CS-233 (EPFL)
Intro to Machine Learning

David Duan

Spring 2020

Machine learning and data analysis are becoming increasingly central in many sciences and applications. In this course, fundamental principles and methods of machine learning will be introduced, analyzed and practically implemented.

Introduction - KNN, data representation, basic optimization...
Linear models - Linear regression, least-square classification, logistic regression, linear SVMs...
Nonlinear method - Polynomial regression, kernel methods.
Deep learning - Multi-layer pereceptron, CNNs...
Unsupervised learning - Dimensionality reduction, clustering...

Overview

We can view machine learning as an optimization problem. Our goal is to minimize

$$E(\mathbf{w}) = \sum_{n=1}^{N} L(y(\mathbf{x}_n; \mathbf{w}), t_n),$$

where

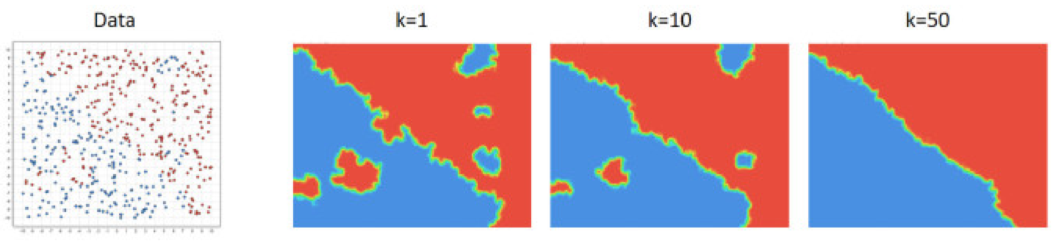
- $\mathbf{w}$: model parameters
- $\mathbf{x}$: feature vector of the n-th sample
- t_n: label associated with the n-th sample
- y: predictor
- L: loss function
- E: error function

Note: This document is merely a highlight of CS-233 at EPFL (written mainly for exam-prep purposes) and is NOT a comprehensive introduction to Machine Learning in any way. The following topics are not included here:

- SVM (separate)
- Trees (separate)
- NNs (omitted)

1. K Nearest Neighbours (Supervised - Classification)

Given an input x, we classify it according to the majority of labels of its k clearest neighbours.



As k becomes larger, the decision boundary becomes smoother; the error on the training set increases but the error on the test set decreases, up to a point. The optimal k can be found using cross-validation.

One critical assumption is that the *training set and the test set are drawn from the same statistical distribution*. Otherwise, there is no reason for a decision boundary learned on the training set to be useful on the test set.

1.1. Disadvantages of KNN

1. The KNN algorithm is prone to misclassifying points near the decision boundaries.
2. If the dataset is unbalanced, meaning there are more samples from one class compared to the rest, then the better represented class is unduly favoured.
3. The computational cost for decision boundaries could be expensive.

1.2. Condensed Nearest Neighbour for Data Reduction

Condensed nearest neighbour is an algorithm designed to reduce the dataset for KNN classification. It selects the set of **prototypes** P from the training data, such that 1NN with P can classify the examples almost as accurately as 1NN does with the whole dataset.

Given a training set X, condensed NN works iteratively:

1. Scan all elements of X, looking for an element x whose nearest prototype from P has a different label than x.
2. Remove x from X and add it to P
3. Repeat the scan until no more prototypes are added to U.

We can replace X by P and yield a similar performance (accuracy) but with a much faster computation time. The samples that are not prototypes are called the *absorbed points*.

1.3. The Curse of Dimensionality

In a high-dimensional space, everything is "far" from everything else. The Euclidean distance becomes less and less meaningful as the distance increases. To guarantee the effectiveness of an estimator, the distance between neighbouring training samples must be less than some value d that depends on the problem.

2. Perceptron (Supervised - Classification)

Splitting data using an n-dimensional hyperplane. Note this only works for *linearly separable* data.

