CS5339 Machine Learning

Representation II

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 2, 2019/20

Outline

1 Linear Combination of Functions

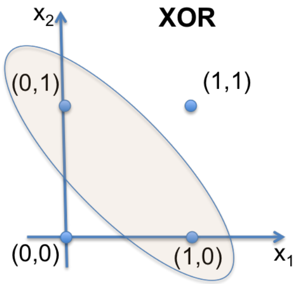
2 Kernel Method

3 Neural Networks

4 Matrix/Tensor Factorization

5 Appendix

Linear Combination of Features



- The XOR function that cannot be represented by a linear classifier can be represented by a thresholded quadratic function, which can represent an ellipse.

$$f(x_1, x_2) = \mathrm{sign}(b + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2)$$

- The quadratic function, or more generally polynomial functions, can also be viewed as a linear combination of features.
    - In this case, the features are the monomials $(x_1, x_2, x_1x_2, x_1^2, x_2^2)$.
    - A feature is just a function of the input, e.g. $g_1(x_1, x_2) = x_1, \dots, g_5(x_1, x_2) = x_2^2$ are the features in this example.
- No longer a linear function of the original input $\mathbf{x}$, but linear when viewed as a function of the features.

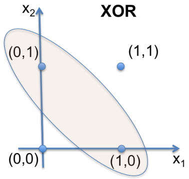
Linear Combination of Functions

- Linear combination of functions is very commonly used in machine learning:
    - Kernel methods
    - Single hidden layer neural networks
    - Matrix/tensor factorization
    - Ensemble methods

Outline

1. Linear Combination of Functions
2. Kernel Method
3. Neural Networks
4. Matrix/Tensor Factorization
5. Appendix

Feature Space Embedding



- We previously saw that XOR cannot be represented by a linear threshold function but can be represented by a thresholded quadratic function:

$$f(x_1, x_2) = \mathrm{sign}(b + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2).$$

- Construct a mapping $\psi : \mathbb{R}^2 \to \mathbb{R}^5$ as

$$\psi(x_1, x_2) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2).$$

- Then we have a linear function in the feature space:

$$f(x_1, x_2) = \mathrm{sign}(b + \mathbf{w}^T \psi(x_1, x_2)).$$

Kernel Trick

- In high dimensions, the number of monomials in a polynomial grows exponentially with degree, so polynomials that contains all monomials cannot be computed efficiently in high dimensions. Example: Degree 3 monomials in 2 variables

$x_1x_1x_1, x_1x_1x_2, x_1x_2x_1, x_1x_2x_2, x_2x_1x_1, x_2x_1x_2, x_2x_2x_1, x_2x_1x_2$ } 8 monomials some repeated

For degree k monomials in d variables d^k terms, some repeated

$\binom{k+d-1}{d-1}$ distinct terms, grows exponentially with k and with d

- We may also want to use infinite dimensional feature spaces for our embedding.

- We use the kernel trick to avoid having to compute the value of each feature.
  - *Kernel* refers to a function that computes the inner product between two elements in the feature space.
  - Given an embedding ψ of the domain space $\mathcal{X}$ into some Hilbert space, we define the kernel function $K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$.

    Example: Degree 2 polynomial in 2 variables

    $\psi(x_1, x_2) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2)$
  - Can view the kernel function as specifying the similarity between two elements in $\mathcal{X}$.
- We use kernels that can be efficiently computed so that inner products in high or even infinite dimensional feature spaces can be efficiently computed.

- Example: polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k$.
  - Denote $x_0 = x'_0 = 1$ to simplify notation.

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{x}') &= (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle) \cdots (1 + \langle \mathbf{x}, \mathbf{x}' \rangle) \\
&= \left(\sum_{j=0}^{d} x_j x'_j \right) \cdots \left(\sum_{j=0}^{d} x_j x'_j \right) \\
&= \sum_{J \in \{0,\ldots,d\}^k} \prod_{i=1}^{k} x_{J_i} x'_{J_i} \\
&= \sum_{J \in \{0,\ldots,d\}^k} \prod_{i=1}^{k} x_{J_i} \prod_{i=1}^{k} x'_{J_i}
\end{aligned}
$$

Example

$k = 2$ in 2d

$(1 + \langle x, x' \rangle)^2$

$= (x_0 x'_0 + \langle x, x' \rangle)^2$

$= (x_0 x'_0 + x_1 x'_1 + x_2 x'_2) \cdot$

$(x_0 x'_0 + x_1 x'_1 + x_2 x'_2)$

$= x_0 x'_0 x_0 x'_0 + x_0 x'_0 x_1 x'_1$

$+ \ldots + x_0 x_1 x'_0 x'_1$

$= x_0 x_0 x'_0 x'_0$

$+ \ldots$

- If we define $\psi(\mathbf{x})$ to contain all monomials up to degree k, i.e. for every $J \in \{0, \ldots, d\}^k$, $\psi(\mathbf{x})$ contains the element $\prod_{i=1}^{k} x_{J_i}$, we have $K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$.

- Example: Gaussian Kernel on $\mathbb{R}$.
  - Consider mapping ψ where for each $n \geq 0$, $\psi(x)_n = \frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n$. Then $\left\langle \left(e^{-x^2/2}, e^{-x^2/2}x, \dots\right), \left(e^{-x'^2}, \dots\right)\right\rangle$

$$\begin{aligned}\langle \psi(x), \psi(x')\rangle &= \sum_{n=0}^{\infty} \left(\frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n\right)\left(\frac{1}{\sqrt{n!}} e^{-\frac{(x')^2}{2}} (x')^n\right) \\ &= e^{-\frac{x^2+(x')^2}{2}} \underbrace{\sum_{n=0}^{\infty} \left(\frac{(xx')^n}{n!}\right)}_{e^{-xx'} \text{ by defn of exp}} \\ &= e^{-\frac{\|x-x'\|^2}{2}}.\end{aligned}$$

- More generally in $\mathbb{R}^d$, given $\sigma > 0$, the Gaussian kernel is defined as

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma}},$$

where σ behaves like a scale parameter.
  - The Gaussian kernel implements the following features: for any n and any monomial of order k there exists an element of $\psi(\mathbf{x})$ that equals $\frac{1}{\sqrt{\sigma n!}} e^{\frac{\|\mathbf{x}\|^2}{2\sigma}} \prod_{i=1}^{n} x_{J_i}$.

Exercise 1

In this exercise, we compare linear with polynomial SVM and RBF SVM on the handwritten digit dataset.

Run the experiment and comment on the results.

- Nonlinear kernels helpful for this task
- Interactions among pixels in diff positions likely discriminating

Hilbert Space

- Given a mapping ψ that maps to a finite feature space, we can then compute inner product and distances in the feature space like we do in $\mathbb{R}^d$.
- But what if the mapping ψ maps the instances to infinite length feature vectors?
- We would like to extend the notion of $\mathbb{R}^d$ to infinite dimensional vector and even function spaces: *Hilbert spaces*.

- We will consider real (as opposed to complex) Hilbert spaces.
- A Hilbert space generalizes the Euclidean space and allows us to extend most of the properties of Euclidean space to more general spaces.
- To define a Hilbert space, first we need to define inner product $\langle u, v\rangle$. For Euclidean space : $\langle x, y\rangle = \sum_{i=1}^{d} x_i y_i$
- The inner product has the following properties:
  - Symmetry: $\langle u, v\rangle = \langle v, u\rangle$
  - Linearity in its first argument:

$$\langle av + bu, w\rangle = a\langle v, w\rangle + b\langle u, w\rangle.$$

  - The inner product af an element with itself is positive definite: $\langle v, v\rangle \geq 0$ and $\langle v, v\rangle = 0 \Leftrightarrow v = 0$

- A vector space with an inner product is an inner product space. spae of linear combinations
- The norm of an element is $\|v\| = \sqrt{\langle v, v\rangle}$.

