CS-UY 4563: Lecture 13
Kernel Methods

NYU Tandon School of Engineering, Prof. Christopher Musco

My new office: `https://nyu.zoom.us/my/cmusco`

- Visit this URL during my office hours or any individual meetings.
- You can also "drop in" even if I'm not in the chat. I will receive an email notification (might take me a few minutes to notice) and can then let you in if I'm free.
- For lectures still use the links on NYU Classes (which allows for automatic recording, transcription, etc.)

By 4/1 (next Wednesday) choose a partner and topic.

- Email me team members, project topic, and a sentence or two about your idea. If you have any data sets in mind, let me know that as well.
- Then set up a meeting at: `https://docs.google.com/spreadsheets/d/1DsR7ia4VfYb5joIavsG8_T1JBgAufkVbdT7bLfPGGQ0/edit?usp=sharing`.

k-NN algorithm: a simple but powerful baseline for classification.

Training data: $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$ where $y_1, \ldots, y_n \in \{1, \ldots, q\}$.

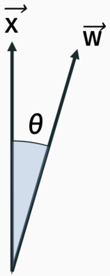
Classification algorithm:

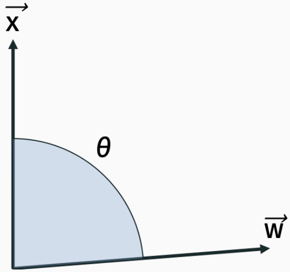
Given new input $\vec{x}_{new}$,

- Compute $sim(\vec{x}_{new}, \vec{x}_1), \ldots, sim(\vec{x}_{new}, \vec{x}_n)$.
- Let $\vec{x}_{j_1}, \ldots, \vec{x}_{j_k}$ be the k training data vectors with highest similarity to $\vec{x}_{new}$.
- Predict y_{new} as $majority(y_{j_1}, \ldots, y_{j_k})$.

Given data vectors $\vec{x}, \vec{w} \in \mathbb{R}^d$, the inner product $\langle \vec{x}, \vec{w} \rangle$ is a natural similarity measure.

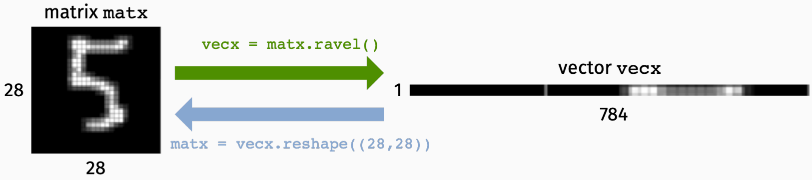
$$\langle \vec{x}, \vec{w} \rangle = \sum_{i=1}^{d} \vec{x}[i]\vec{w}[i] = \cos(\theta)\|\vec{x}\|_2\|\vec{w}\|_2.$$





Each pixel is number from $[0, 1]$. 0 is black, 1 is white.

Represent 28×28 matrix of pixel values as a flattened vector.




```
xmat = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

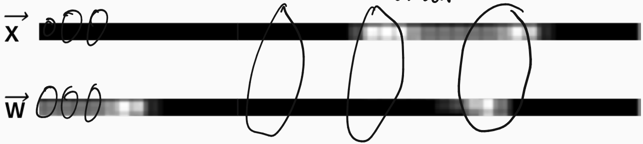
```
xvec = xmat.ravel()
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

INNER PRODUCT FOR MNIST

Inner product between MNIST digits: [0, 1]

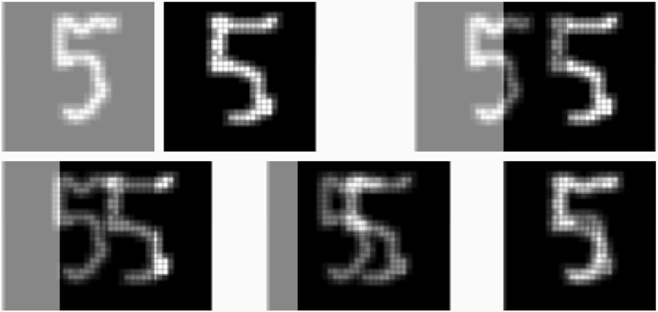
black white



$$\langle \vec{x}, \vec{w} \rangle = \sum_{i=1}^{28} \sum_{j=1}^{28} \underline{\texttt{matx}[i,j]\texttt{matw}[i,j]}.$$

Inner product similarity is higher when the images have large pixel values (close to 1) in the same locations. I.e. when they have a lot of overlapping white/light gray pixels.