2.1. Problem Statement

Given a set $X = \{\mathbf{x}_i\}_{1\le i\le N}$ of n-dimensional samples $\mathbf{x} \in \mathbb{R}^n$, define $\tilde{\mathbf{x}} = [1|\mathbf{x}] \in \mathbb{R}^{n+1}$. Denote the weights by $\tilde{\mathbf{w}} = \langle w_0, w_1, \ldots, w_n\rangle \in \mathbb{R}^{n+1}$ where $\sum_{i=1}^{n} w_i^2 = 1$ and w_0 is the bias. We wish to find $\tilde{\mathbf{w}}$ such that for all/most positive samples, $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$ and for all/most negative samples, $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$.

The decision boundary (hyperplane) is defined as $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = w_0 + \sum_{i=1}^{n} w_i x_i = 0$. The **signed distance** from $\tilde{\mathbf{x}}$ to the decision boundary is given by $h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$. (See Appendix A for more math.)

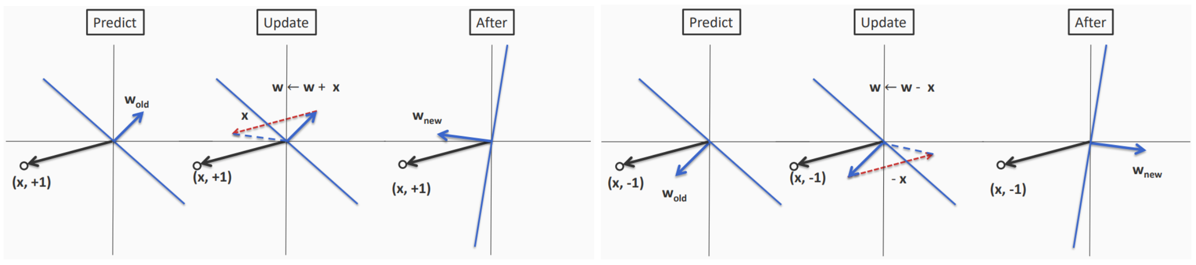
2.2. Algorithm

Given a training set $\{(\mathbf{x}_i, t_i)_{1\le i\le N}\}$ where $\mathbf{x}_i \in \mathbb{R}^n$ and and $t_i \in \{-1, 1\}$,

1. Initialize $\mathbf{w}_0 = 0 \in \mathbb{R}^n$.
2. For each training example $(\mathbf{x}_i, t_i)$,
   a) Predict $t_i' = \text{sgn}(\mathbf{w} \cdot \mathbf{x}_i)$.
   b) If $t_i' \neq t'$, update $\mathbf{w} \leftarrow \mathbf{w} + t_i \mathbf{x}_i$.

2.3. Intuition

Suppose we have made a mistake on a positive example, that is, $t = 1$ but $\mathbf{w} \cdot \mathbf{x} < 0$. Let the new vector be $\mathbf{w}' := \mathbf{w} + \mathbf{x}$. The new dot product will be $\mathbf{w}' \cdot \mathbf{x} = (\mathbf{w} + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x} > \mathbf{w} \cdot \mathbf{x}$. Thus, for a misclassified positive example, the algorithm will increase the score assigned to the same input (left). A mirror argument gives the intuition for misclassified negative examples (right).



2.4. Further Comments

The **convergence theorem** states that if there exists a set of weights that are consistence with the data, i.e., the data is linearly separable, the perceptron algorithm will converge. (See **Mistake Bound Theorem** in Appendix A.) The **cycling theorem** states that if the training data is not linearly separable, then the learning algorithm will eventually repeat the same set of weights and enter an infinite loop.

In summary the perceptron algorithm only works for linearly separable dataset and provides no way to favour one hyperplane over the other. To address these weaknesses, we introduce logistic regression.

3. Logistic Regression (Supervised - Classification)

The perceptron algorithm assigns a label $y(\mathbf{x}) = (\mathbf{w}^T\mathbf{x} > 0)$ to each $\mathbf{x}$. We would like a *softer* version, i.e., a "probability", by replacing the step function by a smoother sigmoid function σ. The prediction thus becomes $y(\mathbf{x}; \tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})$.