  In Euclidean space

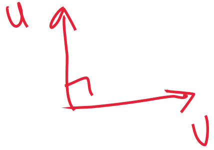
  $$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad \|v\| = \sqrt{v_1^2 + v_2^2}$$

- The distance between two elements is defined in terms of the norm:

$$d(u, v) = \|u - v\| = \sqrt{\langle u - v, u - v\rangle}.$$

- The distance satisfies the axioms of a metric:
  - $d(u, v) \geq 0$ (non-negativity)
  - $d(u, v) = 0$ if and only if $u = v$
  - $d(u, v) = d(v, u)$ (symmetry), and
  - $d(u, v) \leq d(u, w) + d(v, w)$ (triangle inequality).

- The inner product allows us to define orthogonality: u and v are orthogonal if $\langle u, v \rangle = 0$.



$\langle u, v \rangle = 0$

- Another useful fact is the Cauchy-Schwarz inequality: $|\langle u, v \rangle| \leq \|u\| \|v\|$.

- A Hilbert space H is *complete* inner product space, i.e. every Cauchy sequence converges to a point in H.
    - A sequence $v_1, v_2, \ldots$ is a Cauchy sequence if for every $\epsilon > 0$, there exists an integer N, such that for all $m, n > N$, $d(v_m, v_n) < \epsilon$.

Representer Theorem

- We have seen that kernels allow inner products to be done efficiently in very large or even infinite dimensional feature spaces.
- The *representer theorem* shows how to use inner products for solving many learning tasks. (Proof in Appendix)

Theorem (Representer Theorem): Let ψ be a mapping from $\mathcal{X}$ to a Hilbert space. Then there exists a vector $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i)$ is an optimal solution to

number of examples

linear comb of features of training set

$$\min_{\mathbf{w}} (f(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{w}, \psi(\mathbf{x}_m)\rangle) + R(\|w\|)),$$

Regularized loss

← Hilbert space norm

where $f : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically nondecreasing function.

. c.f. support vectors in SVM

Implications of the Representer Theorem

- The inner product can be written as

linearity

$$\begin{aligned}\langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle &= \left\langle \sum_{j=1}^{m} \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \right\rangle = \sum_{j=1}^{m} \langle \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i)\rangle \\ &= \sum_{i=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i).\end{aligned}$$

  - Computing the value of a linear function is just taking the inner product between the weight vector and the input features.

Implications of the Representer Theorem

- Continued ...
  - Example: the 2nd degree polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^2$ in $\mathbb{R}^2$ gives
  $$\psi((x_1, x_2)) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2).$$
  - Using the representer theorem, the weight vector is represented as
  $$\mathbf{w} = \sum_i \alpha_i (1, \sqrt{2}x_{i,1}, \sqrt{2}x_{i,2}, \sqrt{2}x_{i,1}x_{i,2}, x_{i,1}^2, x_{i,2}^2).$$
  - Computing the linear function $\langle \mathbf{w}, \psi(\mathbf{x}) \rangle$ gives
  $$\begin{aligned} \langle \mathbf{w}, \psi(\mathbf{x}) \rangle &= \sum_i \alpha_i (1 + 2x_{i,1}x_1 + 2x_{i,2}x_2 + 2x_{i,1}x_{i,2}x_1x_2 \\ &\qquad + x_{i,1}^2 x_1^2 + x_{i,2}^2 x_2^2) \\ &= \sum_i \alpha_i (1 + \langle \mathbf{x}_i, \mathbf{x} \rangle)^2 = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) \end{aligned}$$

Implications of the Representer Theorem

- In the Hilbert space, the squared norm is

$$\begin{aligned}\|\mathbf{w}\|^2 &= \langle \mathbf{w}, \mathbf{w} \rangle = \left\langle \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \psi(\mathbf{x}_j) \right\rangle \\ &= \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \boldsymbol{\psi}(\mathbf{x}_i), \boldsymbol{\psi}(\mathbf{x}_j) \rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

  - Example: for second degree polynomial kernel, $\mathbf{w} = \sum_i \alpha_i (1, \sqrt{2}x_{i,1}, \sqrt{2}x_{i,2}, \sqrt{2}x_{i,1}x_{i,2}, x_{i,1}^2, x_{i,2}^2)$ and

$$\begin{aligned}\|\mathbf{w}\|^2 = \sum_{i,j=1}^{m} \alpha_i \alpha_j (1 + 2x_{i,1}x_{j,1} + 2x_{i,2}x_{j,2} + 2x_{i,1}x_{i,2}x_{j,1}x_{j,2} \\ + x_{i,1}^2 x_{j,1}^2 + x_{i,2}^2 x_{j,2}^2) \\ = \sum_{i,j=1}^{m} \alpha_i \alpha_j (1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^2 = \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)\end{aligned}$$

Implications of the Representer Theorem

- Instead of optimizing

$$\mathbf{f}(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{w}, \psi(\mathbf{x}_m)\rangle) + R(\|w\|),$$

we can optimize

$$\mathbf{f}\left(\sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_1), \ldots, \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_m)\right) + R\left(\sqrt{\sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)}\right).$$

with respect to the α_j's instead of $\mathbf{w}$.

  - m dimensional instead of very large (or infinite) dimensional.

$\uparrow$ training set size

Implications of the Representer Theorem

- For example, using kernels, the objective function for soft SVM becomes

$$\sum_{i=1}^{m} \max\left\{0, 1 - y_i\left(\sum_{j=1}^{m}\alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)\right\} + \lambda \overbrace{\sum_{i,j=1}^{m}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j)}^{\|w\|^2}.$$

Optimize over α_j, $j=1\ldots m$

- Regularized logistic regression becomes

$$\sum_{i=1}^{m} \log\left(1 + y_i\left(\sum_{j=1}^{m}\alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)\right) + \lambda \sum_{i,j=1}^{m}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j).$$

- For example, using kernels, the objective function for ridge regression becomes

$$\sum_{i=1}^{m}\left(y_i - \sum_{j=1}^{m}\alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right)^2 + \lambda \sum_{i,j=1}^{m}\alpha_i\alpha_j K(\mathbf{x}_i, \mathbf{x}_j).$$

Implications of the Representer Theorem

- Often the objective function is convex, e.g. in the three cases, allowing efficient learning.
- We only need to know the value of the $m \times m$ matrix G such that $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, often called the *Gram* matrix for solving these problems, e.g. Soft SVM can be written as

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} \left(\lambda \boldsymbol{\alpha}^T G \boldsymbol{\alpha} + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i (G\boldsymbol{\alpha})_i\} \right),$$

  where $(G\boldsymbol{\alpha})_i$ s the i-th element of the vector obtained from multiplying G with $\boldsymbol{\alpha}$.

Representation Properties

- With the use of kernel, many techniques for linear functions carry over to non-linear functions, e.g. polynomials.
  - Many techniques for linear functions only access the features using inner products, kernels allow the inner products to be computed in reasonable time
- Certain kernels give rise to universal approximators. In particular, the Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma}}$ gives rise to a universal approximator.

Gram matrix $m \times m$

- Useful for moderate sized datasets
  - Computing kernels for all pairs grows quadratically. Need further approximation techniques to scale to very large datasets.

Exercise 2:
In this exercise, we will compare two ways of representing polyonomial predictors using kernel methods.

I. Expand out the features as a vector of monomials. Represent the weights of the monomials with a weight vector $\mathbf{w}$. Compute $\langle \mathbf{w}, \psi(\mathbf{x})\rangle$ by directly computing the weighted sum of the features.

II. Store the vector $\boldsymbol{\alpha} \in \mathbb{R}^m$ which represents $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ and compute the prediction as $\langle \mathbf{w}, \psi(\mathbf{x})\rangle = \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x})$ where $K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}'\rangle)^k$.

Let $\mathbf{x} \in \mathbb{R}^d$. Let k be the degree of the polynomial, where $k = 1$ gives the linear kernel. Which of the following is true?

A. (I) is better for both $k = 1$ and for large k.

B. (II) is better for both $k = 1$ and for large k.

[✓ C.] (I) is better for $k = 1$ but (II) is better for large k

D. (I) is better for large k but (II) is better for $k = 1$.

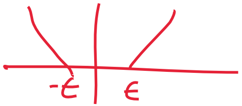
① $\langle w, \psi(x) \rangle$ expand to $\psi(x)$ linear fn with $\psi(x)$

② kernel method

$\langle w, \psi(x) \rangle = \sum_{j=1}^{m} \alpha_j k(x_j, x)$

$k(x, x') = (1 + \langle x, x' \rangle)^k$

- When $k=1$, $\langle w, \psi(x) \rangle$ takes $O(d)$ time and at least md operation for ②
- when k is large, ① takes time exponential in k but ② takes time $c\, md \lg k$.



