Introduction to Machine Learning
Summary

Philip Hartout

March 9, 2020

Contents

1 Linear Regression

Objective, approximate:

$$\begin{aligned} f(x) &= w_1 x_1 + \ldots + w_d x_d + w_0 \\ &= \sum_{i=1}^{d} w_i x_i + w_0 \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

$\forall \mathbf{x}, \mathbf{w} \in \mathbb{R}^d$. This expression can be further compressed to the homogeneous representation where $\forall \tilde{\mathbf{x}}, \tilde{\mathbf{w}} \in \mathbb{R}^{d+1}$, i.e. $\tilde{x}_{d+1} = 1$. We have w.l.o.g.:

$$f(x) = \mathbf{w}^T \mathbf{x}$$

Quantify errors using residuals:

$$\begin{aligned} r_i &= y_i - f(x_i) \\ &= y_i - \mathbf{w}^T \mathbf{x_i} \end{aligned}$$

We can use squared residuals and sum over all residuals to get the cost:

$$\hat{R}(w) = \sum_{i=1}^{n} r_i^2 \tag{1}$$

$$= \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x_i})^2 \tag{2}$$

Optimization objective to find optimal weight vector $\mathbf{w}$ with least squares is the following:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(y_i - w^T x_i\right)^2$$

1.1 Closed form solution

This can be solved in closed form:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where:

$$X = \begin{bmatrix} X_{1,1} & \ldots & X_{1,d} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \ldots & X_{n,d} \end{bmatrix} \text{ and } y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

1.2 Optimization

1.2.1 Requirements

Requires a convex objective function.

Definition 1.1 (Convexity)**.** *A function is convex iff* $\forall \mathbf{x}, \mathbf{x}', \lambda \in [0,1]$ *it holds that* $f(\lambda x + (1 - \lambda)x') \leq \lambda f(x) + (1 - \lambda) f(x')$

Note that the least squares objective function defined in 1 is convex.

1.2.2 Gradient descent

We start with an arbitrary $w_0 \in \mathbb{R}^d$, then for $t = 0, 1, 2, \ldots$ we perform the following operation:

$$w_{t+1} = w_t - \eta_t \nabla \hat{R}(w_t)$$

where η_t is the learning rate.
Under mild assumptions, if the step size is sufficiently small, the gradient descent procedure converges to a stationary point, where the gradient is zero. For convex objectives, it therefore finds the optimal solution. In the case of the squared loss and a constant step size (e.g. 0.5), the algorithm converges at linear rate. If you look at the difference in empirical value at iteration t and compare that with the optimal value, then the gap is going to shrink at linear rate. If we look for a solution within a margin ϵ, it is found in $\mathcal{O}(\ln(\frac{1}{\epsilon}))$ iterations. The fact that the objective function congerges at linear rate can be formally described as follows:

$$\exists t_0 \forall t \geq t_0, \exists \alpha < 1 \text{ s.t. } (\hat{R}(w_{t+1}) - \hat{R}(\hat{w})) \leq \alpha(\hat{R}(w_t) - \hat{R}(\hat{w}))$$

where $\hat{w}$ is the optimal value for the hyperparameters.
For computing the gradient, we recall that:

$$\nabla \hat{R}(\hat{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} \hat{R}(w) & \ldots & \frac{\partial}{\partial w_d} \hat{R}(w) \end{bmatrix}$$

In one dimension, we have that:

$$\begin{aligned} \nabla \hat{R}(w) = \frac{d}{dw} \hat{R}(w) &= \frac{d}{dw} \sum_{i=1}^{n} (y_i - w \cdot x_i)^2 \\ &= \sum_{i=1}^{n} \frac{d}{dw} (y_i - w \cdot x_i)^2 \\ &= 2(y_i - w \cdot x_i) \cdot (-x_i) \\ &= \sum_{i=1}^{n} 2(y_i - w \cdot x_i) \cdot (-x_i) \\ &= -2 \sum_{i=1}^{n} r_i x_i. \end{aligned}$$

In d-dimension, we have that:

$$\nabla \hat{R}(w) = -2 \sum_{i=1}^{n} r_i x_i,$$

where $r_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$

1.2.3 Adaptive step size for gradient descent

The step size can be updates adaptively, via either:

1. **Line search**:
   Suppose at iteration t, we have w_t, $g_t = \nabla \hat{R}(w_t)$. We then define:

   $$y_t^* = \arg \min_{y \in [0, \infty)} \hat{R}(w_t) - \eta g_t$$

2. **Bold driver heuristic**:

   - If the function decreases, increase the step size.

     $$\text{If } \hat{R}(w_{t+1}) < \hat{R}(w_t) : \eta_{t+1} \leftarrow \eta_t \cdot c_{acc}$$

     where $c_{acc} > 1$

- If the function increases, decrease the step size.

$$\text{If } \hat{R}(w_{t+1}) > \hat{R}(w_t) : \eta_{t+1} \leftarrow \eta_t \cdot c_{dec}$$

where $c_{dec} < 1$.

1.2.4 Tradeoff between gradient descent and closed form

Several reasons:

- Computational complexity:

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

$(X^T X)$ can be computed in $\mathcal{O}(nd^2)$, $(X^T X)^{-1}$ can be computed in $\mathcal{O}(d^3)$.
By comparison, for gradient descent calculating $\nabla \hat{R}(w) = \sum_{i=1}^{n}(y_i - w^T x_i)x_i$ can be computed in $\mathcal{O}(nd)$, where $n = \ln(\frac{1}{\epsilon})$

- the problem may not require an optimal solution.
- many problems do not admit a closed form solution.

1.3 other loss functions

Least squares is part of a general case of the following general loss function, which is convex for $p \geq 1$.

$$l_p(r) = |r|^p \tag{3}$$

Least squares is where $p = 2$.

2 Probability (interlude)

2.1 Gaussians

The p.d.f. of a Gaussian distribution is given by:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x-\mu}{2\sigma^2}\right) \tag{4}$$

The p.d.f. of a multivariate Gaussian distribution is given by:

$$\frac{1}{2\pi\sqrt{|\sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \sigma^{-1}(x-\mu)\right) \tag{5}$$

where:

$$\sigma = \begin{pmatrix}\sigma_1^2, \sigma_{12} \\ \sigma_{21}, \sigma_2^2\end{pmatrix} \text{ and } \begin{pmatrix}\mu_1 \\ \mu_2\end{pmatrix} \tag{6}$$

2.2 Expectations

Expected value of a random variable can be calculated as follows:

$$\mathbb{E} = \begin{cases}\sum_x xp(x) & \text{if } X \text{ is discrete} \\ \int xp(x)dx & \text{if } X \text{ is continuous}\end{cases}$$

Expectations respect linear properties, i.e. let X, Y be random variable and $a, b \in \mathbb{R}$, then we have $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$.

3 Generalization and model validation

3.1 Fitting nonlinear functions via linear regression

Using nonlinear features of our data (basis functions), we can fit nonlinear functions via linear regression. Then, the model takes on the form:

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi(\mathbf{x}) \tag{7}$$

where $\mathbf{x} \in \mathbb{R}^d$, $x \mapsto \tilde{x} = \phi(\mathbf{x}) \in \mathbb{R}^d$ and $w \in \mathbb{R}^d$.

- 1 dim.: $\phi(\mathbf{x}) = \left[1, x, x^2, \ldots, x^k\right]$
- 2 dim.: $\phi(\mathbf{x}) = \left[1, x_1, x_2, x_1^2, x_2^2, \ldots, x_1^k, x_2^k\right]$
- p dim.: $\phi(\mathbf{x})$ vector of all monomials in $x_1, \ldots, x_p$ of degree up to k.

3.2 Achieving generalization

3.2.1 Independence and identical distribution

A fundamental assumption needs to be met: the dataset is generated from an independently and identically distributed from some unknown distribution P, i.e:

$$(x_i, y_i) \sim P(\mathbf{X}, Y).$$

The i.i.d. assumption is invalid when:

- we deal with time series data
- spatially correlated data
- correlated noise

If violated, we can still use ML but the interpretation of the results needs to be carefully analyzed. The most important thing is to choose the train/test split to assess the desired generalization properties of the trained model.