3.1. The Sigmoid Function

The sigmoid function $\sigma(a) = \dfrac{1}{1+\exp(-a)}$ is infinitely differentiable with $\partial\sigma/\partial a = \sigma(1-\sigma)$. Also, $\sigma(a) \to 1$ as $a \to \infty$ and $\sigma(a) \to 0$ as $a \to -\infty$. Thus, it acts like a smoothed step function.

3.2. Probabilistic Interpretation

$$y(\mathbf{x}; \tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}) = \frac{1}{1+\exp(-\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})} \in [0, 1].$$

- $y(\mathbf{x}; \tilde{\mathbf{w}}) = 0.5$ when $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$, i.e., $\mathbf{x}$ is on the decision boundary.
- $y(\mathbf{x}; \tilde{\mathbf{w}}) \to 0.0$ or $y(\mathbf{x}; \tilde{\mathbf{w}}) \to 1.0$ if $\mathbf{x}$ is far from the decision boundary.

$y(\mathbf{x}; \tilde{\mathbf{w}})$ can be interpreted as the probability that $\mathbf{x}$ belongs to one class or the other. It can be shown that LR finds the ML solution under the assumption that the noise is Gaussian.

3.3. Algorithm

Given a training set $\{(\mathbf{x}_n, t_n)_{1\le n\le N}\}$ where $\mathbf{x}_n \in \mathbb{R}^n$ and and $t_n \in \{0, 1\}$, minimize the cross-entropy

$$E(\tilde{\mathbf{w}}) = -\sum_n \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

with respect to $\tilde{\mathbf{w}}$, where $y_n = y(\mathbf{x}_n; \tilde{\mathbf{w}})$.

3.4. Multi-class Case

Given a training set $\{(\mathbf{x}_n, [t_n^1, \ldots, t_n^k])_{1\le n\le N}\}$ where $\mathbf{x}_n \in \mathbb{R}^n$ and and $t_n^k \in \{0, 1\}$ is the probability that sample $\mathbf{x}_n$ belongs to class C_k, the activation is $a^k(\mathbf{x}) = \sigma(\mathbf{w}_k^T\mathbf{x})$; the probability that $\mathbf{x} \in C_k$ is

$$y^k(\mathbf{x}) = \frac{\exp(a^k(\mathbf{x}))}{\sum_j \exp(a^j(\mathbf{x}))}.$$

Multi-class entropy and its gradient is given by

$$E(\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_k) = -\sum_n\sum_k t_n^k \ln(y^k\mathbf{x}_n), \quad \nabla E_{\mathbf{w}_j} = \sum_n (y^k(\mathbf{x}_n) - t_n^k)\mathbf{x}_n.$$

3.5. Further Comments

LR is better than perceptron. However, outliers can often cause problems as LR tries to minimize error rate at training time. See the separate document on support vector machines.

4. Adaboost (Supervised - Classification)

Boosting is a general strategy for learning classifiers by combining simpler ones. The idea is to take a "weak classifier" and use it to build a much better classifier, there by boosting the performance of the weak classification algorithm. The most popular boosting algorithm is **AdaBoost**, so-called because it is "adaptive".

4.1. Algorithm

For a training set $\chi = \{\mathbf{x}_n, t_n\}$ where $t_n \in \{-1, 1\}$ for $1 \leq n \leq N$:

1. Initialize data weights uniformly, i.e., $\forall n : w_n^1 = 1/N$.
2. For $t \in \{1, \ldots, T\}$:
   a) Find classifier $y_t : \chi \to \{-1, 1\}$ that minimizes the **weighted error** $\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t$.
   b) Evaluate the **weighted classified error** ε_t and the **weight** for the current classifier α_t:

$$\varepsilon_t = \frac{\sum_{t_n \neq y_t(\mathbf{x}_n)} w_n^t}{\sum_{n=1}^N w_n^t}$$
$$\alpha_t = \log \frac{1 - \varepsilon_t}{\varepsilon_t}$$

   c) Update weights for the next iteration:

$$w_n^{t+1} = w_n^t \exp(\alpha_t I(t_n \neq y_t(\mathbf{x}_n)))$$

3. Output the final classifier:

$$Y(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t y_t(\mathbf{x})\right)$$

