Popular machine learning techniques

ME 416 - Prof. Tron

Saturday 27th April, 2019

The following is a list of the most popular paradigms used for learning (mostly classification).

1 General classifier models

In general, the model for a classifier is a function $f(x;\theta)$, where θ is the set of parameters of the classifier.

The model typically returns values of either two kinds:

For binary classification problems The function $f(x)$ returns a scalar; the label given by the classifier is the positive (respectively, negative) class if the returned value is positive (respectively, negative), i.e., $f(x;\theta) > 0$ (respectively, $f(x;\theta) < 0$).

- The set of samples x for which $f(x;\theta) = 0$ is called the *decision boundary*. For points on this boundary, the classifier is undecided (it does not return either label).
- The absolute value $|f(x;\theta)|$ is called the *confidence* of the classifier; this value is a way to quantify the "distance" of $f(x;\theta)$ from the decision boundary.

See Figure 1 for a visualization in the particular case of a linear classifier.

For multi-class problems with $L > 2$ classes The function $f(x)$ typically returns a vector of L elements, and the predicted label is given by $\hat{y} = \text{argmax}_i[f(x;]_i$, that is, the position of the highest entry gives the label. The *decision boundary* is the set of x where two or more entries are maximum and equal. If $f(x;\theta)$ is normalized such that all the entries sum to one ($\sum_{i=1}[f(x;]_i = 1$), then the entries are called the *pseudo-proabilities*.

The different types of classifiers below differ on the assumptions made on the shape of the classes in the input space, and the corresponding type of function $f(x;\theta)$ used.

2 Linear classifiers

For binary classification, the idea is of this type of models is to define an hyperplane[1] in the space of the inputs that separates the two classes. The model is of the form $f(x;\theta) = w^{\mathrm{T}}x + b$, where the parameters $\theta = \{w, b\}$ are the vector w (the normal to the hyperplane), and the scalar b.

[1] Technically, an hyperplane is a linear subspace of dimension $d-1$, where d is the dimension of x. For instance, if x is 2-D, then an hyperplane is simply a line.

★★Typically w is chosen such that $\|w\| = 1$; the effect of this is that $|f(x;\theta)|$ (i.e., the confidence of the classifier) is then the actual distance of x from the decision boundary .

★★★The *margin* of a linear classifier is the minimum distance of a training point from the boundary. One of the best algorithms for learning linear classifiers is *Support Vector Machines*, which finds the hyperplane with the maximum margin.

See Figure 1 for a visual example of a linear classifier.

★★★For multi-class tasks, a straightforward solution is to train multiple classifiers in a *one-vs-all* fashion (i.e., one classifier per class to separate that class from the others). During testing, all the classifiers are evaluated, an the one returning the highest confidence "wins".