Visualizing the inner product between two images:



Images with high inner product have a lot of overlap.

$q = 10$

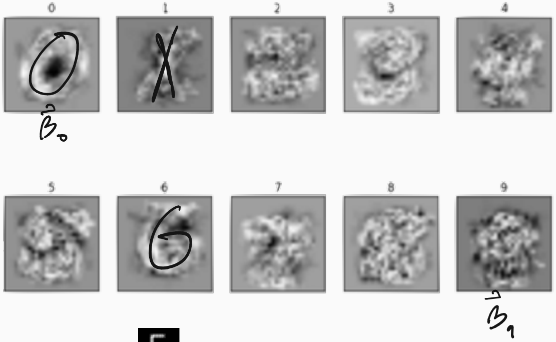
One-vs.-all Classification with Logistic Regression:

- Learn q classifiers with parameters $\vec{\beta}_0, \vec{\beta}_1, \ldots, \vec{\beta}_{q-1}$.
- Given $\vec{x}_{new}$ compute $\langle \vec{x}_{new}, \vec{\beta}_0 \rangle, \ldots, \langle \vec{x}_{new}, \vec{\beta}_{q-1} \rangle$
- Predict class $y_{new} = \arg\max_i \langle \vec{x}_{new}, \vec{\beta}_i \rangle$.

If each $\vec{x}$ is a vector with $28 \times 28 = 784$ entries than each $\vec{\beta}_i$ also has 784 entries. Each parameter vector can be viewed as a 28×28 image.

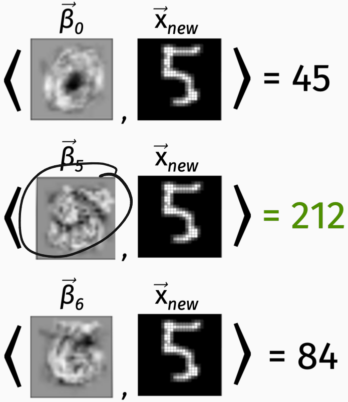
MATCHED FILTER

Visualizing $\vec{\beta}_0, \ldots, \vec{\beta}_9$:



For an input image  , compute inner product similarity with all weight matrices and choose most similar one.

In contrast to k-NN, only need to compute similarity with q items instead of n.



Select class for $\vec{x}_{new}$ which achieves highest "score", as measured by the inner product similarity.

ALTERNATIVE VIEW

Logistic Regression Model:

Given data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ (here $d = 784$) and binary label vector $\vec{y} \in \{0, 1\}^n$ for class i (1 if in class i, 0 if not), find $\vec{\beta}_i \in \mathbb{R}^d$ to minimize the log loss between:

data matrix

$$\vec{y} \qquad \text{and} \qquad h(\mathbf{X}\vec{\beta})$$

where $h(\mathbf{X}\vec{\beta}_i) = \frac{1}{1+e^{-\mathbf{X}\vec{\beta}_i}}$ applies the logistic function entrywise to the vector $\mathbf{X}\vec{\beta}$.

Loss = $-\sum_{j=1}^{n} y_j \log(h(\mathbf{X}\vec{\beta}_i)[j]) + (1 - y_j) \log(1 - h(\mathbf{X}\vec{\beta}_i)[j])$

Logistic Regression Model:

Given data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ (here $d = 784$) and binary label vector $\vec{y} \in \{0, 1\}^n$ for class i (1 if in class i, 0 if not), find $\vec{\beta} \in \mathbb{R}^d$ to minimize the log loss between:

$$\underline{\underline{\vec{y}}} \quad \text{and} \quad h(\mathbf{X}\vec{\beta})$$

logistic function

Reminder from linear algebra: Without loss of generality, can assume that $\vec{\beta}$ lies in the row span of $\mathbf{X}$.