3.3 Expected error and generalization error

Once the iid assumption is verified, our goal is then to minimize the expected error (true risk) under P, i.e.:

$$\begin{aligned} R(\mathbf{w}) &= \int P(\mathbf{x}, y)(y - \mathbf{w}^T\mathbf{x})^2 dxdy \\ &= \mathbb{E}[(y - \mathbf{w}^T\mathbf{x})^2] \end{aligned}$$

The true risk can be estimated by the empirical risk on a sample dataset D:

$$\hat{R}_D(\mathbf{w}) = \frac{1}{|D|} \sum_{\mathbf{x},y \in D} (y - \mathbf{w}^T\mathbf{x})^2$$

The reason behind this approximation is because of the law of large numbers

Definition 3.1 (Law of large numbers)**.** $\hat{R}_D(\mathbf{w}) \to R_D(\mathbf{w})$ *for any fixed* $\mathbf{w}$ *as* $|D| \to \infty$.

$$\hat{\mathbf{w}}_D = \arg\min_{\mathbf{w}} \hat{R}_D(\mathbf{w}) \tag{8}$$

$$\mathbf{w}* = \arg\min_{\mathbf{w}} \hat{R}(\mathbf{w}) \tag{9}$$

We don't want to minimize the empirical risk given in equation 8 but the true risk given in equation 9, which are similar as the amount points in the dataset increases.

3.4 Uniform convergence

For learning via empirical risk minimization, uniform convergence is required, i.e.:

$$\sup_{\mathbf{w}} |R(\mathbf{w}) - \hat{R}_D(\mathbf{w})| \to 0 \text{ as } |D| \to \infty$$

Note that this is not implied by the law of large numbers alone, but depends on model class. It holds for instance for squared loss on data distributions with bouded support. Statistical learning theory is required to define these properties.

3.5 Evaluation of performance on training data

In general in holds that:

$$\mathbb{E}_D[\hat{R}_D(\hat{\mathbf{w}}_D)] \leq \mathbb{E}_D[R_D(\hat{\mathbf{w}}_D)]$$

Proof.

$$\begin{aligned}
\mathbb{E}[\hat{R}_D(\hat{\mathbf{w}}_D)] &= \mathbb{E}_D[\min_{\mathbf{w}} \hat{R}_D(\mathbf{w})] && \text{(ERM)} \\
&\leq \min_{\mathbf{w}} \mathbb{E}_D[\hat{R}_D(\mathbf{w})] && \text{(Jensen's inequality)} \\
&= \min_{\mathbf{w}} \mathbb{E}_D[\frac{1}{|D|}\sum_{i=1}^{|D|}(y_i - wx_i)^2] && \text{(Definition of } \hat{R}_D(.)) \\
&= \min_{\mathbf{w}} \mathbb{E}_D[\frac{1}{|D|}\sum_{i=1}^{|D|}(y_i - wx_i)^2] && \text{(linear expectations)} \\
&= \min_{\mathbf{w}} R(\mathbf{w}) \leq \mathbb{E}[R(\hat{w}_D)]
\end{aligned}$$

□

Thus, we obtain an overly optimistic estimate. A more realistic evaluation would be to use a separate test set from the same distribution P. Then:

- Optimize w on training set:
$$\hat{\mathbf{w}}_{D_{\text{train}}} = \arg\min_{\precapprox} \hat{R}_{\text{train}}(\precapprox)$$
- Evaluate on test set:
$$\hat{R}_{\text{test}}(\hat{\mathbf{w}}) = \frac{1}{|D_{\text{test}}|}\sum_{\mathbf{x},y \in D_{\text{test}}} (y - \hat{\mathbf{w}}^T\mathbf{x})^2$$
- Then:
$$\mathbb{E}_{D_{\text{train}},D_{\text{test}}}[\hat{R}_{D_{\text{test}}}(\hat{\mathbf{w}}_{D_{\text{train}}})] = \mathbb{E}_{D_{\text{train}}}[R(\hat{\mathbf{w}}_{D_{\text{train}}})]$$