Exercise 3:
Support vector regression (SVR) uses the ϵ-insensitive loss function which does not penalize points within ϵ of the true value and penalizes points linearly beyond that.

In the experiment (from `http://scikit-learn.org/stable/auto_examples/plot_kernel_ridge_regression.html`), we compare SVR with kernel ridge regression (KRR) which uses the square rather than ϵ-insensitive loss. Provide your answer to the questions below before running the experiment.

square loss

For training, which of SVR or KRR will be faster, and why? For prediction, which of SVR or KRR will be faster and why?

- for testing, SVR is faster due to sparsity
- for training, for large data sets SVR may be faster because of sparsity. For small data sets KRR may be faster by exploiting closed form solution

Exercise 4:

The representer theorem allows us to represent the weight vector as a linear combination of the feature vectors of training instances. We now investigate the consequence of that for feature sparsity.

Consider learning a weight vector of as a linear combination of the training instances when the instances $(x_1, x_2) \in \{-1, 1\}^2$. Let the true function be positive when x_1 takes the value 1 and negative otherwise. Consider the training set consist of single positive instances $(1, -1)$ and a single negative instance $(-1, 1)$. Discuss whether learning a sparse weight vector where only the first component is non-zero is possible.

For feature sparsity, try ~~re~~ reg with $\|w\|_1$

$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = w = \alpha_1 \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \alpha_2 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Want $w_2 = 0$ $\quad -\alpha_1 + \alpha_2 = 0$

$\Rightarrow \alpha_1 = \alpha_2$

$w_1 = \alpha_1 - \alpha_2 = 0$

Constructing Kernels by Defining Inner Products

- Instead of constructing features ψ directly, we can sometimes define an algorithm for computing $\langle\psi(\mathbf{x}), \psi(\mathbf{x}')\rangle$ instead.
  - As it is an inner product, it immediately gives us a kernel.
  - Of course, this is helpful mainly when $\langle\psi(\mathbf{x}), \psi(\mathbf{x}')\rangle$ can be computed quickly.
- **Example:** Assume we want to find a sequence of characters in a file that indicate whether the file contains a virus – a "signature".
  - We do not know what the signature is, otherwise there is no need to learn.
  - Given a training set of positive and negative examples, we want to build a classifier that will classify a file as positive if the signature is present, and negative otherwise.

- Continued ...
  - Let $\mathcal{X}_d$ denote the set of all strings of length at most d over some alphabet set Σ.
    - For example, if $\Sigma = \{a, p, l, e\}$ and $d = 3$ then elements of $\mathcal{X}_3$ includes the strings "a", "ap", "app", "ppl", "ple", "pl", ...
  - The set of features is then $\psi = \{\psi_v : v \in \mathcal{X}_d\}$, where $\psi_v(x)$ is an indicator feature that takes the value 1 if v is a substring of x and 0 otherwise.
    - In the example, ψ would contain elements such as ψ_{app}, ψ_a, ψ_{ap}, ψ_{ple}, ...
    - For the file $\mathbf{x} = apple$, the features $\psi_a(\mathbf{x})$, $\psi_{ap}(\mathbf{x})$, $\psi_{app}(\mathbf{x})$, $\psi_p(\mathbf{x})$, $\psi_{pp}(\mathbf{x})$, $\psi_{ppl}(\mathbf{x})$, $\psi_{pl}(\mathbf{x})$, $\psi_{ple}(\mathbf{x})$, $\psi_l(\mathbf{x})$, $\psi_{le}(\mathbf{x})$, and $\psi_e(\mathbf{x})$ will take the value 1 while all other features will take value 0.

$$\psi(x) = \begin{bmatrix} \chi_a(x) \\ \chi_{ab}(x) \\ \chi_{abb}(x) \\ \vdots \\ \vdots \end{bmatrix}$$

- Continued ...
  - We can use a mapping ψ from the input to $\mathbb{R}^s$, where $s = |\mathcal{X}_d|$ such that each coordinate of ψ correspond to an element of the vector i.e. $\psi(x)$ is a vector in $\{0,1\}^{|\mathcal{X}_d|}$.
  - However, the feature space size s is exponentially large in d. Enumerating the features would not be effective.
  - In contrast, the inner product $K(\mathbf{x}, \mathbf{x}')$ simply counts the number of common substrings in $\mathbf{x}$ and $\mathbf{x}'$.
    - There are fewer than $d|\mathbf{x}|$ number of substrings of length ~~no more than~~ d.
    - Counting the number of common substrings can be done in polynomial time.

- Continued ...
    - We just described a type of *string kernel* that counts the number of common substrings between two strings.
    - Back to our original problem, a function which detects the presence of a virus (substring) can be represented with a single weight on the indicator of the substring and a threshold, hence can be learned using the kernel.

- In the example, computing the kernel is relatively simple. We can even efficiently maintain a sparse weight vector $\mathbf{w}$ – substrings that have never been observed in the training data do not need to be kept.

- We can define more complex features in strings
  - For example, indicator features for subsequence instead of substring. A string s is a subsequence of s' if we can make $s' = s$ by deleting some elements in s'.
  - In this case, we need a more complex dynamic programming algorithm for counting the number of common subsequences as the inner product, and maintaining a weight vector $\mathbf{w}$ directly no longer seem effective.
- We can also define kernels over other structured objects
  - Tree kernels that count the number of common subtrees or the number of common tree fragments between two trees.

Exercise 5:
We have shown that the string kernel we constructed can be used to learn to detect the presence of a virus (substring). How do we use it to detect the presence of any virus from a set of viruses?

- Implement OR function of the indicators
- Can represent OR.

Characterizing Kernel Functions

- We have seen how we construct a kernel by constructing a mapping to feature space, then defining the kernel as the inner product between features.
- Assume we are given a similarity function of the form $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. When is it a kernel, i.e. represent an inner product between $\psi(\mathbf{x})$ and $\psi(\mathbf{x}')$ for some feature mapping ψ?

- We will need the notion of a positive semidefinite matrix. A $n \times n$ matrix A is positive semidefinite if $\mathbf{x}^T A\mathbf{x} \geq 0$ for every nonzero vector $\mathbf{x}$.

Example : $A = \begin{bmatrix} 5 & -11 \\ 11 & 25 \end{bmatrix}$

$x = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$ $x^T A x = \begin{bmatrix} -2 & 3 \end{bmatrix} \begin{bmatrix} 5 & -11 \\ 11 & 25 \end{bmatrix} \begin{bmatrix} -2 \\ 3 \end{bmatrix}$

$= \begin{bmatrix} -2 & 3 \end{bmatrix} \begin{bmatrix} 23 \\ 53 \end{bmatrix} = -46 + 159 > 0$

- Given m data points $\mathbf{x}_1, \ldots, \mathbf{x}_m$, we call the $m \times m$ matrix G such that $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ the *Gram* matrix.

Ex : $x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $x_2 \begin{bmatrix} 3 \\ 4 \end{bmatrix}$

linear kernel $G = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix}$

Lemma: A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implements an inner product in some Hilbert space if and only if it is positive semidefinite; i.e. for all $\mathbf{x}_1, \ldots, \mathbf{x}_m$, the Gram matrix $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ is a positive semidefinite matrix.

Proof in the Appendix.

- The lemma can be used to show that a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel.
- Also the Gram matrix being positive semidefinite is useful for optimizing the regularized loss

$$\sum_{i=1}^{m} \ell\left(y_i, \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right) + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

convex in α

  - $\sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$ is convex in $\boldsymbol{\alpha}$ if G is positive semidefinite, and
  - if the loss function $\ell(y_i, \cdot)$ is convex in $\boldsymbol{\alpha}$ as well, the regularized loss is also convex.

