CS-GY 6923: Lecture 8
k-Nearest Neighbors, Kernel Methods

NYU Tandon School of Engineering, Prof. Christopher Musco

NON-LINEAR METHODS

- Previous methods studied (regression, logistic regression) are considered linear methods. They make predictions based on $\langle \mathbf{x}, \boldsymbol{\beta} \rangle$ – i.e. based on weighted sums of features.
- In the next part of the course we move on to non-linear methods. Specifically, **kernel methods** and **neural networks**.
- Both are very closely related to feature transformations!

WARM UP: k-NEAREST NEIGHBOR METHOD

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

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

Classification algorithm:

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

- Compute $sim(\mathbf{x}_{new}, \mathbf{x}_1), \ldots, sim(\mathbf{x}_{new}, \mathbf{x}_n)$.[1]
- Let $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_k}$ be the training data vectors with highest similarity to $\mathbf{x}_{new}$.
- Predict y_{new} as $majority(y_{j_1}, \ldots, y_{j_k})$.

[1] $sim(\mathbf{x}_{new}, \mathbf{x}_i)$ is any chosen similarity function, like $1 - \|\mathbf{x}_{new} - \mathbf{x}_i\|_2$.

k-NEAREST NEIGHBOR METHOD

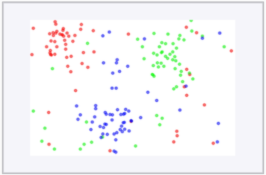


Fig. 1. The dataset.

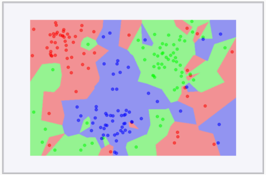


Fig. 2. The 1NN classification map.

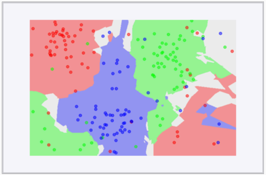


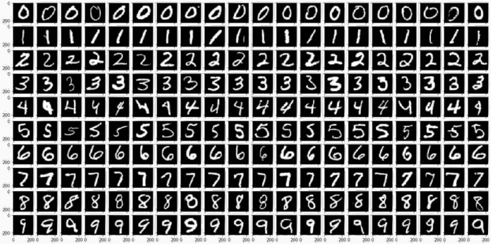
Fig. 3. The 5NN classification map.

- Smaller k, more complex classification function.
- Larger k, more robust to noisy labels.

Works remarkably well for many datasets.

MNIST IMAGE DATA

Especially good for large datasets with lots of repetition. Works well on MNIST for example:



≈ 95% **Accuracy out-of-the-box.**[2]

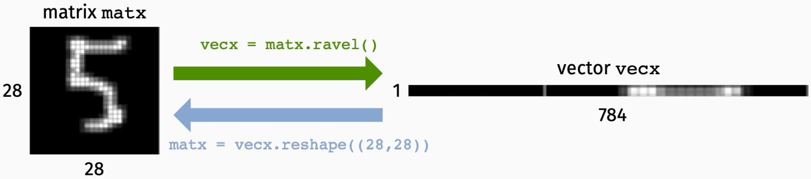
Let's look into this example a bit more...

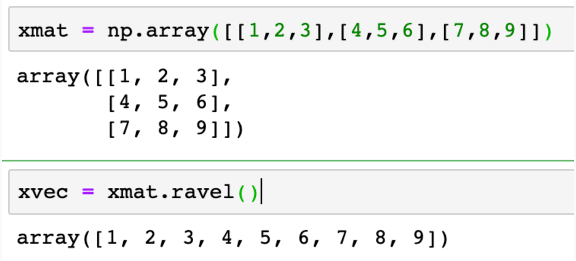
[2] Can be improved to 99.5% with a fancy similarity function!