4.2. Comments

- The ***weighted error*** is simply the sum of weights of all misclassified data points.
- The ***weight classified error*** ε_t is the percentage of misclassified weights; it is bounded by 0 and 1 and is less than 0.5 if the classifier y_t performs better than chance.
- The ***weight for the current classifier*** α_t depends on the performance of the current classifier y_t; it is positive if y_t performs better than chance.
- After each iteration, the weight of misclassified samples is increased.

Remark. See the separate document for more information on tree-based models.

5. K-Means Clustering (Unsupervised - Clustering)

Given a set of input samples, **k-means clustering** aims to partition n observation into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

5.1. Center of Gravity

The mean of points $\{x_1, \ldots, x_N\}, x_i \in \mathbb{R}^D$ is $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \in \mathbb{R}^D$. If the x_i's were physical points of equal mass, then μ would be their center of gravity. Note this applies in any dimension.

5.2. K-Means Clusters

Suppose cluster k is formed by the points $\{x_{i_1^k}, \ldots, x_{i_{n^k}^k}\}$ with center of gravity μ_k.

- Superscript $k \in \{1, \ldots, K\}$ denotes which cluster the point belongs to.
- Subscript $j \in \{1, \ldots, n_k\}$ denotes the index of the point within the cluster.

Intuitively, distances between the points within a cluster should be small and the distances across clusters should be large. We can encode this via the distance to cluster centers $\{\mu_1, \ldots, \mu_K\}$:

$$\min \sum_{k=1}^{K} \sum_{j=1}^{n_k} (x_{i_j^k} - \mu_k)^2$$

where $\{x_{i_1^k}, \ldots, x_{i_{n^k}^k}\}$ are the n^k samples that belong to cluster k.

However, we don't know what points belong to what cluster, and we also don't know the center of gravity of the clusters.

5.3. Algorithm

The algorithm iteratively updates the cluster assignment for each point and the cluster centers.

- Initialize $\{\mu_1, \ldots, \mu_K\}$, randomly if necessary.
- Until convergence,
  - Assign each point x_i to the nearest center μ_k.
    - For each point x_i, compute the Euclidean distance to every center $\{\mu_1, \ldots, \mu_K\}$.
    - Find the smallest distance. The point is said to be assigned to the corresponding cluster. Note that each point is assigned to a single cluster.
  - Update each center μ_k given the points assigned to it. Recompute each center μ_k as the mean of the points that were assigned to it.

The algorithm is guaranteed to converge to a stable solution, where the cluster centers and the point assignments are fixed. Note it does not always converge to the best (desired) solution, as random initialization often matters. In practice, try several different randomly initialization and keep the one that yields the best result in term of the sum of square distances.

6. Principal Components Analysis (Unsupervised - Dimension Reduction)

Principal component analysis is a method for identifying patterns in data and expressing the data in such a way as to highlight their similarities and differences. Once you have found these patterns, you can compress the data, i.e., by reducing the number of dimensions, without much loss of information.

6.1. Intuition

Suppose we are given observations with measurements on a set of n features. Most of the times, not all of these dimensions are equally interesting. PCA seeks a smaller number of dimensions that are as interesting as possible, where the concepts of *interesting* is measured by the amount that the observations vary along each dimension. Each of the dimension found by PCA is a linear combination of the n features and the dimensions are orthogonal to each other.

Alternatively, principal components provide low-dimensional linear surfaces that are *closest* to the observations. For example, the first principal component is the line in n-dimensional space that is closest to the observation (wrt the Euclidean metric). Indeed, we seek a single dimension of the data that lies as close as possible to all of the data points, since such a line will likely to provide a good summary of the data.