So for any $\vec{\beta} \in \mathbb{R}^d$, there exists a vector $\vec{\alpha} \in \mathbb{R}^n$ such that:

$$\vec{\beta} = \mathbf{X}^T\vec{\alpha}.$$

$$X\beta = XX^T\alpha$$

Logistic Regression Equivalent Formulation:

Given data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ (here $d = 784$) and binary label vector $\vec{y} \in \{0, 1\}^n$ for class i (1 if in class i, 0 if not), find $\vec{\alpha} \in \mathbb{R}^n$ to minimize the log loss between:

$$\vec{y} \qquad \text{and} \qquad h(\mathbf{X}\mathbf{X}^T\vec{\alpha}).$$

[0, 0, 1, 1, 0, 1 ...]

Can still be minimized via gradient descent:

$$\nabla L(\vec{\alpha}) = \mathbf{X}\mathbf{X}^T(h(\mathbf{X}\mathbf{X}^T\vec{\alpha}) - \vec{y}).$$

REFORMULATED VIEW

What does classification for a new point $\vec{x}_{new}$ look like? Recall that for a given one-vs-all classification i, the original parameter vector $\vec{\beta}_i = \mathsf{X}^T\vec{\alpha}_i$. $\vec{\alpha}_0, \ldots, \vec{\alpha}_q$

- Learn q classifiers with parameters $\vec{\alpha}_1, \vec{\alpha}_2, \ldots, \vec{\alpha}_q$.
- Given $\vec{x}_{new}$ compute $\langle \vec{x}_{new}, \mathsf{X}^T\vec{\alpha}_1\rangle, \ldots, \langle \vec{x}_{new}, \mathsf{X}^T\vec{\alpha}_q\rangle$,
- Predict class $y_{new} = \arg\max_i \langle \vec{x}_{new}, \mathsf{X}^T\vec{\alpha}_i\rangle$.

$\langle \vec{x}_{new}, \vec{B}_i \rangle$

$(724 \times 1) \quad (724, 1)$

$X = n \begin{bmatrix} & \end{bmatrix}$, 724

Score for class i:

$$\langle \vec{x}_{new}, \mathsf{X}^T\vec{\alpha}_i \rangle = (\vec{x}_{new}^T \mathsf{X}^T)\vec{\alpha}_i$$
$$= \langle \mathsf{X}\vec{x}_{new}, \vec{\alpha}_i \rangle$$
$$= \sum_{j=1}^{n} \vec{\alpha}_i[j] \langle \vec{x}_{new}, \vec{x}_j \rangle.$$

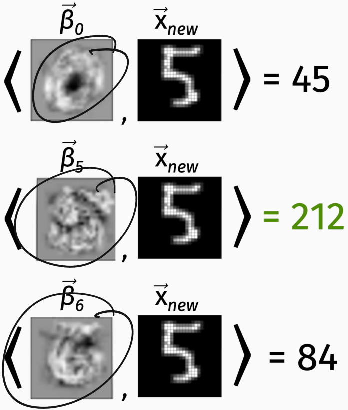
724 x 1

$(724 \times n) \cdot (n \times 1)$

$= (724 \times 1)$

weight for every

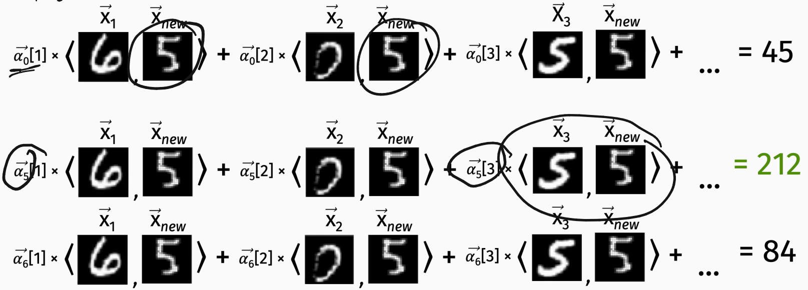
$\vec{x}_j$ in training data


$\vec{\beta}_0$
$\vec{x}_{new}$
= 45
$\vec{\beta}_5$
$\vec{x}_{new}$
= 212
$\vec{\beta}_6$
$\vec{x}_{new}$
= 84

