separating hyper-plane including a margin parallel to the hyper-plane, and then try to find the plane that has the biggest separation between the two classes (maximum margin).

The limits of the margin would be given by the data points, meaning that the hyper-plane not only has to separate both classes, but also *touch* the data points closest to the hyper-plane.

This hyper-plane then would be unique, which solves the theoretical issue.

Maximum Margin Formulation

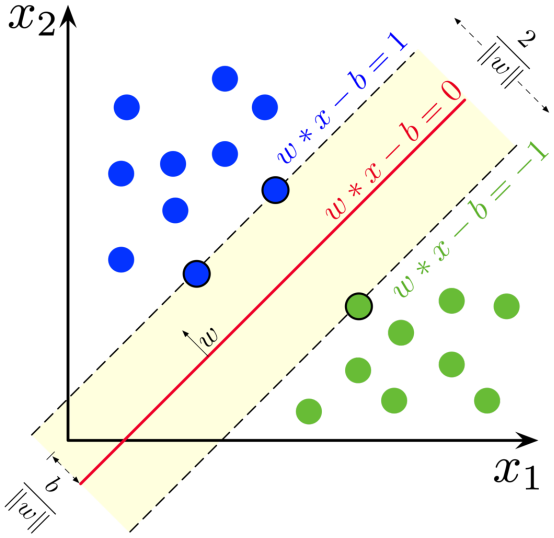


Figure from `https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:SVM_margin.png`

Hard Maximum Margin Formulation

The hard margin formulation applies when the data is linearly separable, using two parallel hyperplanes, defined by:

$$\mathbf{wx} + b = 1 \qquad \text{Positive Class} \tag{35}$$

$$\mathbf{wx} + b = -1 \qquad \text{Negative Class} \tag{36}$$

Anything above the hyperplane $\mathbf{wx} + b = 1$ is classified as the positive class, and anything below the hyperplane $\mathbf{wx} + b = -1$ is classified as the negative class.

The distance between these hyper-planes is $\frac{2}{||\mathbf{w}||}$, so in order to maximize the margin, we would like to minimize $||\mathbf{w}||$.

Hard Maximum Margin Formulation

To consider the labels y_i and constrain points to not fall inside the margin, we can use the following constraints.

$$\begin{aligned} \mathbf{w}\mathbf{x}_i + b \geq & \quad 1 \qquad \text{if } y_i = 1 \qquad (37) \\ \mathbf{w}\mathbf{x}_i + b \leq & \ -1 \qquad \text{if } y_i = -1 \qquad (38) \end{aligned}$$

These constraints can be compacted into:

$$y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 \qquad (39)$$

From where the following optimization problem can be derived:

$$\text{Minimize } ||\mathbf{w}|| \text{ subject to } y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 \, \forall i \in [1, n] \qquad (40)$$

Hard Maximum Margin Formulation

Once the SVM is learned, predictions can be made with:

$$f(x) = \text{sign}(\mathbf{w}\mathbf{x}_i + b) \tag{41}$$

Note that in this formulation, the labels are 1 for the positive class, and -1 for the negative class.

Soft Maximum Margin Formulation

In the case that the data is not linearly separable, the margin can be made to be *soft*, by relaxing the constraints using positive slack variables ξ_i:

$$y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 - \xi_i \tag{42}$$

$$\xi_i \geq 0 \tag{43}$$

The idea is to minimize the total of these slack variables, which leads to the following optimization problem.

$$\text{minimize } ||\mathbf{w}|| + C\sum_i \xi_i \tag{44}$$

$$\text{subject to } y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 - \xi_i \tag{45}$$

$$\xi_i \geq 0 \tag{46}$$

Soft Maximum Margin Formulation

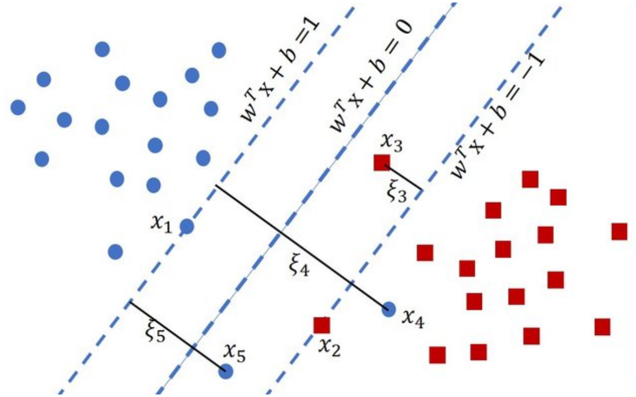
The constraints can be integrated into a single loss function:

$$L = ||\mathbf{w}|| + C \sum_i \max(0, 1 - y_i(\mathbf{w}\mathbf{x}_i + b)) \tag{47}$$

The part $\max(0, 1 - y_i(\mathbf{w}\mathbf{x}_i + b))$ is called the hinge loss, and controls the constraints implicitly.