6.2. Procedure

1. Subtract the mean from each of the data dimensions to produce a dataset with mean zero.
2. Calculate the covariance matrix K where $K_{i,j} = \text{Cov}(X_i, X_j)$.
3. Calculate the eigenvectors for matrix K and order them by eigenvalues, highest to lowest.
    - Eigenvectors of K are actually the dimensions of the axes where there is the most variance and that we call *principal components*.
    - Eigenvalues are simply the coefficients attached to eigenvectors, which give the amount of variance carried in each principal component.
4. Choose what to do with these components and form a feature vector.
    - You can either keep all of the components or discard those of less significance and form a matrix called **feature vector** whose columns are the eigenvectors.
    - If particular, if you choose to keep the top p eigenvectors/components out of n, the final data set will have only p dimensions.
5. Recast the data along the principal components axes:

$$\text{FinalDataset} = \text{FeatureVector}^T\,\text{StandardizedOriginalDataset}^T\,.$$

References.

- A step by step explanation of Principal Components Analysis.
- A tutorial on Principal Components Analysis
- ISLR Chapter 10.

7. LDA (Supervised - Classification / Dimension Reduction)

Linear Discriminant Analysis seeks to best separate (discriminate) the samples in the training dataset by their class value. Specifically, the model seeks to find a linear combination of input variables that achieves the maximum separation for samples between classes and the minimum separation of samples within each class.

7.1. Procedure

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
   - For each class i, compute $m_i = \frac{1}{n_i} \sum_{x \in D_i} x_k \in \mathbb{R}^d$.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
   - Within-class scatter matrix S_W:

     $$S_W = \sum_{i=1}^{c} S_i, \quad \text{where } S_i = \sum_{x \in D_i} (x - m_i)(x - m_i)^T.$$

   - Between-class scatter matrix S_B:

     $$S_B = \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T,$$

     where m is the overall mean vector and m_i, N_i are the sample mean and sizes of the respective class.
3. Compute and sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ matrix W (columns are eigenvectors).
   - Solve the generalized eigenvalues problem for the matrix $S_W^{-1} S_B$ to obtain linear discriminants.
   - Roughly speaking, eigenvectors with high eigenvalues bear more information.
4. Use this matrix W to transform the sample onto the new subspace: $Y = XW$.

7.2. PCA vs. LDA

- Both are linear transformation techniques.
- LDA is supervised whereas PCA is unsupervised.
- PCA seek component axes that maximize the variance; LDA seek component axes that maximize between-class variance and minimize within-class variance.
- LDA makes assumptions about normally distributed classes and equal class covariances.

A. Math Prerequisites for Linear Classification

A.1. Normal Vector to a Line

Prop. 1. A **normal vector** to line $ax + by + c$ is a scalar multiple of $u = \langle a, b\rangle$.

Proof. A normal vector of a 2-dimensional line will have the direction vector of an orthogonal line to it. Rewriting $ax + by + c = 0$ in the point-slope form, we get

$$y = -\frac{a}{b}x - \frac{c}{b}.$$

Any vector that is perpendicular to L must have slope b/a, or equivalently, any normal vector to L will be a non-zero multiple of $u = \langle a, b\rangle$. □

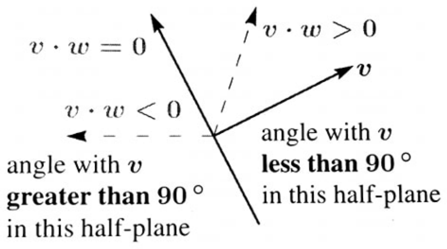
Cor. 2. A **unit normal vector** to line $ax + by + c$ is of the form $\hat{u} = \frac{1}{\sqrt{a^2+b^2}}\langle a, b\rangle$.

