So far (Supervised learning)

I → linear regression

- → Residual sum of squares loss
- → training through gradient descent
- → Normal equations
- → Regularization (Ridge, LASSO, Best subset)
- → Bias Variance tradeoff
- → Cross validation

II → linear classification

- → Binary classification through least squares
- → Extension to multiple classes (one vs one one vs rest, 1 of K encoding + single discrim.

→ Probabilistic classifiers

→ Discriminative classifiers $p(t \mid x)$

→ logistic regression (MLE + log loss)

→ Generative classifiers $p(x \mid t)$

→ Gaussian discriminant analysis (in particular Linear Discriminant Analysis)

→ Naive Bayes

$\text{\textcircled{III}}$ → Neural Neural Network

→ Perceptron (including training though perceptron learning rule

$\rightarrow$ Extension to Neural Networks

(including training through MLE + minimization of log loss)

Today $\rightarrow$ Kernels (in particular the kernel trick)

$\rightarrow$ Recitation (implementation of kernel classifier)

$\rightarrow$ Support vector Machines / Max Margin classifiers

Kernels

→ So far, we have discussed how to train a linear model through the RSS loss

$$\ell(\beta) = \frac{1}{N}\sum_{i=1}^{N}\left(t^{(i)} - \underbrace{(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_D x_D^{(i)})}_{\beta^T \tilde{x}^{(i)}}\right)^2$$

→ Using grad type iterations to learn the classifier, we get

$$\beta^{(k+1)} = \beta^{(k)} - \frac{2\eta}{N}\sum_{i=1}^{N}\left(t^{(i)} - (\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_D x_D^{(i)})\right)\cdot(-\tilde{x}^{(i)})$$

Where $\tilde{x}^{(i)} = [1, x^{(i)}] \in \mathbb{R}^{D+1}$

If we wanted to learn a classifier on more features (stored in the vector $\tilde{\Phi}(x^{(i)})$

e.g $\tilde{\phi}(x^{(i)}) = [1, x_2^{(i)}, \dots, x_D^{(i)}, (x_1^{(i)})^2, x_1^{(i)}x_2^{(i)}, \dots, (x_D^{(i)})^2, \dots]$

the loss and the grad iterations would turn to

$$\ell(\beta) = \frac{1}{N}\sum_{i=1}^{N}\left(t^{(i)} - \beta^T\tilde{\phi}(x^{(i)})\right)^2$$

$$\boxed{\beta^{(k+1)} = \beta^{(k)} - \frac{2\eta}{N}\sum_{i=1}^{N}\left(t^{(i)} - \beta^T\tilde{\phi}(x^{(i)})\right) \cdot -\tilde{\phi}(x^{(i)})} \quad (**)$$

Problem : for large feature vectors (needed for complex data) the grad iterations become costly (in terms of the memory storage)

As an example, if we take $\tilde{\phi}(x^{(i)})$ to encode all the monomials of degree up to 3, $\tilde{\phi}(x^{(i)}) = [1, x_1^{(i)}, \dots x_D^{(i)}, (x_1^{(i)})^2, \dots, (x_D^{(i)})^2, \dots (x_D^{(i)})^3]$

for a $D = 10^3$ the storage needed for $\tilde{\phi}(x^{(i)})$ is already on the order of 10^9.

$\rightarrow$ In order to alleviate this difficulty, we will rely on the so-called "Kernel trick"

Main idea: $\beta = \sum_{i=1}^{N} \lambda_i \tilde{\phi}(x^{(i)})$ $(*)$

if we start the grad iterations at $\beta^{(0)} = 0$ the assumption is trivially satisfied

$$\vec{0} = \beta^{(0)} = \sum_{i=1}^{N} 0 \cdot \tilde{\phi}(x^{(i)})$$

Can we then show that all $\beta^{(k)}$ will read as $\beta^{(k)} = \sum_{i=1}^{N} \lambda_i^{(k)} \tilde{\phi}(x^{(i)})$