MNIST IMAGE DATA

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]])
```

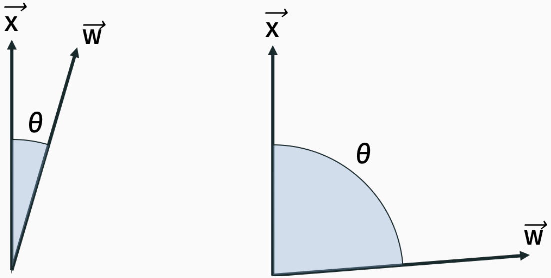
```
xvec = xmat.ravel()
```

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

INNER PRODUCT SIMILARITY

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

$$\langle \mathbf{x}, \mathbf{w} \rangle = \sum_{i=1}^{d} x_i w_i = \cos(\theta) \|\mathbf{x}\|_2 \|\mathbf{w}\|_2.$$



Also called “cosine similarity”.

INNER PRODUCT SIMILARITY

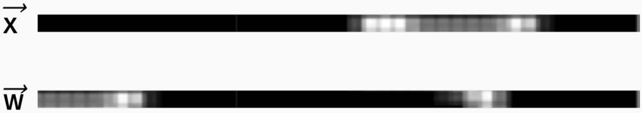
Connection to Euclidean (ℓ_2) Distance:

$$\|\mathbf{x} - \mathbf{w}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{w}\|_2^2 - 2\langle \mathbf{x}, \mathbf{w} \rangle$$

If all data vectors has the same norm, the pair of vectors with largest inner product is the pair with smallest Euclidean distance.

INNER PRODUCT FOR MNIST

Inner product between MNIST digits:

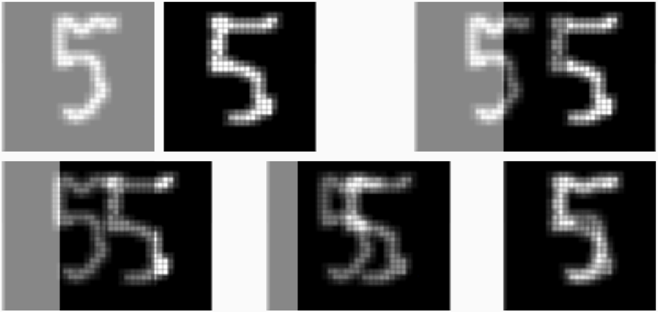


$$\langle \mathtt{x}, \mathtt{w} \rangle = \sum_{i=1}^{28} \sum_{j=1}^{28} \mathtt{matx}[i,j] \cdot \mathtt{matw}[i,j].$$

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.

INNER PRODUCT FOR MNIST

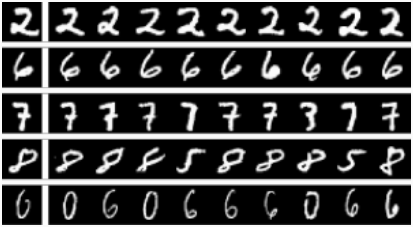
Visualizing the inner product between two images:



Images with high inner product have a lot of overlap.

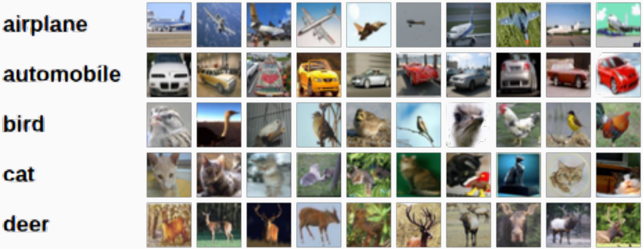
K-NN ALGORITHM ON MNIST

Most similar images during k-nn search, $k = 9$:



K-NN FOR OTHER IMAGES

Does not work as well for less standardized classes of images:



CIFAR 10 Images

Even after scaling to have same size, converting to separate RGB channels, etc. something as simple as k-nn won't work.

ANOTHER VIEW ON LOGISTIC REGRESSION

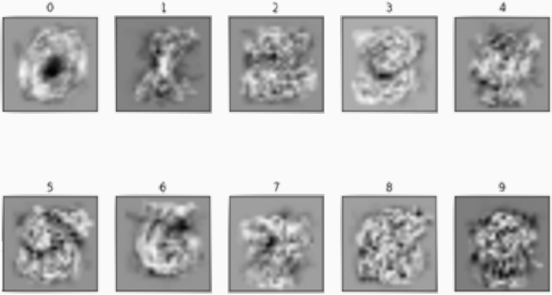
One-vs.-all or Multiclass Cross-entropy Classification with Logistic Regression:

- Learn q classifiers with parameters $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \ldots, \boldsymbol{\beta}^{(q)}$.
- Given $\mathbf{x}_{new}$ compute $\langle \mathbf{x}_{new}, \boldsymbol{\beta}^{(1)} \rangle, \ldots, \langle \mathbf{x}_{new}, \boldsymbol{\beta}^{(q)} \rangle$
- Predict class $y_{new} = \arg\max_i \langle \mathbf{x}_{new}, \boldsymbol{\beta}^{(i)} \rangle$.

If each $\mathbf{x}$ is a vector with $28 \times 28 = 784$ entries than each $\boldsymbol{\beta}^{(i)}$ also has 784 entries. Each parameter vector can be viewed as a 28×28 image.

MATCHED FILTER

Visualizing $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_q$:



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.

ALTERNATIVE VIEW

Logistic Regression Model:

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

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

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

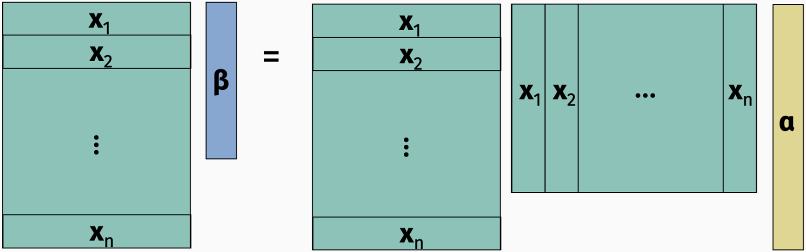
Loss = $-\sum_{j=1}^{n} y_j \log(h(\mathbf{X}\boldsymbol{\beta})_j) + (1 - y_j) \log(1 - h(\mathbf{X}\boldsymbol{\beta})_j)$

ALTERNATIVE VIEW

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

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

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



ALTERNATIVE VIEW

Logistic Regression Equivalent Formulation:

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

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

Can still be minimized via gradient descent:

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

REFORMULATED VIEW

What does classification for a new point x_{new} look like? Recall that for a given one-vs-all classification fro class i, the original parameter vector $\boldsymbol{\beta}_i = \mathrm{X}^T\boldsymbol{\alpha}_i$.

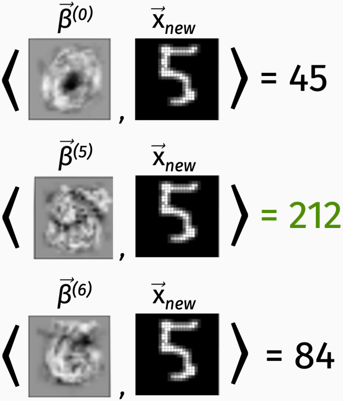
- Learn q classifiers with parameters $\boldsymbol{\alpha}^{(1)}, \boldsymbol{\alpha}^{(2)}, \ldots, \boldsymbol{\alpha}^{(q)}$.
- Given x_{new} compute $\langle \mathrm{x}_{new}, \mathrm{X}^T\boldsymbol{\alpha}^{(1)}\rangle, \ldots, \langle \mathrm{x}_{new}, \mathrm{X}^T\boldsymbol{\alpha}^{(q)}\rangle$
- Predict class $y_{new} = \arg\max_i \langle \mathrm{x}_{new}, \mathrm{X}^T\boldsymbol{\alpha}^{(i)}\rangle$.

REFORMULATED VIEW

Score for class *i*:

$$\begin{aligned}\langle \mathsf{x}_{new}, \mathsf{X}^T \boldsymbol{\alpha}_i \rangle &= \mathsf{x}_{new}^T \mathsf{X}^T \boldsymbol{\alpha}^{(i)} \\ &= \langle \mathsf{X}\mathsf{x}_{new}, \boldsymbol{\alpha}^{(i)} \rangle \\ &= \sum_{j=1}^{n} \alpha_j^{(i)} \langle \mathsf{x}_{new}, \mathsf{x}_j \rangle.\end{aligned}$$

ORIGINAL VIEW OF LOGISTIC REGRESSION



$$\langle \vec{\beta}^{(0)}, \vec{x}_{new} \rangle = 45$$

$$\langle \vec{\beta}^{(5)}, \vec{x}_{new} \rangle = 212$$

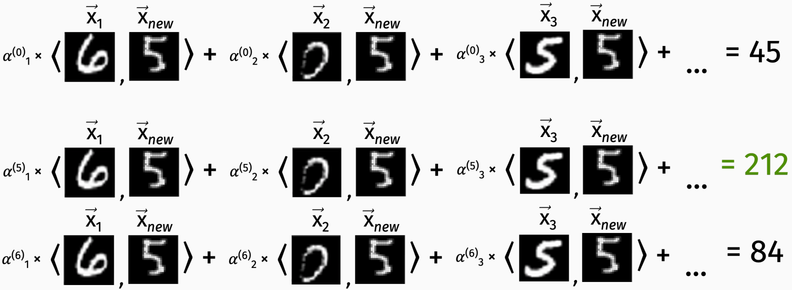
$$\langle \vec{\beta}^{(6)}, \vec{x}_{new} \rangle = 84$$

NEW VIEW OF LOGISTIC REGRESSION

$\vec{x}_1$	$\vec{x}_2$	$\vec{x}_3$	$\vec{x}_4$	$\vec{x}_5$	$\vec{x}_6$	
						$\cdots$
$\alpha^{(0)}{}_1 = .4$	$\alpha^{(0)}{}_2 = .9$	$\alpha^{(0)}{}_3 = .2$	$\alpha^{(0)}{}_4 = .1$	$\alpha^{(0)}{}_5 = .8$	$\alpha^{(0)}{}_6 = .05$	$\cdots$