	$\vec{x}_1$	$\vec{x}_2$	$\vec{x}_3$	$\vec{x}_4$	$\vec{x}_5$	$\vec{x}_6$	...
$\vec{\alpha}_0$	$\vec{\alpha}_0[1] = .4$	$\vec{\alpha}_0[2] = .9$	$\vec{\alpha}_0[3] = .2$	$\vec{\alpha}_0[4] = .1$	$\vec{\alpha}_0[5] = .8$	$\vec{\alpha}_0[6] = .05$	...
$\vec{\alpha}_1$	$\vec{\alpha}_1[1] = .05$	$\vec{\alpha}_1[2] = .1$	$\vec{\alpha}_1[3] = .2$	$\vec{\alpha}_1[4] = .02$	$\vec{\alpha}_1[5] = .1$	$\vec{\alpha}_1[6] = .95$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\vec{\alpha}_5$	$\vec{\alpha}_5[1] = .1$	$\vec{\alpha}_5[2] = .2$	$\vec{\alpha}_5[3] = .85$	$\vec{\alpha}_5[4] = .75$	$\vec{\alpha}_5[5] = .1$	$\vec{\alpha}_5[6] = .05$	...
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Learn n length parameter vectors $\vec{\alpha}_0, \ldots, \vec{\alpha}_9$, one for each class.

$\sum_{j=1}^{n} \vec{\alpha}_i[j] \langle \vec{x}_j, \vec{x}_{new} \rangle$ for all $i = 0, \dots, 9$



Classification looks similar to k-NN: we compute the similarity between $\vec{x}_{new}$ and every other vector in our training data set. A weighted sum of the similarities leads to scores for each class.

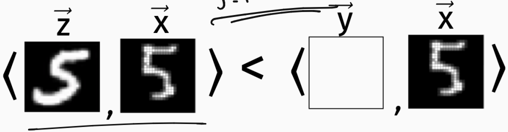
Assign $\vec{x}_{new}$ to the class with highest score.

Often the inner product does not make sense as a similarity measure between data vectors. Here's an example (recall that smaller inner product means less similar):

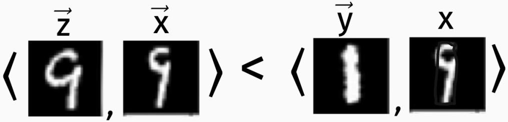
$\sum_{j=1}^{724} z_j x_j$

$z_j \le 1$

$\sum_{j=1}^{724} y_j x_j$



But clearly the first image is more similar.



Here's a more realistic scenario.

A kernel function $k(\vec{x}, \vec{y})$ is simply a similarity measure between data points.

$$k(\vec{x}, \vec{y}) = \begin{cases} \text{large if } \vec{x} \text{ and } \vec{y} \text{ are similar.} \\ \text{close to 0 if } \vec{x} \text{ and } \vec{y} \text{ are different.} \end{cases}$$

(or negative)

Example: The Radial Basis Function (RBF) kernel, aka the Gaussian kernel:

some scaling factor

$e^{-\|x-y\|_2^2} < e^{-\|x-z\|_2^2}$

closer to 0 — closer to 1

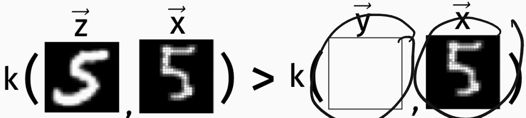
$$k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2^2/\sigma^2}$$

$\to$ 0 when $\|x-y\|_2^2$ is large

for some scaling factor σ.