→ Since the decomposition holds at $k=0$ to show that it holds throughout all the iterations, it suffices to show that if it holds for the kth iterate, it still holds for $\beta^{(k+1)}$

using (**) we get

$$\beta^{(k+1)} = \beta^{(k)} + \frac{2\eta}{N}\sum_{i=1}^{N}\left(t^{(i)} - \beta^T\tilde{\phi}(x^{(i)})\right)\cdot\tilde{\phi}(x^{(i)})$$

using the assumption $\beta^{(k)} = \sum_{i=1}^{N}\lambda_i^{(k)}\tilde{\phi}(x^{(i)})$

$$\beta^{(k+1)} = \sum_{i=1}^{N}\lambda_i^{(k)}\tilde{\phi}(x^{(i)}) + \frac{2\eta}{N}\sum_{i=1}^{N}\left(t^{(i)} - \beta^T\tilde{\phi}(x^{(i)})\right)\cdot\tilde{\phi}(x^{(i)})$$

$$\beta^{(k+1)} = \sum_{i=1}^{N}\left(\lambda_i^{(k)} + \frac{2\eta}{N}\left(t^{(i)} - \beta^T\tilde{\phi}(x^{(i)})\right)\right)\tilde{\phi}(x^{(i)})$$

$$\beta^{(k)} = \sum_{i=1}^{N} \lambda_i^{(k)} \tilde{\phi}(x^{(i)}) \Rightarrow \beta^{(k+1)} = \sum_{i=1}^{N} \lambda_i^{(k+1)} \tilde{\phi}(x^{(i)})$$

Where $\lambda_i^{(k+1)} = \lambda_i^{(k)} + \frac{2\eta}{N}\left(t^{(i)} - \beta^T \tilde{\phi}(x^{(i)})\right)$

From this, we see that we have derived an update rule on the λ_i's. However, once we have found the optimal λ_i's the β can be recovered easily by plugging the λ_i's inside

(*) $$\beta^* = \sum_{i=1}^{N} \lambda_i^* \tilde{\phi}(x^{(i)})$$

As a result, we have derived an alternative algorithm which can be summarized by the following 3 steps:

Step 1 Initialization $\lambda_i^{(0)} = 0 \quad \forall i$

Step 2 Update the λ_i's

$$\boxed{\lambda_i^{(k+1)} = \lambda_i^{(k)} + \frac{2\eta}{N}\left(t^{(i)} - \beta^T \tilde{\phi}(x^{(i)})\right)} \quad (*)$$

Step 3 Derive the final classifier as

$$\beta^* = \sum_{i=1}^{N} \lambda_i^* \tilde{\phi}(x^{(i)}) \quad \longrightarrow \quad y(x) = (\beta^*)^T \hat{\phi}(x)$$

→ there are 2 more difficulties that remain:

1) the updates (*) still include a reference to β

2) these updates still require us to store and process the $\tilde{\phi}(x^{(i)})$

① can be fixed by using $\beta^{(k)} = \sum_{i=1}^{N} \lambda_i^{(k)} \tilde{\phi}(x^{(i)})$

this gives updates of the form

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} + \frac{2\eta}{N}\left(t^{(i)} - \sum_{j=1}^{N} \lambda_j^{(k)} \left(\tilde{\phi}(x^{(j)})^T \tilde{\phi}(x^{(i)})\right)\right) \quad (**)$$

Fixing our first problem, we see that our update (*) now only depends on the inner product between the $\tilde{\phi}(x^{(i)})$

and that inner product will often be easier to compute and to store. Moreover, we see that this inner product does not depend on k and hence, should only be computed once before the updates.

As an illustration of the difference between storing and processing the feature vectors and computing the inner product, consider the degree 3 monomials. For these we get

$$\tilde{\phi}(x^{(i)}) = [1, \underline{x_2^{(i)}, x_2^{(i)} \dots x_D^{(i)}}, \underline{(x_1^{(i)})^2, \dots, (x_D^{(i)})^2}, \dots, (x_D^{(i)})^3]$$

$$\tilde{\phi}(x^{(i)})^T \tilde{\phi}(x^{(j)}) = 1 + \sum_{k=1}^{N} x_k^{(i)} x_k^{(j)} + \sum_{k=1}^{D} \sum_{l=1}^{D} (x_k^{(i)} x_l^{(i)} x_k^{(j)} x_l^{(j)})$$

$$+ \sum_{k=1}^{N} \sum_{l=1}^{N} \sum_{m=1}^{N} (x_k^{(i)} x_l^{(i)} x_m^{(i)} x_k^{(j)} x_l^{(j)} x_m^{(j)})$$

$$= 1 + (x^{(i)T} x^{(j)})^1 + ((x^{(i)})^T x^{(j)})^2 + ((x^{(i)})^T x^{(j)})^3$$

→ In this case $\tilde{\phi}(x^{(i)})^T \tilde{\phi}(x^{(j)})$ can be computed much more efficiently by 1) computing $(x^{(i)})^T x^{(j)}$

2) raise $(x^{(i)})^T x^{(j)}$ to the appropriate power

As we saw the inner product $\tilde{\phi}(x^{(i)})^T \tilde{\phi}(x^{(j)})$ which is central to the kernel trick only needs to be computed once at the initialization