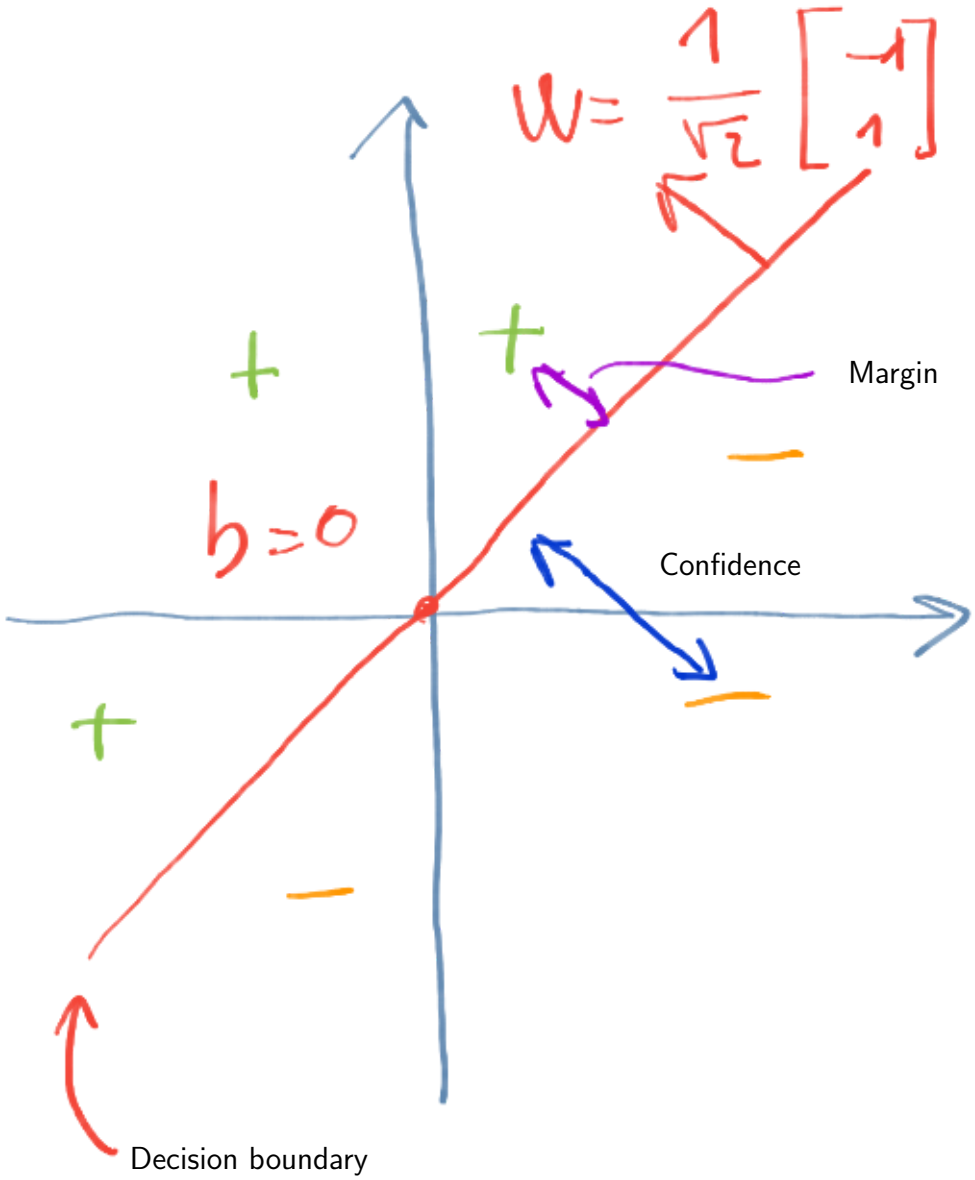


Figure 1: Example of linear classifier with corresponding general concepts.

2.1 Pros and cons

+ Fast to evaluate
+ Confidence values easy to understand
- Rarely classes are linearly separable

3 Gaussian classifiers

The assumption underlying this type of classifiers is that the input vectors from the same class form a "cloud" or "cluster".

More precisely, the model is based on a Gaussian distribution

$$f(x;\theta) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left(\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\right) - c. \tag{1}$$

The parameters of this model are the mean μ (a vector of dimension d), the covariance Σ (a positive definite matrix of dimensions $d \times d$, where d is the dimension of x) and the threshold c.

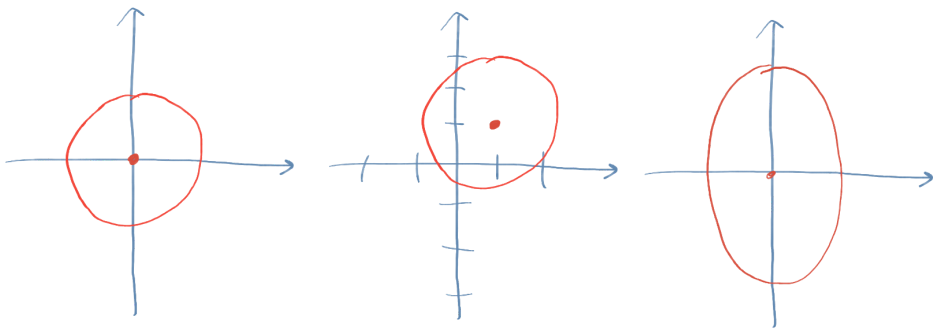
The decision boundary of this classifier is an ellipse, centered at the mean μ and whose shape is given by Σ, and whose scale is given by c.