Proof. Let $D_train = D$, $D_test = V$ and $D, V \sim P$. Then:

$$\begin{aligned}
\mathbb{E}_{D,V}[\hat{R}_V(\hat{\mathbf{w}}_D)] &= \mathbb{E}_D[\mathbb{E}_V[\hat{R}_V(\hat{\mathbf{w}}_D)]] && \text{independence of } D, V \\
&= \mathbb{E}_D[\mathbb{E}_V[\frac{1}{|V|}\sum_{i=1}^{|V|}(y_i - \hat{\mathbf{w}}_D^T x_i)^2]] && \text{(Definition of } \hat{R}_D(.)) \\
&= \mathbb{E}_D[\frac{1}{|V|}\sum_{i=1}^{|V|}\mathbb{E}_{x_i,y_i}(y_i - \hat{\mathbf{w}}_D^T x_i)^2] && \text{since } (x_i, y_i) \perp D \\
&= \mathbb{E}_D[R(\hat{\mathbf{w}}_D)]
\end{aligned}$$

□

3.6 Evaluation for model selection

For each candidate model m, we repeat the following procedure for $i = 1 : k$:

- We split the same dataset into training and validation sets:
$$D = D_{\text{train}}^{(i)} \biguplus D_{\text{val}}^{(i)}$$
- We train the model:
$$\hat{\approx}_{[}i, m] = \arg\min_{\mathbf{w}} \hat{R}_{\text{train}}^{(i)}(\mathbf{w})$$
- Then we estimate the error:
$$\hat{R}_m^{(i)} = \hat{R}_{\text{val}}^{(i)}(\hat{\mathbf{w}}_i)$$

Finally, select the model:

$$\hat{m} = \arg\min_m \frac{1}{k} \sum_{i=1}^{k} \hat{R}_m^{(i)}$$

3.7 Splitting the data for model selection

This splitting can be done randomly through Monte Carlo cross-validation.

- Pick training set of given size uniformly at random
- Validate on remaining points
- Estimate prediction error by averaging the validation error over multiple random trials.

It can also be achieved through k-fold cross-validation, which is the default choice.

- Partition the data into k folds
- Train on $k-1$ folds, evaluating on remaining fold.
- Estimate prediction error by averaging the validation error obtained while varying the validation fold.

Note that the cross-validation error is almost unbiased for large enough k. The following should be considered to pick k:

- Too small:
  - Risk of overfitting on test set
  - Using too little data for training
  - Risk of underfitting to training set
- Too large:
  - In general, leads to better performance. $k = n$ is perfectly fine, specific instance called leave-one-out cross-validation
  - Higher computational complexity.

In practice, $k = 5$ or $k = 10$ is ofen used and works well.

3.8 Best practice for evaludating models in supervised learning

Follow the following steps:

- Split data set into training and test set
- Never look at test set when fitting the model. For example, use k-fold cross-validation on training set
- Report final accuracy on test set, but never optimize on it.

Note that this procedure only works if the data is i.i.d. I.e. one should be careful if there are temporal trends or other dependencies.

4 Regularization

We want to avoid having overly complex models when minimizing the loss function. This can be achieved through regularization, which encourages small weights via penalty functions, which are called regularizers.

4.1 Ridge regression

This is a regularized optimization problem:

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T x_i)^2 + \lambda \|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} \mathbf{w}_j^2 \qquad \forall \lambda \geq 0$$

4.1.1 Closed form solution

This can be optimized using the closed form solution or gradient descent. The closed form for Ridge regression is:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{R})^{-1}\mathbf{X}^T y$$

where $I \in \mathbb{R}^{d\times d}$ is the identity matrix.

4.1.2 Gradient descent

$$\nabla\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_2^2\right) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\|\mathbf{w}\|_2^2$$

One step of the gradient descent is therefore performed as follows:

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_t(\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}_t) + 2\lambda\mathbf{w}_t) \\ &= (1 - 2\lambda\eta_t)\mathbf{w}_t - \eta_t\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}_t) \end{aligned}$$

Choosing the regularization parameter is done through cross-validation. Typically, the choice is between values of λ values which are logarithmically spaced.