→ In particular, we can thus store the inner products into a fixed matrix indexed by the training examples such that

$$\boxed{K(i,j) = \tilde{\phi}(x^{(i)})^T \tilde{\phi}(x^{(j)})} \quad (*)$$

More generally we are going to call "Kernel" the map $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and such that

$$K(x, y) = \phi(x)^T \phi(y)$$

Substituting $(*)$ into our update $(**)$

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} + \frac{2\eta}{N} \left(t^{(i)} - \sum_{j=1}^{N} \lambda_j K(x^{(i)}, x^{(j)}) \right)$$

or in vector form

$$(***) \quad \boxed{\vec{\lambda} = \vec{\lambda} + \frac{2\eta}{N} \left(\vec{t} - \underline{\underline{K}} \vec{\lambda} \right)}$$

where $\vec{\lambda}, \vec{t} \in \mathbb{R}^{\boxed{N}}$

$K \in \mathbb{R}^{\boxed{N \times N}}$

→ Complexity reduces from $O(D^k)$ to $O(N)$

→ Kernel trick is therefore particularly interesting for relatively small dataset for which a large number of features are needed.

Following from our final update (***) it seems interesting to take a different perspective on the learning problem based on efficient kernel matrices vs efficient feature vectors.

→ in other words, given a data set $\{x^{(i)}, t^{(i)}\}_{i=1}^{N}$, we might start to wonder what are efficient kernel matrices as opposed to what are efficient feature vectors.

Let us consider 2 examples: Ex 1

$$K(x,y) = (x^T y)^2 \qquad x, y \in \mathbb{R}^D$$

$$= \left(\sum_{i=1}^{D} x_i y_i \right)^2$$

$$= \sum_{i=1}^{D} \sum_{j=1}^{D} x_i y_i x_j y_j$$

$$= \phi(x)^T \phi(y) \quad \text{with} \quad \phi(x) = [x_1^2, x_1 x_2, x_1 x_3, \dots x_D^2]$$

$$\phi(y) = [y_1^2, y_1 y_2, y_1 y_3, \dots, y_D^2]$$

$\rightarrow$ $K(x,y) = (x^T y)^2$ has a decomposition as a product of 2 feature vectors.

Ex 2

$$K(x,y) = (x^T y + c)^2$$

$$K(x,y) = \left(\sum_{i=1}^{D} x_i y_i + c\right)^2 \leftarrow$$

$$= \left(\sum_{i=1}^{D} x_i y_i\right)^2 + c^2 + 2c\sum_{i=1}^{D} x_i y_i$$

$$= \sum_{i=1}^{D}\sum_{j=1}^{D} x_i y_i x_j y_j + c^2 + 2c\sum_{i=1}^{D} x_i y_i$$

$$= \phi(x)^T \phi(y) \qquad \phi(x) = \left[c, \sqrt{2c}\,\vec{x}, (x_i x_j)_{\substack{1\le i\le D\\ 1\le j\le D}}\right]$$

$$\phi(y) = \left[c, \sqrt{2c}\,\vec{y}, (y_i y_j)_{\substack{1\le i\le D\\ 1\le j\le D}}\right]$$

Again, $K(x,y)$ can read as a product of feature vectors

Note that when K is defined as an inner product of 2 feature vectors, it is symetric by construction

Another observation that can be derived from the assumption

$K(x,y) = \phi(x)^T\phi(y)$ for some feature vectors $\phi(x), \phi(y)$

is that K has to be positive semidefinite which means

For every $z \in \mathbb{R}^N$ we must have $z^T K z \geq 0$

to see this, note that we have

$$z^T K z = \sum_{i=1} \sum_{j=1} z_i z_j K(i,j)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j \, \phi(x^{(i)})^T \phi(x^{(j)})$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j \sum_{k=1}^{K} \phi_k(x^{(j)}) \phi_k(x^{(i)})$$

$$= \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N} z_i \phi_k(x^{(i)}) z_j \phi_k(x^{(j)})$$

$$= \sum_{k=1}^{K} \left(\sum_{i=1}^{N} z_i \phi_k(x^{(i)}) \right)^2 \geq 0$$

These 2 observations are summarized by <u>Mercer's theorem</u>

A kernel K is a <u>valid kernel</u> (meaning there exists a decomposition $K(x,y) = \phi(x)^T \phi(y)$) iff $K(x,y)$ is symmetric and positive semidefinite

a.k.a Mercer kernel.