TODO: insert illustration

★★★Technically, the major axes of the ellipse, and the corresponding radii are given by the *eigenvectors* and *eigenvalues* of the matrix Σ.

The simplest case is when Σ is *isotropic*, i.e., it is proportional to the identity matrix ($\Sigma = \sigma^2 I$); then, the decision boundary is a sphere (a circle, in 2-D). The second-simplest case is where Σ is a diagonal matrix (with all strictly positive numbers on the diagonal); then, the principal axes of the ellipse are aligned with the coordinate axes. See Figure 2 for illustrations of these cases.

★★★Training a Gaussian classifier is very easy. The mean is set to the empirical mean of the training data, i.e., $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$, while the covariance matrix is set to the empirical covariance $\Sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^{\mathrm{T}}$.



(a) $\mu = 0, \Sigma = I$ (b) $\mu = \begin{bmatrix}1\\1\end{bmatrix}, \Sigma = I$ (c) $\mu = 0, \Sigma = \begin{bmatrix}1 & 0\\0 & \frac{1}{2}\end{bmatrix}$

Figure 2: Decision boundaries of a Gaussian classifier for different values of μ and Σ.

3.1 ★★★ Mahalanobis distance

Taking the logarithm of $f(x;\theta) = 0$ (describing the decision boundary), one can see that an equivalent classifier can be obtained by using $f(x;\theta) = \|x-\mu\|_\Sigma^2 - \log(c)$, where the quantity $\|x-\mu\|_\Sigma^2 = (x-\mu)^{\mathrm{T}}\Sigma(x-\mu)$ is the *Mahalanobis distance*. This distance takes into account the shape of the distribution (points along the elongated axes of the ellipse must be further away to have the same Mahalanobis distance of points along the shortened axes).

See the Wikipedia article on Mahalanobis distance for further details.

3.2 Application to multi-class problems

Gaussian classifiers are very well suited for multi-class problems where each class forms a cluster.

TODO: insert graphical illustration of multiple clusters

If all the classes have the same "spherical" shape (isotropic covariance), one can predict the label from the cluster with the closest mean, i.e.,

$$\hat{y} = \underset{i}{\operatorname{argmin}} \, ||x - \mu_i||^2 \tag{2}$$

If the clusters have very different size and shape (non-isotropic variance), then one should use the Mahalanobis distace discussed in § 3.1.

3.3 Pros and cons

+ Fast to train and evaluate.
+ This type of clusters can be learned in an unsupervised way (no manual labeling necessary)
- The assumption of Gaussian clusters is not always realistic