4.1.3 Generalization of a tradeoff in ML

A lot of supervised learning problems can be written in this way:

$$\min_{\mathbf{w}} \hat{R}(\mathbf{w}) + \lambda C(\mathbf{w}).$$

It's possible to control complexity by varying regulzation parameter λ.

4.2 Renormalizing data through standardization

This process ensures that each feature has zero mean and unit variance:

$$\tilde{x}_{i,j} = \frac{(x_{i,j}) - \hat{\mu}_j}{\hat{\sigma}_j}$$

where $x_{i,j}$ is the value of the j^{th} feature of the i^{th} data point:

$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \qquad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \hat{\mu}_j)^2$$

5 Classification

Definition 5.1 (Classification)**.** *Classification is an instance of supervised learning where Y is discrete (categorical). We wanto to assign data points X (documents, queries, images, user visits) a label Y (spam/not spam, topic such as sports, politics, entertainment, click/no-click etc).*

The input of the model is labeled data set with positive and negative examples, see Figure 1. The output is a decision rule, i.e. a hypothesis. Given a dataset $D = \{(x_1, y_1), \dots (x_n, y_n)\}$, we

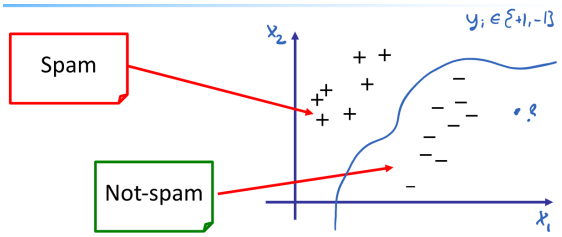

Figure 1: Illustration of binary classification

have:

$$y \approx h_{\mathbf{w}}(x) = \text{sign}(w^T x)$$

Linear classification works well in high-dimensional settings when using the right features; prediction is typically very efficient despite linear classification seeming very restrictive at first.

5.1 Finding linear separators

Writing the search for a classifier can be seen as an optimization problem: we seek the set of weights $\mathbf{w}$ that minimizes the number of mistakes, i.e.:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w} \in \mathbb{R}} \sum_{i=1}^{n} [y_i \neq \text{sign}(w^T x_i)]$$

$$= \begin{cases} 1 \text{ if } y_i \neq \text{sign}(w^T x_i) \\ 0 \text{ otherwise.} \end{cases}$$

The goal is then to optimize the following function:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w} \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} [y_i \neq \text{sign}(w^T x_i)]$$

$$= \frac{1}{n} \sum_{i=1}^{n} \updownarrow(w; x_i, y_i)$$

Note that this poses as challenge as it is not convex or even differentiable. Therefore we need to replace this loss by a tractable loss function for the sake of optimization/model fitting. When evaluating a model, we then use the original cost/performance function. The function we can use to optimize in this case is the surrogate loss:

$$l_P(\mathbf{w}; y_i, x_i) = \max(0, -y_i \mathbf{w}^T x_i)$$

which is also referred to as the perceptron loss.

The gradient of the perceptron loss function can be computed as follows:

$$\hat{R(w)} = \sum_{i=1}^{n} \max(0, -y_i w^T x_i)$$

$$\nabla \hat{R(w)} = \sum_{i=1}^{n} \nabla_{\mathbf{w}} \max(0, -y_i w^T x_i)$$

$$= \begin{cases} 0 \text{ if } y_i w^T x_i \geq 0 \text{ i.e. correctly classified} \\ -y_i x_i \text{ otherwise} \end{cases}$$

So we have the following update rule:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}w_t + \eta_t \sum_{i:(x_i,y_i)incorrectlyclassifiedbyw}^{x_i y_i}$$

5.2 Stochastic gradient descent

Computing the gradient requires summing over all data, which is inefficient for large datasets. Additionally, our initial estimates are likely very wrong and we can get a good unbiased gradient estimate by evaluating the gradient on few points. In the worst case, we can evaluate only one randomly chosen point, which is a procedure called stochastic gradient descent. It consists of the following steps:

1. Start at an arbitrary $\mathbf{w}_0 \in \mathbb{R}^d$
2. For $t = 1, 2, \ldots$ do:
   - Pick data point $(\mathbf{x}', y') \in D$ from training set uniformly at random (with replacement), and set:
   $$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \updownarrow(\mathbf{w}_t; \mathbf{x}', y')$$

Where η_t is called the learning rate. Guaranteed to converge under mild conditions, if:

$$\sum_t \eta_t = \infty \text{ and } \sum_t \eta_t^2 < \infty$$

for instance $\eta_t = \frac{1}{t}$ and $\eta_t = \min(c, \frac{c'}{t})$.
The perceptron algorithm is just stochastic gradient descent on the perceptron loss function $\updownarrow_P$ with learning rate 1.