Constructing Kernels

The following properties allow valid kernels to be constructed from other valid kernels:

- If k_1, k_2 are kernels, $\alpha_1 k_1 + \alpha_2 k_2$ is a kernel for $\alpha_1, \alpha_2 \geq 0$.
- If k_1, k_2 are kernels, $(k_1, k_2)(x, x') = k_1(x, x')k_2(x, x')$ is also a kernel.
- For any mapping $h : \mathcal{X} \to \mathcal{X}$ and kernel k_1, $k(x, x') = k_1(h(x), h(x'))$ is a kernel.
- If g is a polynomial with positive coefficients and k_1 is a kernel, $k(x, x') = g(k_1(x, x'))$ is a kernel.
- If k_1 is a kernel, $k(x, x') = \exp(k_1(x, x'))$ is a kernel.

Kernel Methods in Practice

- For small to moderate data set sizes, SVM with non-linear kernels (e.g. Gaussian, polynomial) are often quite effective in practice.
  - Convex optimization, relatively easy to train with fewer parameters to tune.
  - Regularization (large margin) methods fairly effective at preventing overfitting.
- For very large data sets, scaling becomes an issue. If kernel at each data point is used, Gram matrix would not fit into memory – use iterative methods. Sparsity helps.
- For the input, usually helpful to tranform categorical variables into dummy/one hot encoding. Scaling/normalization is usually helpful.
- Custom designed kernels such as the string kernel is sometimes useful.

Outline

1 Linear Combination of Functions

2 Kernel Method

3 Neural Networks

4 Matrix/Tensor Factorization

5 Appendix

Single Hidden Layer Neural Networks

- Instead of fixed features, it is also natural to parameterize the features and learn the parameters.
- This can be represented as a single hidden layer neural network.

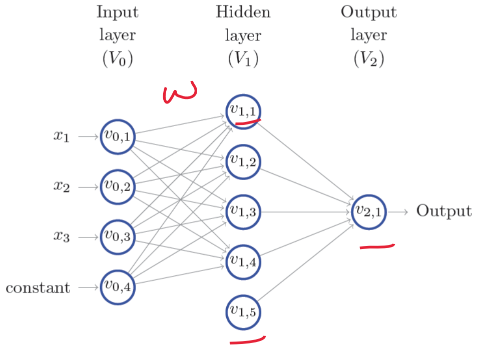
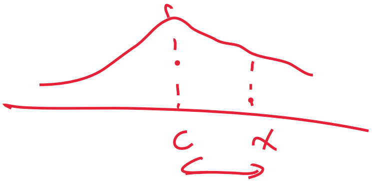
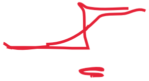


Figure from SSBD.

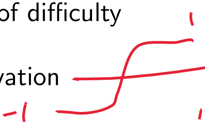
- The functions are usually represented in the form $f(\mathbf{x}) = b + \sum_{i=1}^{d} w_i \sigma(\mathbf{v}_i^T \mathbf{x})$
- The function $\sigma(\mathbf{v}_i^T \mathbf{x})$ is often called a hidden unit.
- The function $\sigma(\cdot)$ is called an *activation function*.
- Other functions such are sometimes used as hidden units instead of $\sigma(\mathbf{v}_i^T \mathbf{x})$
  - If the output of the hidden unit depends only on the distance to a center $\phi(\|\mathbf{x} - c\|)$, where c is a parameter, it is often called a radial basis function.
  - One common function to use for a radial basis function is a Gaussian $\phi(\|\mathbf{x} - c\|) = \exp(\beta\|\mathbf{x} - c\|^2)$.

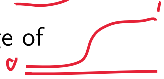




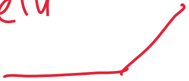
- Commonly used functions for hidden units include:
  - Linear threshold function: $\sigma(\mathbf{x}, \mathbf{v}) = \mathrm{step}(\mathbf{v}^T\mathbf{x} + \mathbf{v}_0)$, where the step function is 1 if the input is greater than or equal to 0 and 0 otherwise. Not actually used in practice because of difficulty in training, but easier to analyse.
  - Linear functions composed with sigmoid: tanh activation $\sigma(\mathbf{x}, \mathbf{v}) = \tanh(\mathbf{v}^T\mathbf{x} + \mathbf{v}_0)$ or logistic activation $\sigma(\mathbf{x}, \mathbf{v}) = 1/(1 + e^{-(\mathbf{v}^T\mathbf{x}+\mathbf{v}_0)})$. These have the advantage of being differentiable, so enabling gradient type learning methods.
  - Rectifier linear function: $\sigma(\mathbf{x}, \mathbf{v}) = \max(0, \mathbf{v}^T\mathbf{x} + \mathbf{v}_0)$. This has become very popular in recent neural network architectures, particularly for deep networks.

easy to analyse



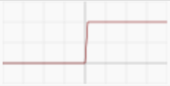
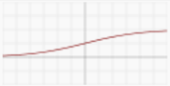
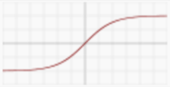
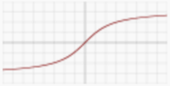
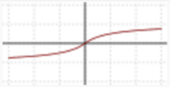
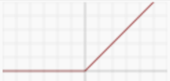


Relu



easier to train

Common activation functions (Wikipedia):

Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Rectifier[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

Representing All Boolean Functions

- We will analyse linear threshold activation functions. For Boolean functions (and classification) we threshold the output unit as well.
- Consider networks consisting of only AND, OR, and NOT gates.
- For such networks, if we consider $\neg x$ as input (rather than a hidden unit) when we have input x, a single hidden layer network is sufficient to represent all functions.
  - Can use the Disjunctive Normal Form (DNF): an OR gate as output with AND gates as the first hidden layer.
  - Can also use the Conjunctive Normal Form (CNF): an AND gate as output with OR gates as the first hidden layer.
- Since a linear threshold unit can represent AND and OR gates, a single hidden layer neural network with linear threshold units can represent all Boolean functions as well (with negated inputs added).

Representing PARITY

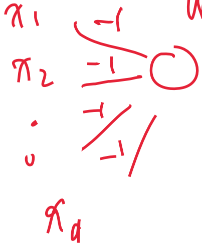
It is known that constant depth circuits using AND, OR, and NOT gates, including DNF and CNF requires size exponential in the input dimension d to represent PARITY.

The following theorem shows that allowing a more powerful linear threshold unit representation can reduce the size to linear in d.

Theorem: A single hidden layer neural network with linear threshold hidden and output units can represent the PARITY function using $d + 1$ hidden units.

Proof:

- We first argue that a linear threshold function can represent the indicator function $\mathbb{1}_k$ of the set $\{\mathbf{x} : \sum_{i=1}^n x_i \cancel{\geq} k\}$ $\leq k$, i.e. the linear threshold function fires when the number of ones in the input is k or fewer.
    - To represent that function, simply set all weights to -1 and threshold to ~~$(k - 0.5)$.~~ $-(k+0.5)$



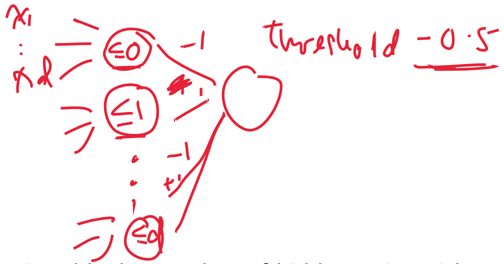
Threshold $= -(k+0.5)$

Example: $k = 5$

Threshold $= -5.5$

Unit fire when $\#$ 1 in input ≤ 5

- To represent the parity function, we simply set the second layer weights of $\mathbb{1}_k$ to 1 when k is odd and to -1 when k is even, for $0 \leq k \leq d$. The threshold for the output unit is set to -0.5.



  - If the parity is odd, the number of hidden units with weight 1 that fires exactly cancels out the number of hidden units with weights -1 that fires.
  - If the parity is even, the number of hidden units with weight -1 that fires is one more than the number of hidden units with weights 1 that fires. □

Radial basis function networks provide another example of exponential gain in representation size for PARITY.