4 Nearest neighbors (NN)

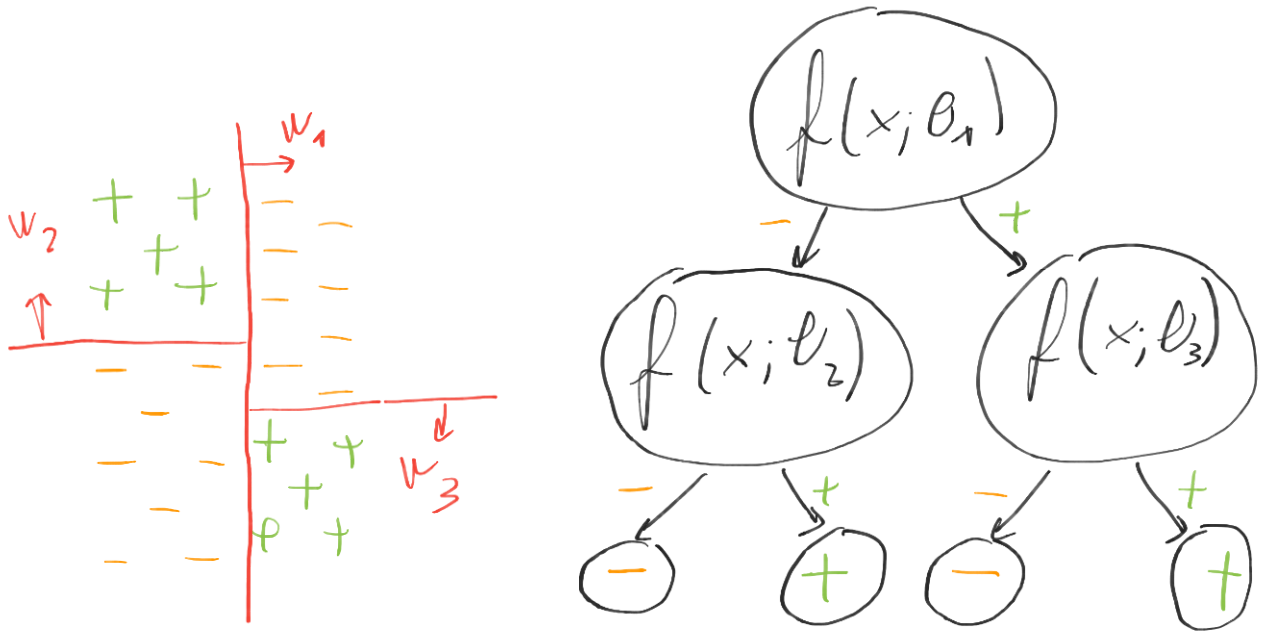
This is the easiest type of classifier. The predicted label $\hat{y}$ is simply the label of the closest input point in the training set.

4.1 Pros and cons

+ Can represent classes of arbitrary "shapes" (i.e., there is no assumption of linear separability or Gaussian clusters)
+/- In the simplest implementation, training is fast (just store the training set). However, testing is typically slow, as each new input point needs to be compared with the entire training set.

5 Decision trees

Decision trees predict a class label based on a hierarchy of rules. Each node in the tree represents a decision to make (typically based on a linear classifier), except for those without children, which provide the final label. During testing on a data point x, first the rule at the root of the tree is evaluated. Depending on the result, one of the children is selected and the process is repeated until a final label is returned



(a) Training data, the three linear classifiers, and the decision boundary of the decision tree.

(b) Tree visualization of the relation between the tree classifiers. A test point x starts from the root (top), and, depending on the result of the classifier in that node, it is sent to another classifier, repeating the process until one of the leaves of the tree (bottom) is reached. The reached leaf gives the final result for the overall classifier.

Figure 3: Example of decision tree where each node in the tree is a linear classifier.

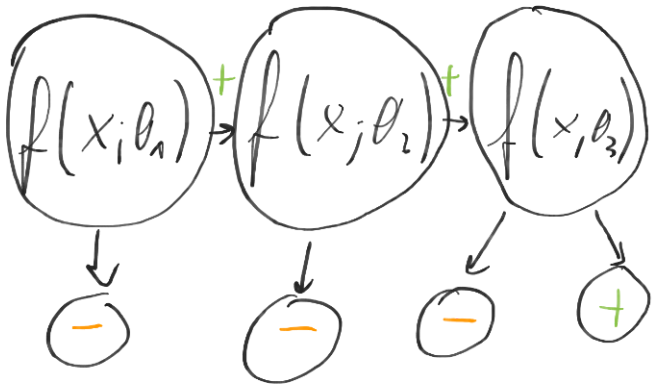


Figure 4: Example of a cascade

+ Relatively fast to evaluate
+ Can lead to interpretable results (i.e., one can see "why" a particular label was output)
- No "optimal" way to train, only heuristics.

5.1 Cascades

Cascades are a particular type of tree for binary classification, say, "positive" vs "negative". The tree is actually in the form of a chain, where each decision leads to either the "negative" label, or to another decision (except for the last one, which can output the "positive" label).

Typically, the classifiers are more precise but more computationally intensive the deeper we get in the cascade.