A.2. Dot Product

Recall that $\mathbf{u} \cdot \mathbf{v} = ||\mathbf{u}||\,||\mathbf{v}|| \cos\theta$ where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$. Equivalently, we have

$$\cos\theta = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||\,||\mathbf{v}||}.$$

Prop. 3. Let $\mathbf{v}$ and $\mathbf{w}$ be two vectors with non-zero dot product. The sign of $\mathbf{v} \cdot \mathbf{w}$ tells us their relative position from each other.


$v \cdot w = 0$
$v \cdot w > 0$
v
$v \cdot w < 0$
angle with v less than 90 ° in this half-plane
angle with v greater than 90 ° in this half-plane

Proof. Fix $\mathbf{v}$. Note that $\cos\theta = 0$ when $\mathbf{v}$ and $\mathbf{w}$ are perpendicular to each other, i.e., $\mathbf{v} \cdot \mathbf{w} = 0$; $\cos\theta > 0$ in the first and second quadrants, i.e., when $\mathbf{w}$ is in the same half-plane as $\mathbf{v}$; $\cos\theta < 0$ in the third and fourth quadrants, i.e., when $\mathbf{w}$ is in the opposite half-plane as $\mathbf{v}$. □

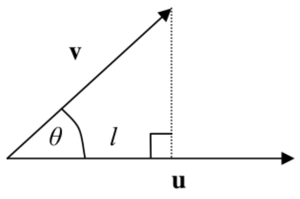
A.3. Vector Projection

The **vector projection** of a vector $\mathbf{v}$ onto a non-zero vector $\mathbf{u}$ is a vector parallel to $\mathbf{u}$, defined as $\mathbf{v}_1 = v_1\hat{\mathbf{u}}$, where v_1 is a scalar, called the **scalar projection** of $\mathbf{v}$ onto $\mathbf{u}$, and $\hat{\mathbf{u}}$ is the unit vector in the direction of $\mathbf{u}$.

Prop. 4. The scalar projection of $\mathbf{v}$ onto $\mathbf{u}$ is given by

$$||\text{Proj}_{\mathbf{u}}\mathbf{v}|| = \frac{\mathbf{v} \cdot \mathbf{u}}{||\mathbf{u}||}.$$

In other words, the component of $\mathbf{v}$ in the direction of $\mathbf{u}$ is equal to the dot product between $\mathbf{v}$ and $\mathbf{u}$, the unit vector in the direction of $\mathbf{u}$.



Proof. From the picture, we see that $\ell = \|\mathbf{v}\| \cos\theta$. Using the cosine property of dot product, we have

$$\ell = \|\mathbf{v}\| \cos\theta = \frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|}$$

as desired. □

Cor. 5. The vector projection of **v** onto **u** is given by

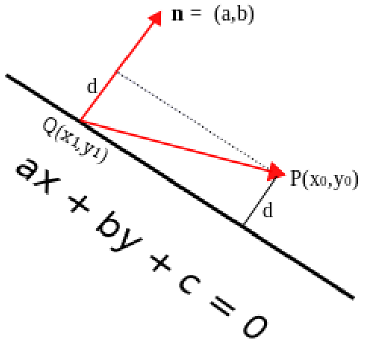
$$\text{Proj}_{\mathbf{u}}\mathbf{v} = \left(\frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|}\right) \frac{\mathbf{u}}{\|\mathbf{u}\|}.$$

Proof. By definition, the vector projection of **v** onto **u** is the unit vector **u** multiplied by scalar projection of **v** onto **u**. □