$\|\vec{x}-\vec{y}\|_2^2 > \|\vec{x}-\vec{z}\|_2^2$ is large

$\to$ 1 when $\|x-y\|_2^2$ is small.



$\|y-x\|_2^2 = \|x-y\|_2^2$

$\boxed{5}$ $\vec{y}-\vec{x}$

Lots of kernel functions functions involve transformations of $\langle\vec{x}, \vec{y}\rangle$ or $\|\vec{x} - \vec{y}\|_2$:

- Gaussian RBF Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2^2/\sigma^2}$
- Laplace Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2/\sigma}$
- Polynomial Kernel: $k(\vec{x}, \vec{y}) = (\langle\vec{x}, \vec{y}\rangle + 1)^q$. a positive integer

But you can imagine much more complex similarity metrics.

For a simple algorithm like k-NN you can swap our the inner product similarity with any similarity function you could possibly imagine.

For a methods like logistic regression, this is not the case...

Recall: We learned a parameter vector $\vec{\alpha}$ to minimize $LL(\vec{y}, \mathbf{X}^T\vec{\alpha})$ where $LL()$ denotes the logistic loss. Then we classified via:

$X\vec{\beta} = X(X^T\alpha)$

$$\langle \vec{x}_{new}, \mathbf{X}^T\vec{\alpha}\rangle = \vec{x}_{new}^T\mathbf{X}^T\vec{\alpha} = \sum_{j=1}^{n} \vec{\alpha}[j]\langle \vec{x}_{new}, \vec{x}_j\rangle.$$

The inner product similarity came from the fact that our predictions were based on the linear function $\mathbf{X}^T\alpha$.

KERNEL FUNCTIONS AS FEATURE TRANSFORMATION

A positive semidefinite (PSD) kernel is any similarity function with the following form:

$$k(\vec{x}, \vec{w}) = \phi(\vec{x})^T \phi(\vec{w})$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^m$ is a some feature transformation function.

Example: Degree 2 polynomial kernel, $k(\vec{x}, \vec{w}) = (\vec{x}^T \vec{w} + 1)^2$.

$d = 3$

$m = 10$

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \phi(\vec{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_3 \\ x_1^2 \\ x_2^2 \\ x_3^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \sqrt{2}x_2x_3 \end{bmatrix}$$

$$\begin{aligned}
(\vec{x}^T \vec{w} + 1)^2 &= (x_1 w_1 + x_2 w_2 + x_3 w_3 + 1)^2 \\
&= 1 + 2x_1w_1 + 2x_2w_2 + 2x_3w_3 + x_1^2w_1^2 + x_2^2w_2^2 + x_3^2w_3^2 \\
&\quad + 2x_1w_1x_2w_2 + 2x_1w_1x_3w_3 + 2x_2w_2x_3w_3 \\
&= \phi(\vec{x})^T \phi(\vec{w}).
\end{aligned}$$

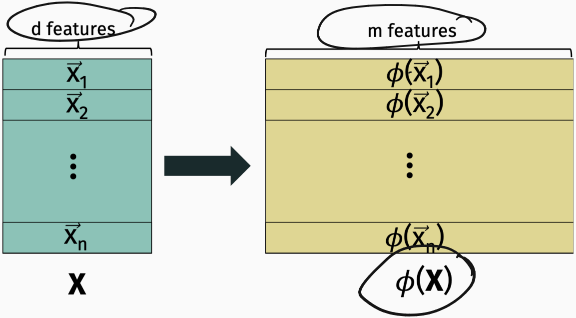
Not all similarity metrics you are positive semidefinite (PSD), but all of the ones we saw earlier are:

- Gaussian RBF Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2^2/\sigma^2}$
- Laplace Kernel: $k(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|_2/\sigma}$
- Polynomial Kernel: $k(\vec{x}, \vec{y}) = (\langle\vec{x}, \vec{y}\rangle + 1)^q$.

And there are many more...

Feature transformations $\iff$ new similarity metrics.