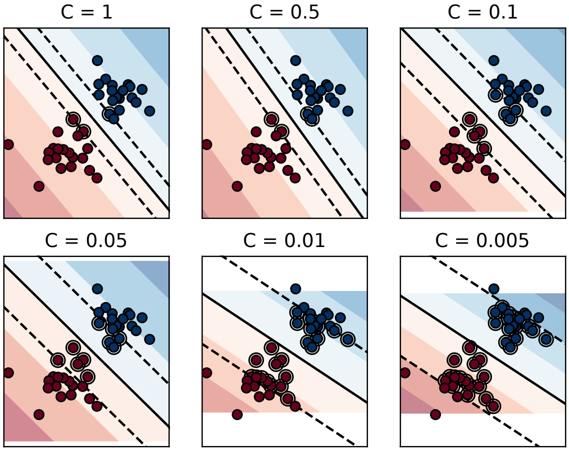
The coefficient C works as a regularization coefficient, where it controls the weight associated to the hinge loss, and varying it controls the *softness* of the margin (How many misclassifications are allowed).

Soft Maximum Margin Formulation



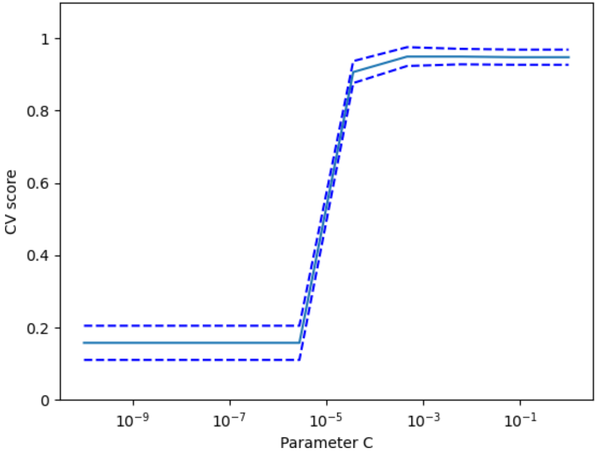
Soft Maximum Margin Formulation

Effect of Varying C



Soft Maximum Margin Formulation

Effect of Varying C on Digits Dataset



SVM Concepts

Margin

It is the area between the two separating hyperplanes, and ideally it should not contain any data points (hard margin). For a soft margin it can contain data points, depending on the value of C.

Support Vector

The points that lie in the border of the margin are called support vectors, since they are the ones that define the geometry of the margin and the values of the weights $\mathbf{w}$. Data points beyond the margin do not really contribute to training.

Training SVMs

Training SVMs is a bit different than other algorithms that we have covered.

Hard Margin

This is quadratic programming problem, and a quadratic solver needs to be used. The loss is convex so there is always a unique solution.

Soft Margin

The slack variable (ξ) formulation is also a quadratic problem, trainable with a quadratic solver. The hinge loss formulation is trainable using gradient descent.

Non-Linear Classification - Kernels

There is another advantage commonly associated with SVMs, that they can also perform non-linear classification using what is called a kernel.

The idea of the Kernel is that it can transform the SVM from a linear classifier, into a non-linear classifier. This is done by changing the shape of the decision boundary, and thus producing a non-linear margin.

The *kernel trick* is applied for this purpose, the formulation of the SVM does not change, only a small part is replaced with a kernel function. We will see this with a concrete example.

Kernel - Example

Let's assume that we have two vectors $\phi(x)$ and $phi(y)$ given by:

$$\phi(x) = (x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_{n-1}x_{n-2}, \ldots, \tag{48}$$

$$\sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \ldots, \sqrt{2}cx_n, \ldots, \sqrt{2}cx_1, c) \tag{49}$$

Then we compute the following dot product (inner product):

$$\phi(x) \cdot \phi(y) = \sum_{i=1}^{n} (x_i^2)(y_i^2) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (\sqrt{2}x_i x_j)(\sqrt{2}y_i y_j) \tag{50}$$

$$+ \sum_{i=1}^{n} (\sqrt{2}cx_i)(\sqrt{2}cy_i) + c^2 \tag{51}$$

Kernel - Example

This can be simplified with algebra to the following:

$$K(\mathbf{x}, \mathbf{y}) = \phi(x) \cdot \phi(y) = \left(\sum_{i=1}^{n} x_i y_i + c \right)^2 \qquad (52)$$

This means that the dot product $\phi(x) \cdot \phi(y)$ for $\phi(x)$ has a closed form that is easily computable.
The essence of the Kernel Trick is to use a easily computable function that corresponds to a dot product of the features in a highly dimensional space.