Theorem 1 (Perceptron algorithm)**.** *If the data is linearly separable, the perceptron will obtain a linear separator.*

The variance of the gradient estimate can be reduced by averaging over the gradients w.r.t. multiple randomly selected points, which are called minibatches. Adaptive learning rates can be additionally applie. There exist various approaches for adaptively tuning the learning rate. Often times, these even use a different learning rate per feature. Examples of adaptive learning rate algorithms include AdaGrad, RMSProp, Adam, ...

5.3 Hinge loss vs. perceptron loss

The Hinge loss encourages the margin of the classifier, and is defined as follows:

$$\updownarrow_H(\mathbf{w}; \mathbf{x}, y) = \max\left\{0, 1 - y\mathbf{w}^T\mathbf{x}\right\}$$

6 Support vector machines vs. perceptron

The optimization objective for the support vector machine is defined as minimizing Hinge loss while also adding another regularization term. There are several lines that need to be considered in the max. margin linear classification, which are summarised in figure 2

Support vector machinea re widely used, very effective linear classifiers. They behave almost like a perceptron. The only differences include:

- Optimize slightly different, shifted loss (hinge loss)
- They regularize the weights

It can be optimized using a stochastic gradient descent. A safe choice for the learning rate is:

$$\eta_t = \frac{1}{\lambda t}$$

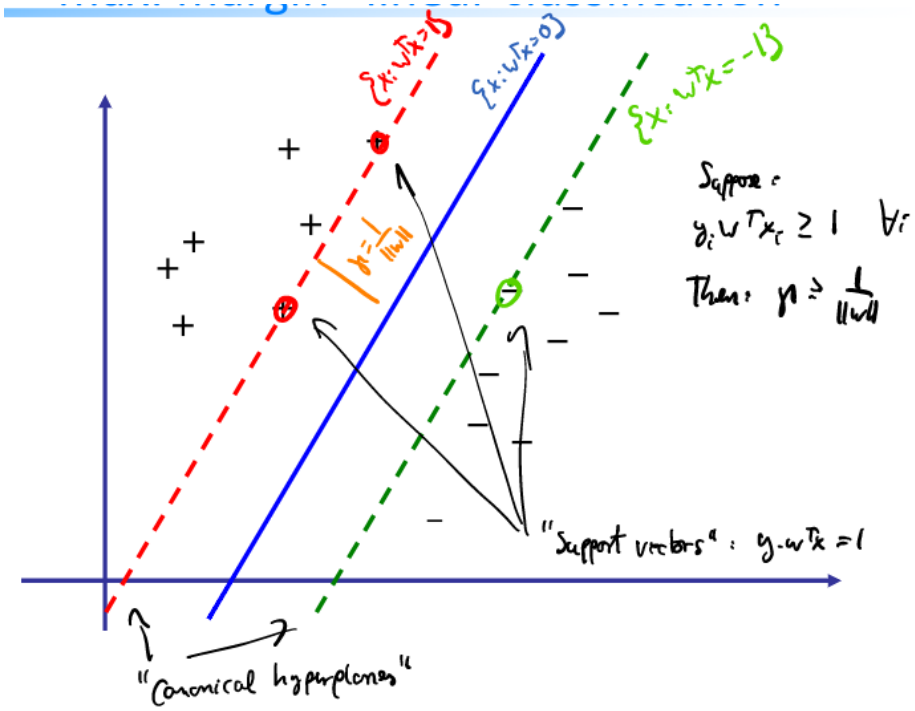

Figure 2: Important elements and their defining equations for support vector machines.