- To represent PARITY in d dimension using Gaussian kernels with centers fixed at $\{0,1\}^d$, $2^d - 1$ non-zero coefficients are required [1].
  - This applies to kernel methods as their centers are fixed at data points.
- If the centers of the Gaussians and the bandwidth σ can be optimized, can represent PARITY using $d+1$ Gaussians [1].
- Optimizing the parameters can increase representational power, but some of the nice properties, e.g. convex learning problem, representer theorem, no longer holds. This is also done in practice and called radial basis function networks.

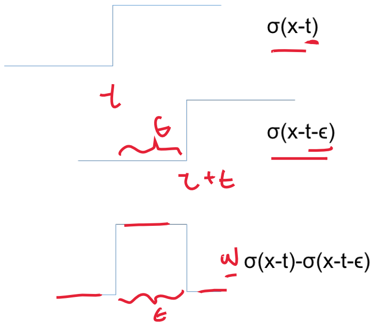
Approximating Real-valued Functions

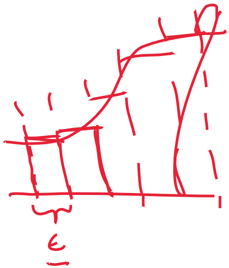
We now consider what real-valued functions a single hidden layer neural networks can represent:

$$f(\mathbf{x}) = b + \sum_{i=1}^{k} w_i \sigma(\mathbf{v}_i^T \mathbf{x} + v_{i0}),$$

where k may be arbitrarily large.
This class of functions are **universal approximators** under fairly mild conditions.

We will build some intuition with linear threshold functions. The sigmoid function can approximate the linear threshold function to arbitrary accuracy by using large enough weights.





- Consider approximating a one dimensional continuous function.
- We can construct a function that is constant on an interval and zero elsewhere using the difference of two linear threshold functions as shown.
- By partitioning the line into intervals, we can uniformly approximate the continuous function over a bounded domain using a single hidden layer NN.

- To extend to continuous function over $[0,1]^d$, we note that continuous functions over $[0,1]^d$ can be uniformly approximated by linear combinations of sinusoids $\cos(\mathbf{w}^T\mathbf{x}+b)$ (Fourier transform, as a consequence of Stone-Weiertrass theorem).
- We can uniformly approximate each of the sinusoids using linear threshold functions – use the same weight $\mathbf{w}$ as the sinusoid, then approximate as a one dimensional function.
- Hence linear combination of linear threshold functions can uniformly approximate continuous functions over $[0,1]^d$.

Exercise 6:

In this experiment, we will try to learn PARITY with a single hidden layer neural network. We will use input size of 10 and 11 hidden units (which we know to be sufficient to represent PARITY). The training set and test set sizes are 1000. We do 10 runs with random initialization using the lbfgs solver in Scikit Learn.

Before running the algorithm, write down what you expect to see.

Change some of the parameters, e.g. train_size to 10000, the solver to 'sgd' and 'adam', the activation function to 'relu', 'tanh', the number of hidden units, etc. and rerun.

- Possible to represent PARITY with small network, but difficult to optimize.

Representational Properties Summary

- Single hidden layer neural networks are universal approximators for many types of hidden units.
- When the hidden units are learnable, exponential gains in size of representation is sometimes achieved, compared to fixed features, for some functions. In particular, for PARITY
  - Linear threshold units compared to AND and OR gates
  - Radial basis functions with fixed centers, compared to adjustable centers and variances.
  - Unclear how practically important this is.

Single Hidden Layer NN in Practice

- Reasonable method when non-linear classifier/regressor required.
- Optimization is non-convex, so may depend on initialization. Harder to tune compared to kernel methods.
- For smaller dataset, non-convex optimization may be slower in practice than convex methods.
- For very large datasets, stochastic gradient used with neural networks may scale better than kernel methods, but does not provide guarantee.
- Usually good to preprocess categorical variables using dummy/one hot encoding, and rescale/normalize inputs.
- For some applications, e.g. in computer vision, specialized deeper achitectures using networks such as convolutional neural networks often work better.

Outline

1 Linear Combination of Functions

2 Kernel Method

3 Neural Networks

4 Matrix/Tensor Factorization

5 Appendix

Unsupervised Learning of Features

- In this section, we switch from using supervised learning for constructing features, e.g. in neural networks, to using unsupervised learning to learn features from unlabeled data.
    - We will look at dimension reduction, which can help supervised learning methods that are sensitive to irrelevant features.
    - Unsupervised learning can also be used to bring in information from large amounts of unlabeled data to help supervised learning when the amount of labeled data is small.

Principal Component Analysis

- Consider an autoencoder with a function pair: an encoder $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ to compress the information and a decoder $\phi : \mathbb{R}^k \mapsto \mathbb{R}^d$ to reconstruct the data from the compressed representation.

$\psi(x) = u \quad x \in \mathbb{R}^d, \quad u \in \mathbb{R}^k$

$\phi(u) = \hat{x} \quad u \in \mathbb{R}^k, \quad \hat{x} \in \mathbb{R}^d$

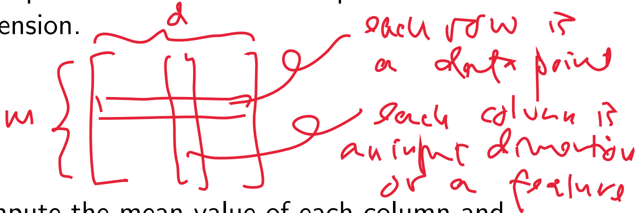
- In PCA, $\psi(\mathbf{x}) = V_k^T \mathbf{x}$ is a linear transformation. The decoder $\phi(\mathbf{u}) = V_k \mathbf{u}$ is also a linear transformation.

$\psi(x) = k\{\underbrace{\left[\ V_k^T\ \right]}_{d} \underbrace{\left[x\right]}_{d} \qquad \phi = d\{\overbrace{\left[\ V_k\ \right]}^{k} \left[u\right]\}k$

- The aim is to minimize the reconstruction error: $\sum_i \|\mathbf{x}_i - \phi(\psi(\mathbf{x}_i))\|^2$.

- Consider the data as a $m \times d$ matrix X where each row represents a data point and each column represents the values of an input dimension.

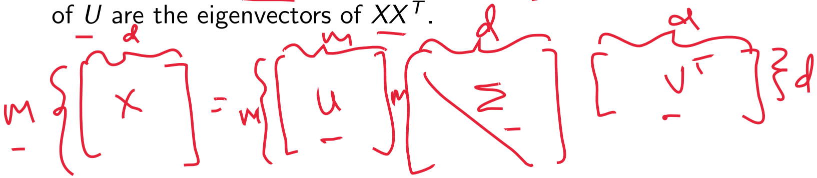
data matrix X



- In PCA, we compute the mean value of each column and remove the mean from each entry. The mean vector of the new data matrix (still called X for convenience) is now the zero vector.
- We then do a singular value decomposition (SVD) of the matrix $X = U\Sigma V^T$.

- Properties of SVD, $X = U\Sigma V^T$ (see background material on linear algebra for proofs):
  - U is a $m \times m$ orthonormal matrix, i.e. $U^T U = I$. The columns of U are the eigenvectors of XX^T.

$[u^T][u] = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$



$V^T V = I$

  - V is a $d \times d$ orthonormal matrix. The column of V are the eigenvectors of $X^T X$.
    - Dividing $X^T X$ by m we get (an estimate of) the covariance matrix. V gives the eigenvectors of the covariance matrix.
    - V can also be computed via eigendecomposition of $X^T X$ instead of using SVD.

$X = U\Sigma V^T$

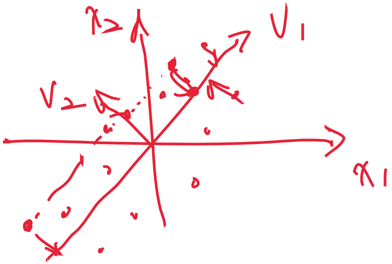
- Σ is a diagonal $m \times d$ matrix. The diagonal entries σ_i are called the singular values of X and are commonly listed in descending order.

$$m \left\{ \overbrace{\begin{bmatrix} \sigma_1 & & & 0 \\ 0 & \sigma_2 & \ddots & \sigma_m \end{bmatrix}}^{d} \right.$$

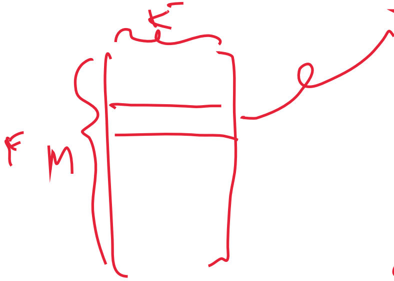
- $X_k = U_k \Sigma_k V_k^T$, formed by zeroing out all but the largest k singular values, gives the best rank k approximation of X in the Frobenius norm ($\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{d} a_{ij}^2}$). The approximation error is $\sqrt{\sum_{i>k} \sigma_i^2}$.