TODO: ★★★Random forests

6 Boosting

Boosting relies on the idea of combining the output of a large number of "weak" classifiers (i.e., classifiers that, alone, have low performance) into a single "strong" classifiers by weighted majority voting. In practice, given a test sample x, the output of the overall classifier is produced in three steps:

1) Evaluates a number of weak classifiers $\{f(x;\theta_i)\}_{i=1}^N$ with different parameters θ_i. ★★★Each i is sometimes called a *stage* of the classifier.

2) Transform the confidence of each classifier into a discrete $+1/-1$ output $\{\sigma\big(f(x;\theta_i)\big)\}$, where σ is the sign function

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

3) Weight each output with a weight α_i, and then do majority voting (i.e., sum all the responses). The final output of the overall classifier is then $f(x;\{\theta_i\}) = \sum_{i=1}^N \alpha_i \sigma\big(f(x;\theta_i)\big)$.

Typically, weak classifiers are chosen to be very simple (e.g., linear classifiers), but in general this does not need to be.

The main question is on how to generate the weak classifiers. One of the classical methods is AdaBoost, where the weak classifiers are generated incrementally by re-weighting the datasets to give more importance to the errors made by the combination of the previous classifiers. The final result of the overall classifier is by transforming

6.1 Pros and cons

+ One can make the final classifier faster by selecting only the most important weak classifiers (sparse boosting)
+ In some instances, it can give very high accuracies.
+ The weight of each classifier explains how it contributes to the final result.
- It is typically hard to intuitively interpret why each classifier was selected.

Figure 5: ReLU non-linear function

7 (Deep) Neural Networks

One way to intuitively understand how neural network classifiers work is to start with boosted linear classifiers. Referring to § 6, assume that each weak classifier is linear, i.e., $f(x;\theta_i) = w_i^{\mathrm{T}} x + v_i$. We can stack together all the outputs of the classifiers in a vector, all the normals w_i into a matrix A_1, and all the constants v_i into another vector b_1.

$$\begin{bmatrix} \vdots \\ f(x;\theta_i) \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \vdots \\ w_i^{\mathrm{T}} \\ \vdots \end{bmatrix}}_{A_1} x + \underbrace{\begin{bmatrix} \vdots \\ v_i \\ \vdots \end{bmatrix}}_{b_1} \tag{4}$$

We can then redefine the function σ for vector inputs so that it is applied element-wise. Then, we can collect all the weights α_i into a row vector A_2. The final output of the classifier is then given by:

$$f(x;\{\theta_i,\alpha_i\}) = \underbrace{\begin{bmatrix} \cdots & \alpha_i & \cdots \end{bmatrix}}_{A_2} \sigma(A_1 x + b_1) = A_2 \sigma(A_1 x + b_1), \tag{5}$$

which is given by a sequence of an affine transformation (multiplication by A_1 and summation by b_1), followed by a non-linear operation (sign function σ), followed by another affine transformation (multiplication by A_2).

Neural networks are a generalization of this idea. The sequence of an affine transformation and a non-linear operation is called a *layer* of the network. The nonlinear transformation σ is typically substituted with a *Rectified Linear Unit (ReLU)* (see Fig. 5), although, in principle, any almost-everywhere-differentiable function could be used.

Deep neural networks are obtained by concatenating several layers, sometimes with different non-linear operations:

$$f(x;\theta) = A_n \sigma_n(A_{n-1}\sigma_{n-1}(\cdots A_1 x + b_1)\cdots) + b_{n-1}) + b_n). \tag{6}$$

The parameters of the classifier are the entries of the matrices and vectors for the affine operations, $\theta = \{A_i, b_i\}_{i=1}^n$; these are called the *weights* of the network.

TODO: Training algorithms.

7.1 Pros and cons

+ Can potentially model arbitrary decision boundaries.
- Requires lots of training data and computations.
- Deciding the structure of the network, and the parameters for the training algorithm, is more an art than a science (although there are recent algorithms trying to automate these decisions).
- Hard to interpret. It is usually hard to tell what is the effect of each weight or layer on the end result.