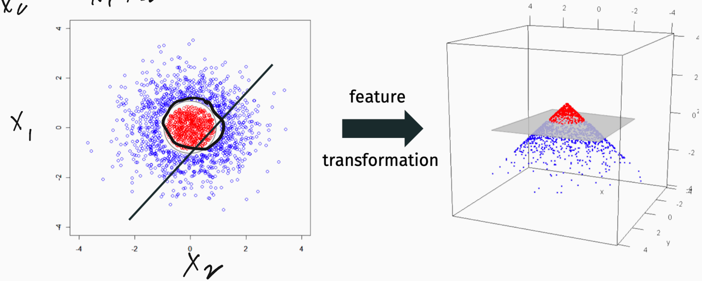
Using $k(\cdot,\cdot)$ in place of the inner product $\langle\cdot,\cdot\rangle$ is **equivalent** to replacing every data point $\vec{x}_1,\ldots,\vec{x}_n$ in our data set with $\phi(\vec{x}_1),\ldots,\phi(\vec{x}_n)$.



TAKEAWAY ONE

We can improve performance by replacing the inner product with another kernel $k(\cdot, \cdot)$ for the same reason that feature transformations improved performance.

$x_1^2 \quad x_2^2 \quad x_1 x_2$



When you add features, it becomes possible to learn more complex decision boundaries (in this case a circle) with a linear classifier.

PSD kernel functions give a principled way of “swapping out” the inner product with a new similarity metric for linear algorithms like multiple linear regression or logistic regression.

For non-PSD kernels it is not clear how to do this.

Standard logisitic regression

XX^T

Loss function:

Los Loss

$$L(\vec{\alpha}) = LL(\vec{y}, XX^T\vec{\alpha}).$$

Gradient:

$$\nabla L(\vec{\alpha}) = XX^T(h(XX^T\vec{\alpha}) - \vec{y}).$$

Prediction:

$\langle \vec{\beta}, x_{new} \rangle$

$$z = \sum_{j=1}^{n} \vec{\alpha}[j] \langle \vec{x}_{new}, \vec{x}_j \rangle.$$

$$y_{new} = \mathbb{1}[z > 0]$$

Kernel logisitic regression

$\phi(X)\phi(X)^T$

Loss function:

$\phi(X)$

$$L(\vec{\alpha}) = LL(\vec{y}, \phi(X)^T\vec{\alpha}).$$

Gradient:

$$\nabla L(\vec{\alpha}) = \phi(X)\phi(X)^T(h(\phi(X)\phi(X)^T\vec{\alpha}) - \vec{y}).$$

Prediction:

$= K(\vec{x}_{new}, \vec{x}_j)$

$$z = \sum_{j=1}^{n} \vec{\alpha}[j] \langle \phi(\vec{x}_{new}), \phi(\vec{x}_j) \rangle$$

$e^{-\|x_{new} - x_j\|_2^2}$

$$y_{new} = \mathbb{1}[z > 0]$$

Standard linear regression

Loss function: $\|y - X\vec{\beta}\|_2$

$$L(\vec{\alpha}) = \|\vec{y} - \mathbf{X}\mathbf{X}^T\vec{\alpha}\|_2$$

Gradient: $\vec{x}^T\alpha = \vec{\beta}$

$$\nabla L(\vec{\alpha}) = \underline{2\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T\vec{\alpha} - \vec{y})}.$$

Prediction:

$$y_{new} = \sum_{j=1}^{n} \vec{\alpha}[j]\langle \vec{x}_{new}, \vec{x}_j \rangle.$$

$[\;] = XX^T\vec{\alpha}$

Kernel linear regression

Loss function:

$$L(\vec{\alpha}) = \|\vec{y} - \phi(\mathbf{X})\phi(\mathbf{X})^T\vec{\alpha}\|_2$$

Gradient:

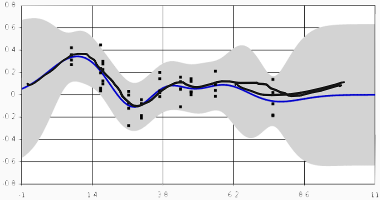
$$\nabla L(\vec{\alpha}) = 2\underline{\phi(\mathbf{X})\phi(\mathbf{X})^T}(\phi(\mathbf{X})\phi(\mathbf{X})^T\alpha - \vec{y}).$$

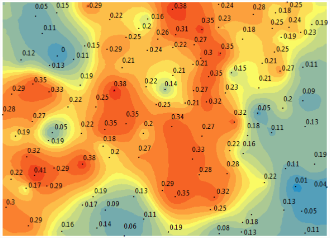
Prediction: $\underline{k(\vec{x}_{new}, \vec{x}_j)}$

$$y_{new} = \sum_{j=1}^{n} \vec{\alpha}[j]\langle \underline{\phi(\vec{x}_{new}), \phi(\vec{x}_j)} \rangle.$$

KERNEL REGRESSION

We won't study kernel regression in detail, but it's a very important statistical tool, especially when dealing with spatial or temporal data.



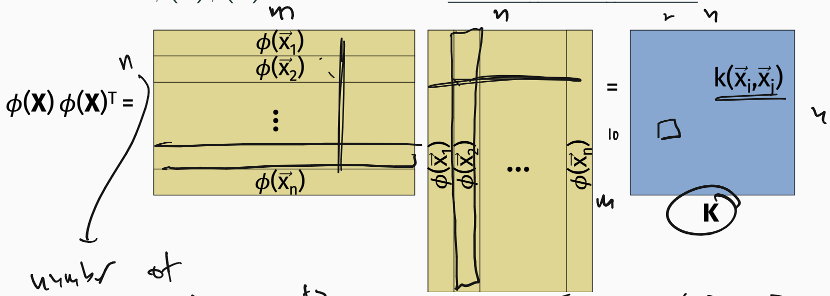


Also known as Gaussian Process (GP) Regression or Kriging.

XX^T "Gram matrix"

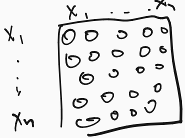
or "Gramian matrix"

$K = \phi(X)\phi(X)^T$ is called the kernel Gram matrix.



number of training data points

$\Phi(X)^T$

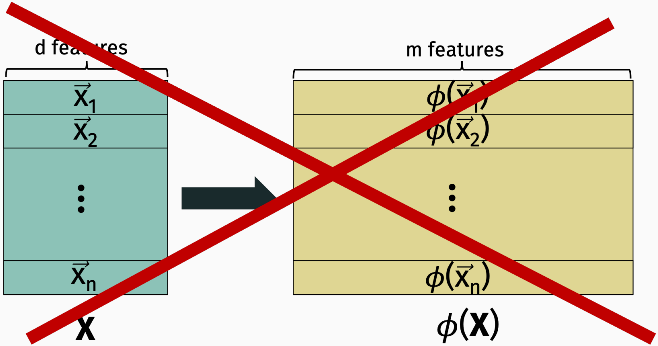


$K(\vec{x}_2, \vec{x}_{10})$

$= \langle \Phi(\vec{x}_2), \Phi(\vec{x}_{10}) \rangle$

We never need to actually compute $\phi(\vec{x}_1), \ldots, \phi(\vec{x}_n)$ explicitly:

- For training we just need the kernel matrix $\mathbf{K}$, which requires computing $k(\vec{x}_i, \vec{x}_j)$ for all i, j.
- For testing we just need to compute $k(\vec{x}_{new}, \vec{x}_i)$ for all i.



This can lead to significant computational savings!

- Transform dimension m is often very large: e.g. $m = O(d^q)$ for a degree q polynomial kernel.
- For many kernels (e.g. the Gaussian kernel) m is actually *infinite*.

The kernel matrix $\mathbf{K}$ is still $n \times n$ though which is huge when the number of data examples n is large. Has made the kernel trick less appealing in some modern ML applications.

Many algorithmic advances in recent years partially address this computational challenge (random Fourier features methods, Nystrom methods, etc.)