- Can be computed fairly efficiently ($O(md^2 + m^2 d + d^3)$, faster with approximations).

- The eigenvectors V_k are called the principle axes or directions of the data. $XV = U\Sigma V^T V$
- Given $X = U\Sigma V^T$, $XV = U\Sigma$ gives the projection of the data on the principle axes V, called the principle components.
  - The first principal component $\mathbf{v}_1$ gives the direction where the projection has highest variance, i.e. $\|X\mathbf{v}_i\|^2 = \sigma_1^2$ is the highest for any unit vector $\mathbf{v}$.
  - Correspondingly the i-th principal component $\mathbf{v}_i$ gives the direction of i-th highest variance.



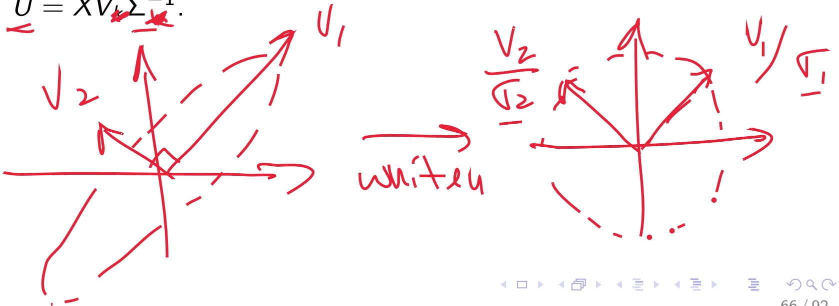
- $F = XV_k$ is a $m \times k$ matrix that gives the first k principle components for each of the data points.
  - Row i of F is the feature vector for instance i.
  - For a new d-dimensional data point $\mathbf{x}$, we can compute the principle components (projection on space spanned by V) as $\hat{\mathbf{u}} = V_k^T x$, a k-dimensional vector.
  - When reconstructed as $\hat{X} = XV_kV_k^T$ we get the best rank k reconstruction of X in terms of the Frobenius norm.



rows of coffe of x_i in new coord system, projection x_i on $v_1, \dots v_k$

Whitening

- The principle components $\overset{F}{\underline{XV}} = \underline{U\Sigma}$ have unequal variance.
- Sometimes desirable to normalize each principle component (feature) to have the same variance – called $\underline{\text{whitening}}$.
- To normalize, divide component i by $\underline{\sigma_i}$, the i-th singular value (std dev of the i-th component).
- The normalized matrix of principle components is $\underline{U} = XV\underline{\Sigma^{-1}}$.



v_1
v_2
whiten
$\frac{v_2}{\underline{\sigma_2}}$
$\frac{v_1}{\underline{\sigma_1}}$

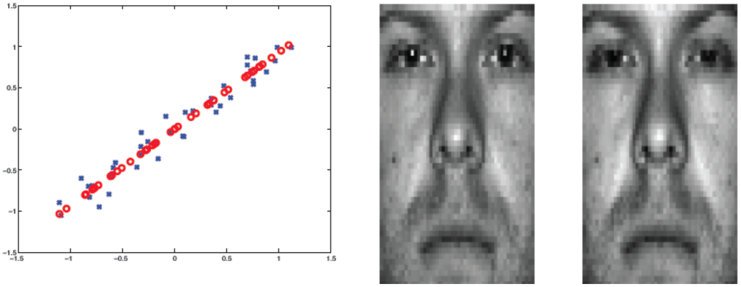
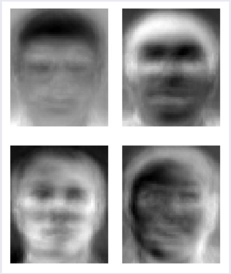


Figure: From SSBD: One dimensional reconstruction of 2D data on the left, and 10 dimensional reconstruction of face image on the right.

Eigenfaces [2]

- Eigenfaces was a face recognition method developed in the early 1990s.
  - Compute PCA and represent each image with its k largest principle components (unsupervised dimension reduction).
  - Can do face recognition using the nearest neighbour on the principle components (in the low dimensional subspace). Improved over using nearest neighbour with raw images.



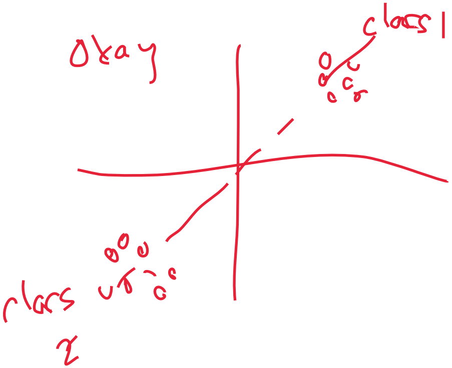
Exercise 7:

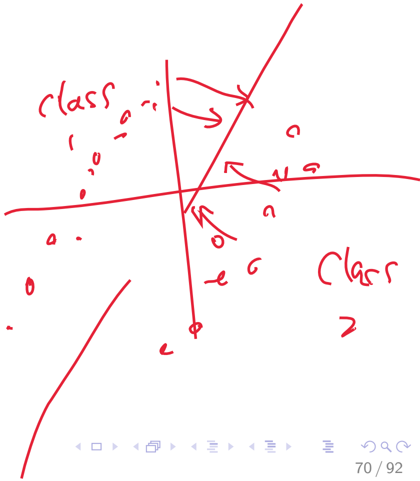
In this experiment, we implement Eigenfaces: do PCA then nearest neighbour classification.

Try different number of PCA components (20, 40 80) and whether whitening is done. What effects do they have on nearest neighbour accuracy? Why?

- Using PCA with 40 components helped a bit
- 20 components lost info
- With whitening, rescale each feature same variance
- good results with 20 components
  Poor with too many components
  amplifying noise

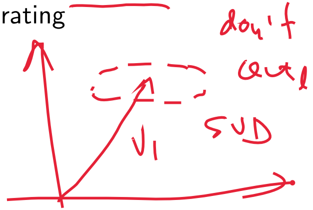
Exercise 8: One approach to feature selection is to do PCA followed by supervised learning, e.g. nearest neighbour. This method does not use the labels. What implicit assumptions are we making about the target function?

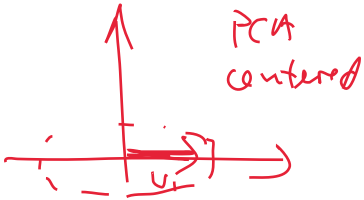




PCA Vs SVD

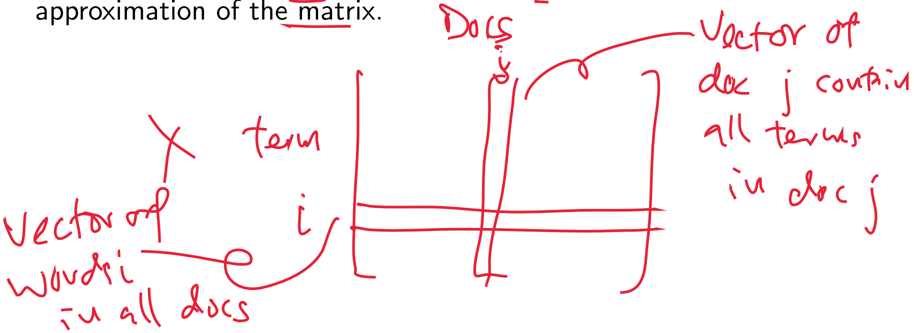
- In PCA, we perform SVD after removing the mean from each column of the matrix.
- Singular value decomposition (without the mean removed) is also used in various applications.
- The difference is whether we are interested in analysing the behaviour/statistics as we move away from the mean, or we are just interested in approximating the matrix
  - For example, in movie recommendation, we may be interested in modeling the change in rating as we move away from the average movie rating, rather than as we move away from zero rating