Kernel - Example

Now, if we go back to the general form of a linear model:

$$f(x) = \underbrace{\mathbf{w} \cdot \mathbf{x}}_{\text{Dot Product}} + b \tag{53}$$

There is a dot product between the input features and the weight vector. In a Kernelized linear model, we can do the following:

$$f(x) = \phi(\mathbf{w}) \cdot \phi(\mathbf{x}) + b \tag{54}$$

For the vectors we previously defined, we can replace the dot product $\phi(\mathbf{w})\phi(\mathbf{x})$ with the closed form solution:

$$f(x) = \left(\sum_{i=1}^{n} x_i w_i + c \right)^2 + b \tag{55}$$

Kernel - Example

This process is called Kernelization or the Kernel Trick. And it works because in Kernel space, this is still a linear model.

What this means is that the model is now non-linear in w space, but it is still linear in $\phi(w)$ space.

The Kernel Trick is a bit difficult to understand in an abstract way, but it is widely used as a way to make a linear classifier non-linear.

The trick for understanding is that the kernel projects the data into a highly dimensional space ($3n$ for the example here), and it is more likely to be linearly separable in this new space.

Kernel Trick - Kernel Definition

A Kernel function is defined as:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \tag{56}$$

For some function $\phi(x)$. For the kernel trick, we are only interested in the function $K(x, y)$.

The Kernel Trick is to replace every dot product in the linear model by $K(x, y)$.

This works because the data is projected/transformed into a higher dimensional space (where it is more likely to be linearly separable), but this is done *without* having to compute the feature transformation, only the dot product with the Kernel function.

Kernel Trick

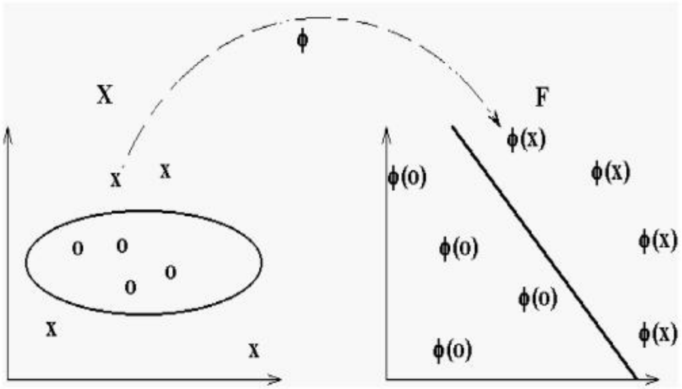


Figure from `https://en.wikipedia.org/wiki/Polynomial_kernel#/media/File:Svm_8_polinomial.JPG`

Kernel Trick

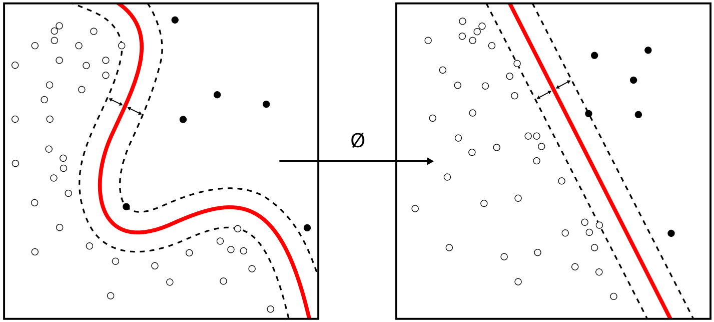
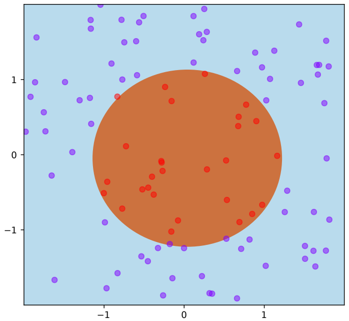


Figure: Left is the original feature space, and on the right is the kernel transformed feature space, where the features are more likely to be linearly separable.

Figure from `https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_Machine.svg`

Kernel Trick



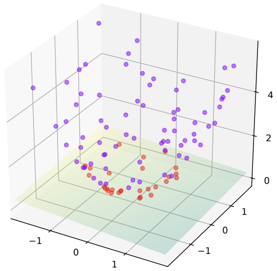


Figure: Left is the data in the original space, while on the right the data has been transformed into a 3D space, where it is linearly separable, unlike in the original space.