A.4. Signed Distance to Hyperplane

Prop. 6. Let $P = (x_0, y_0) \in \mathbb{R}^2$, L be the line $ax + by + c = 0$, $Q = (x_1, y_1)$ be any point on L, and **n** be the vector $\langle a, b \rangle$ starting at the point Q. The distance from point P to line L is given by

$$d = \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}}.$$



Proof. By **Prop 1**, **n** is perpendicular to L. The distance d from point P to L is equal to the length of the orthogonal projection of $\overrightarrow{QP}$ on **n**. By **Prop 5**, the length of this scalar projection is given by

$$d = \frac{\overrightarrow{QP} \cdot \mathbf{n}}{\|\mathbf{n}\|}.$$

Plugging in $\overrightarrow{QP} = \langle x_0 - x_1, y_0 - y_1\rangle$, we have

$$d = \frac{\langle x_0 - x_1, y_0 - y_1\rangle \cdot \langle a, b\rangle}{\sqrt{a^2+b^2}} = \frac{a(x_0 - x_1) + b(y_0 - y_1)}{\sqrt{a^2+b^2}}.$$

Since Q is a point on the line, $c = -ax_1 - by_1$, and so,

$$d = \frac{ax_0 + by_0 + c}{\sqrt{a^2+b^2}}.$$

□

Cor. 7. If $\mathbf{n}$ is a unit vector, then the distance from point P to line L is given by $d = ax_0 + by_0 + c$.

Cor. 8. Point P is on the line if $d = 0$, positive if it is on the same side (wrt L) as $\mathbf{n}$, and negative if it is on the other side.

A.5. Mistake Bound Theorem

Thm. 9. If there exists a number $\gamma > 0$ and a parameter vector $\mathbf{w}^*$ with $\|\mathbf{w}^*\| = 1$ such that

$$\forall n : t_n(\mathbf{x}_n \cdot \mathbf{w}^*) \geq \gamma,$$

then the perceptron algorithm makes at most R^2/γ^2 errors, where $R = \max_n \|\mathbf{x}_n\|$.

Remark.

- We can always find such an R: just look at the farthest data point from the origin.
- The margin γ is the complexity parameter that defines the *separability* of data.
- The theorem is only true when samples are linearly separable, i.e., $\gamma > 0$ exists!

Proof. Let $\mathbf{w}^k$ be the parameter vector when the algorithm makes its kth error. If the kth error is made on an example n, we have

$$\mathbf{w}^{k+1} \cdot \mathbf{w}^* = (\mathbf{w}^k + t_n\mathbf{x}_n) \cdot \mathbf{w}^* = \mathbf{w}^k \cdot \mathbf{w}^* + t_n(\mathbf{x}_n \cdot \mathbf{w}^*) \geq \mathbf{w}^k \cdot \mathbf{w}^* + \gamma$$

where the last inequality follows from the separability of dataset by a margin γ. Since $\|\mathbf{w}^1\| = 0$, it follows by induction that $\mathbf{w}^{k+1} \cdot \mathbf{w}^* \geq k\gamma$. By Cauchy-Schwarz inequality, the fact that $\|\mathbf{w}^{k+1}\| \times \|\mathbf{w}^*\| \geq \mathbf{w}^{k+1} \cdot \mathbf{w}^*$ and $\|\mathbf{w}^*\| = 1$, we have $\|\mathbf{w}^{k+1}\| \geq k\gamma$.

Next, we want to show that after k mistakes, $\|\mathbf{w}^{k+1}\|^2 \leq kR^2$. We also have

$$\begin{aligned}\|\mathbf{w}^{k+1}\|^2 &= \|\mathbf{w}^k + t_n\mathbf{x}_n\|^2 \\ &= \|\mathbf{w}^k\|^2 + t_n^2\|\mathbf{x}_n\|^2 + \underbrace{2t_n(\mathbf{x}_n \cdot \mathbf{w}^k)}_{<0 \text{ as misclassified}} \\ &\leq \|\mathbf{w}^k\|^2 + R^2\end{aligned}$$

Again by induction, $\|\mathbf{w}^{k+1}\| \leq kR^2$. Combining two inequalities yields

$$k^2\gamma^2 \leq \|\mathbf{w}^{k+1}\| \leq kR^2 \implies \forall k : k \leq \frac{R^2}{\gamma^2}. \quad \square$$

Remark. If γ is small, randomization helps.