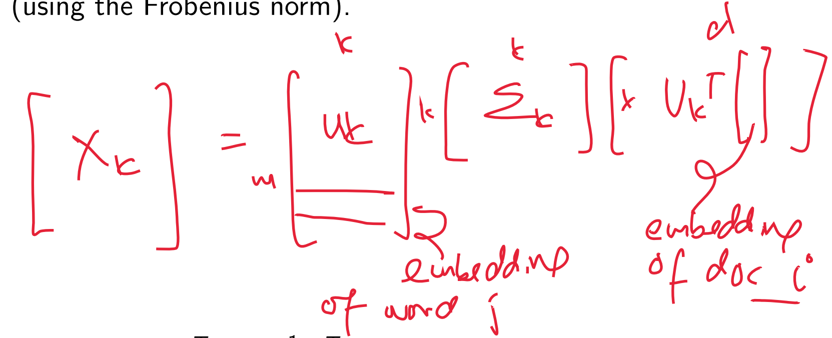
Latent Semantic Analysis

Given a term document matrix X where element $x_{i,j}$ describe the occurrence of term (word) i in document j, LSA finds a low-rank approximation of the matrix.


Docs
j
Vector of doc j contain all terms in doc j
X
term
i
Vector of word i in all docs

- Usually use singular value decomposition to get $X = U\Sigma V^T$ (without removing mean).

- Keep the k largest singular values (zero the rest) to get $X_k = U_k \Sigma_k V_k^T$. X_k is the best rank k approximation for X (using the Frobenius norm).



- Column i of $V_k^T = \Sigma_k^{-1} U_k^T X$ is embedding of document i.
- The word embedding is represented in $U_k = X V_k \Sigma_k^{-1}$, row j of U_k represents embedding of word j.

- Given a new document (or query) d_j compute low dimensional embedding of the document $\hat{d}_j = \Sigma_k^{-1} U^T d_j$.
  - Original document d_j is a vector whose length is the same as the vocabulary size, whereas $\hat{d}_j$ has length k.
  - Can do inner product with columns of V_k^T to find most similar document.
  - Can be helpful for matching documents that contains synonyms instead of matching words exactly.
- The j-th row of U_k, $\hat{t}_j$, is a low dimensional representation of word j.
  - The embedding $\hat{t}_j$ maps similar words to nearby location, e.g. "doctor" and "nurse" would be close by.
  - This is done unsupervised, and can be useful when the representation is used for supervised learning, e.g. for words unseen in the supervised training set.

Representation as Linear Combination of Functions

- We can write the $\underline{\text{factorization}}$ as a sum of rank 1 matrices $\mathbf{u}_i\mathbf{v}_i^T$.

$$\hat{X} = \underline{U}\underline{\Sigma_k}\underline{V}^T = \sum_{i=1}^{k} \underline{\sigma_i}\underline{\mathbf{u}_i\mathbf{v}_i^T}.$$

- For convenience, we absorb the diagonal matrix $\underline{\Sigma}$ into the other matrices, e.g. denoting $\underline{U\Sigma}$ as $\underline{U}$, giving $\hat{X} = \underline{UV}^T = \sum_{i=1}^{k} \underline{\mathbf{u}_i\mathbf{v}_i^T}$.
  - Illustration, with $\underline{k = 2}$ $\;\; n^2$

$$\begin{bmatrix} \hat{x}_{1,1} & \cdots & \hat{x}_{1,m} \\ \vdots & \ddots & \vdots \\ \hat{x}_{n,1} & \cdots & \hat{x}_{n,m} \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} \\ \vdots & \vdots \\ u_{n,1} & u_{n,2} \end{bmatrix} \begin{bmatrix} v_{1,1} & \cdots & v_{m,1} \\ v_{1,2} & \cdots & v_{m,2} \end{bmatrix} \Big\} 2$$

$$= \begin{bmatrix} u_{1,1} \\ \vdots \\ u_{n,1} \end{bmatrix} \underbrace{\begin{bmatrix} v_{1,1} & \cdots & v_{m,1} \end{bmatrix}}_{n \times m \text{ matrix}} + \begin{bmatrix} u_{1,2} \\ \vdots \\ u_{n,2} \end{bmatrix} \underbrace{\begin{bmatrix} v_{1,2} & \cdots & v_{m,2} \end{bmatrix}}_{n \times m}$$

- We can view the entries of $\hat{X}$ as a function of the rows and columns, and similarly the vector entries of $\mathbf{u}_i$, $\mathbf{v}_i$. Hence we can write
$$\hat{X}(a,b) = \sum_{i=1}^{k} \mathbf{u}_i(a)\mathbf{v}_i(b),$$
a linear combination of functions.
- We can learn the function $\hat{X}(a,b)$ from a training set just like any other learning problem: in this case we learn the parameters $\mathbf{u}_i(a)$ and $\mathbf{v}_i(b)$.
- A matrix is a two-dimensional table. The idea can be extended to q-dimensional tables called tensors as well:
$$\hat{X}(a_1,\ldots,a_q) = \sum_{i=1}^{k} \mathbf{u}_i^1(a_1)\mathbf{u}_i^2(a_2)\ldots\mathbf{u}_i^q(a_q).$$

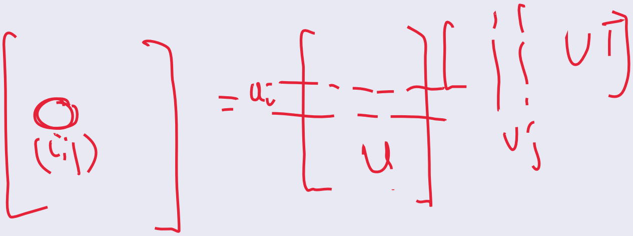
CP decomposition

Word2Vec [4]

- In Word2Vec, we represent the probability $p(w_j|w_i)$ of a word j appearing in a window ($\pm c$ words) around a target word i in text collections using

$$p(w_j|w_i) = \sigma(X(i,j)) = \sigma\left(\sum_{r=1}^{k} \mathbf{u}_r(i)\mathbf{v}_r(j)\right) = \sigma(\mathbf{u}_i^T\mathbf{v}_j),$$

where σ is the logistic function.



- For scalable training, [4] used a sampled approximation of a negative sampling loss

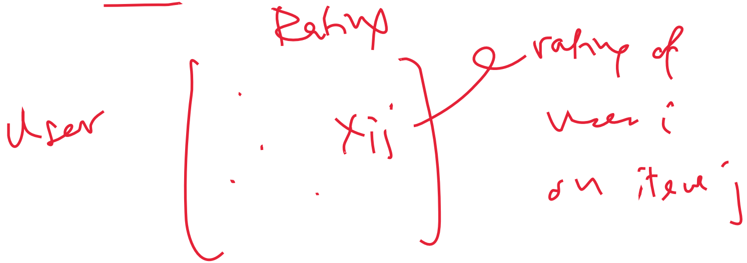
$$-\log \sigma(\mathbf{u}_i^T \mathbf{v}_j) - \sum_{j'=1}^{k} E_{\mathbf{w}_{j'} \sim P_n(w)}[\log \sigma(-\mathbf{u}_i^T \mathbf{v}_{j'})],$$

  where the k words are sampled from a noise distribution P_n.
- This is essentially logistic regression where observed pairs are positive samples and we generate k negative samples from each positive sample using a noise distribution.
- In Word2Vec, learned vector $\mathbf{u}_i$ is the embedding of word i.
- Highly scalable, e.g. has been trained for corpus of 100 billion words to construct 300 dimensional vectors for 3 million words and phrases (`https://code.google.com/p/word2vec`).