Kernel Functions

Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^d \tag{57}$$

Gaussian Kernel or Radial Basis Function (RBF)

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{(||\mathbf{x} - \mathbf{y}||)^2}{2\sigma^2}\right) \tag{58}$$

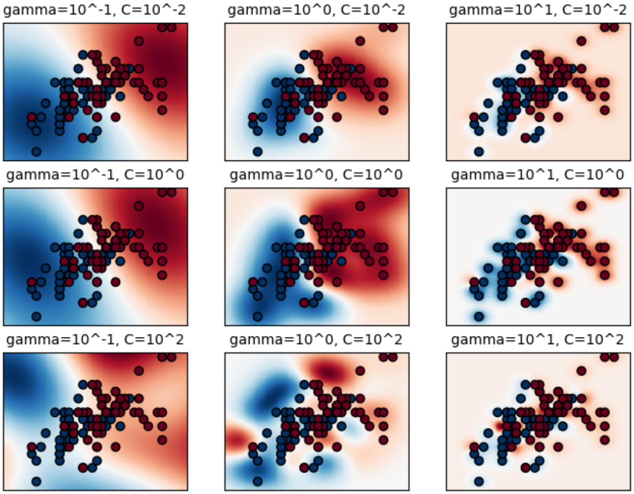
Laplace RBF

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{y}||}{2\sigma^2}\right) \tag{59}$$

Laplace RBF

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x} \cdot \mathbf{y} + c) \tag{60}$$

Parameters of RBF Kernel



Multi-Class SVMs

Current SVM formulations are only for binary classification. They can be transformed into multi-class classifiers by applying the two strategies we covered before:

One vs One
: Train $0.5C(C-1)$ classifiers, one for each pair of classes.

One vs All / Rest
: Train C classifiers, one class versus the rest. This is a good default option to use.

Unfortunately there are no other formulations to allow multi-class classification.

Support Vector Regression

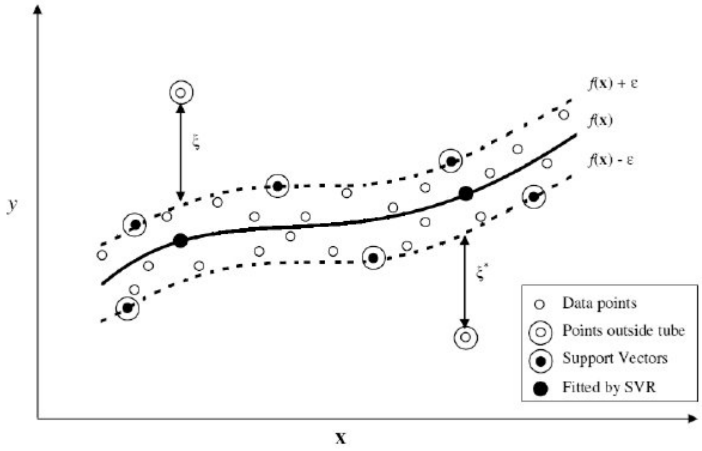
The idea of an SVM can also be extended for regression problems, this is called SVR.

The formulation is to have a tube around the linear regression line, where all points that are at distance ϵ from the line receive no penalty (zero loss), and points outside of this tube do receive a standard mean absolute error loss.

The value of ϵ is a tunable hyper-parameter that trades off acceptable errors. The formulation is:

$$\text{Minimize } ||\mathbf{w}|| \text{ subject to } |\mathbf{w}\mathbf{x}_i + b - y_i| \leq \epsilon \, \forall i \in [1, n] \tag{61}$$

Support Vector Regression



Questions to Think About

1. What is the basic concept underpinning SVMs?
2. How do ML Classification methods relate to Linear Separability?
3. Explain the concept of the Kernel Trick and its relationship with Kernel Functions.
4. Why is k-NN Non-Linear?
5. Why does an Ensemble improve performance?
6. How to transform a binary classifier into a multi-class one?

Take Home Messages

- We covered a lot of methods, there are important underpinning concepts like linear models, linear separability, kernel tricks, etc.
- Linear methods are easy to understand and implement, and generally have good performance.
- Kernel methods allow to transform Linear methods into non-linear ones, with simple tricks.
- Non-linear methods can have better performance but they are difficult to understand and implement.
- In the end performance depends on the features and if they are linearly separable (classification) or fit a line (regression).

Questions?