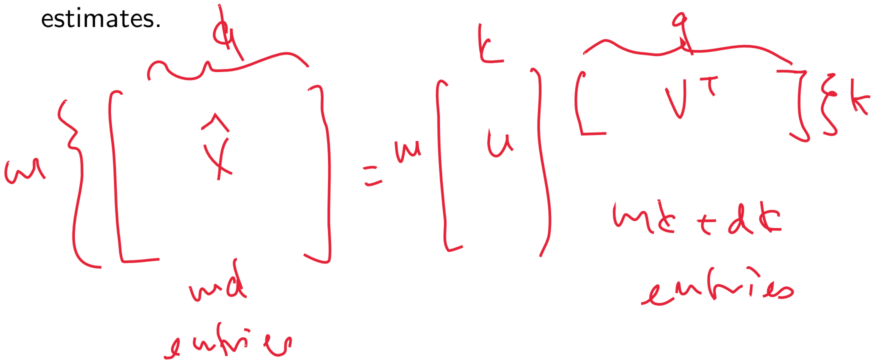
- The resulting embedded representation works well even with arithmetic operations, e.g. $\mathbf{u}_{\text{Paris}} - \mathbf{u}_{\text{France}} + \mathbf{u}_{\text{Italy}} \approx \mathbf{u}_{\text{Rome}}$. ~~Demo: `http://lucasestevam.com/w2vdemo/`~~.
- Adding Word2Vec as additional features for word sense disambiguation using SVM improved performance on three benchmark datasets (Semeval 2, Semeval 3 and Semeval 7) by 4.6%, 2.3%, and 1.5% [3].
  - Additional information from the unlabeled data helped supervised learning.
- Becoming a standard practice to use embedded representation trained from very large corpus: exploit unlabeled data to get a better representation of words, compared to one-hot encoding.

Collaborative Filtering

- By treating the matrix as a function $\hat{X}(a, b) = \sum_{i=1}^{k} \mathbf{u}_i(a)\mathbf{v}_i(b)$, we can use also matrix factorization for supervised learning.
- In a collaborative filtering task, we are given a partially filled $m \times d$ matrix X of ratings, where $x_{i,j}$ is the rating of user i on item (e.g. movie) j.

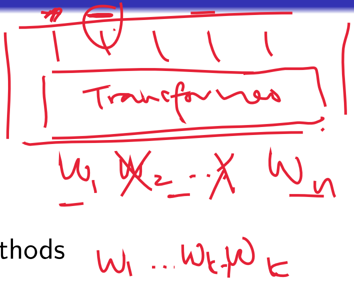


- To predict the unknown entries in X, we often try to learn a low rank matrix $\hat{X} = UV^T$ where U and V are $m \times k$ and $d \times k$ matrices. The number of parameters in U and V is $mk + dk$ and is much smaller than md, so we hope to be able to generalize and fill in the missing values in X with good estimates.



- Can treat as a regression problem and do supervised learning of observed ratings, e.g. approximate rating $r_{a,b}$ with $\hat{X}(a, b) = \sum_{i=1}^{k} \mathbf{u}_i(a)\mathbf{v}_i(b)$, trained by e.g. minimizing squared loss.
- Matrix factorization was one of the commonly used techniques in the Netflix $1M collaborative filtering competition (usually used with other techniques in an ensemble).

Reading



- SSBD Chapter 16 on Kernel Methods
- SSBD Chapter 23.1 on PCA

References I

[1] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. *The curse of dimensionality for local kernel machines*. Tech. rep. University of Montreal, 2005.

[2] *Eigenfaces*. [Online: https://en.wikipedia.org/wiki/Eigenface#/media/File:Eigenfaces.png, accessed July 2017].

[3] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. "Embeddings for Word Sense Disambiguation: An Evaluation Study." In: *ACL*. 2016.

[4] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

Outline

1 Linear Combination of Functions

2 Kernel Method

3 Neural Networks

4 Matrix/Tensor Factorization

5 Appendix

Kernel Methods Proofs

Theorem (Representer Theorem): Let ψ be a mapping from $\mathcal{X}$ to a Hilbert space. Then there exists a vector $\alpha \in \mathbb{R}^m$ such that $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i)$ is an optimal solution to

$$\min_{\mathbf{w}} (f(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{w}, \psi(\mathbf{x}_m)\rangle) + R(\|w\|),$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically nondecreasing function.

Proof:

- Let $\mathbf{w}^*$ be an optimal solution to the problem.
- As $\mathbf{w}^*$ is an element in the Hilbert space, it can be rewritten as

$$\mathbf{w}^* = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i) + \mathbf{u},$$

  where $\mathbf{u}$ is orthogonal to the subspace spanned by $\{\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)\}$, i.e. $\langle \mathbf{u}, \psi(\mathbf{x}_i)\rangle = 0$ for all i.
- Let $\mathbf{w} = \mathbf{w}^* - \mathbf{u}$, where $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i)$ is in the span of $\{\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)\}$.
- Then $\|w^*\|^2 = \|w\|^2 + \|u\|^2$ giving $\|w\| \leq \|w^*\|$.
- As R is nondecreasing, we have $R(\|w\|) \leq R(\|w^*\|)$.

- We also have

$$\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^* - \mathbf{u}, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^*, \psi(\mathbf{x}_i) \rangle,$$

as $\langle \mathbf{u}, \psi(\mathbf{x}_i) \rangle = 0$ for all i.

- So

$$f(\langle \mathbf{w}, \psi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}, \psi(\mathbf{x}_m) \rangle) = f(\langle \mathbf{w}^*, \psi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}^*, \psi(\mathbf{x}_m) \rangle).$$

- Together with $R(\|w\|) \leq R(\|w^*\|)$, we see that $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ is as good a solution as $\mathbf{w}^*$. □

Lemma: A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implements an inner product in some Hilbert space if and only if it is positive semidefinite; i.e. for all $\mathbf{x}_1, \ldots, \mathbf{x}_m$, the Gram matrix $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ is a positive semidefinite matrix.
Proof:

- If K implements some inner product then

$$\begin{aligned}\boldsymbol{\alpha}^T G \boldsymbol{\alpha} &= \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \left\langle \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \psi(\mathbf{x}_j) \right\rangle = \langle \mathbf{w}, \mathbf{w} \rangle = \|\mathbf{w}\|^2 \geq 0.\end{aligned}$$

- We now show: if the kernel function is positive semidefinite, it implements an inner product in some Hilbert space.
- We do that by constructing an inner product using the kernel function.
- For each element $\mathbf{x} \in \mathcal{X}$, we define a feature function (instead of a feature vector) $\psi(\mathbf{x}) = K(\cdot, \mathbf{x})$. So

$$\langle K(\cdot, \mathbf{x}), K(\cdot, \mathbf{x}') \rangle = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = K(\mathbf{x}, \mathbf{x}').$$

- Define a vector space by taking all linear combinations of functions of the form $K(\cdot, \mathbf{x})$.
- On this vector space, we define an inner product

$$\left\langle \sum_i \alpha_i K(\cdot, \mathbf{x}_i), \sum_j \beta_j K(\cdot, \mathbf{x}'_j) \right\rangle = \sum_{i,j} \alpha_i \beta_j K(\mathbf{x}_i, \mathbf{x}'_j).$$

- This is a valid inner product: it is symmetric, linear in the first argument, and positive definite.
- To see that it is positive definite, note that the Gram matrix is assumed to be positive semidefinite (almost there).
- Note the following an interesting property: evaluation of a function at $\mathbf{x}$ can be done by taking an inner product with a kernel function parameterized by $\mathbf{x}$. Let $f(\mathbf{x}) = \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i)$. Then

$$\langle K(\cdot, \mathbf{x}), f \rangle = \langle K(\cdot, \mathbf{x}), \sum_i \alpha_i K(\cdot, \mathbf{x}_i) \rangle = \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) = f(\mathbf{x}).$$

- This property is called the reproducing kernel property and the corresponding Hilbert space is called a reproducing kernel Hilbert space.

- The reproducing kernel property allows us to show that the inner product is positive definite, completing the proof:

  $$|f(x)|^2 = |\langle K(\cdot, \mathbf{x}), f\rangle|^2 \leq \langle K(\cdot, \mathbf{x}), K(\cdot, \mathbf{x})\rangle\langle f, f\rangle = K(\mathbf{x}, \mathbf{x})\langle f, f\rangle.$$

  So $\langle f, f\rangle = 0$ means that $f(x) = 0$ for all x, and the inner product is positive definite. $\square$