$\alpha^{(1)}{}_1 = .05$	$\alpha^{(1)}{}_2 = .1$	$\alpha^{(1)}{}_3 = .2$	$\alpha^{(1)}{}_4 = .02$	$\alpha^{(1)}{}_5 = .1$	$\alpha^{(1)}{}_6 = .95$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\alpha^{(5)}{}_1 = .1$	$\alpha^{(5)}{}_2 = .2$	$\alpha^{(5)}{}_3 = .85$	$\alpha^{(5)}{}_4 = .75$	$\alpha^{(5)}{}_5 = .1$	$\alpha^{(5)}{}_6 = .05$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Learn *n* length parameter vectors $\boldsymbol{\alpha}^{(0)}, \ldots, \boldsymbol{\alpha}^{(9)}$, one for each class.

NEW VIEW OF LOGISTIC REGRESSION



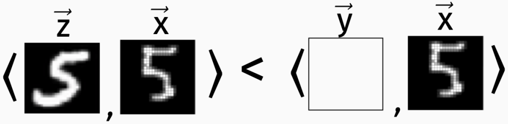
$\alpha^{(0)}{}_1 \times \langle \vec{x}_1, \vec{x}_{new} \rangle + \alpha^{(0)}{}_2 \times \langle \vec{x}_2, \vec{x}_{new} \rangle + \alpha^{(0)}{}_3 \times \langle \vec{x}_3, \vec{x}_{new} \rangle + \ldots = 45$
$\alpha^{(5)}{}_1 \times \langle \vec{x}_1, \vec{x}_{new} \rangle + \alpha^{(5)}{}_2 \times \langle \vec{x}_2, \vec{x}_{new} \rangle + \alpha^{(5)}{}_3 \times \langle \vec{x}_3, \vec{x}_{new} \rangle + \ldots = 212$
$\alpha^{(6)}{}_1 \times \langle \vec{x}_1, \vec{x}_{new} \rangle + \alpha^{(6)}{}_2 \times \langle \vec{x}_2, \vec{x}_{new} \rangle + \alpha^{(6)}{}_3 \times \langle \vec{x}_3, \vec{x}_{new} \rangle + \ldots = 84$

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

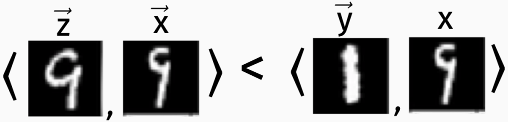
Assign $\mathbf{x}_{new}$ to the class with highest score.

DIVING INTO SIMILARITY

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):


$\langle \vec{z}, \vec{x} \rangle < \langle \vec{y}, \vec{x} \rangle$

But clearly the first image is more similar.


$\langle \vec{z}, \vec{x} \rangle < \langle \vec{y}, x \rangle$

Here's a more realistic scenario.

KERNEL FUNCTIONS: PERSPECTIVE ONE

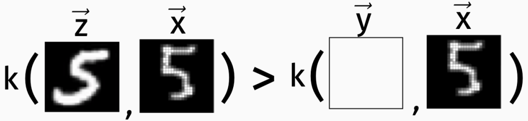
A kernel function $k(\mathbf{x}, \mathbf{y})$ is simply a similarity measure between data points.

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

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

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

for some scaling factor σ.



KERNEL FUNCTIONS: PERSPECTIVE ONE

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

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

But you can imagine much more complex similarity metrics.

KERNEL FUNCTIONS: PERSPECTIVE TWO

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 $\boldsymbol{\alpha}$ to minimize $LL(\mathbf{y}, \mathbf{X}^T\boldsymbol{\alpha})$ where $LL()$ denotes the logistic loss. Then we classified via:

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

The inner product similarity came from the fact that our predictions were based on the linear function $\langle \mathbf{x}_{new}, \mathbf{X}^T\boldsymbol{\alpha} \rangle$.

KERNEL FUNCTIONS AS FEATURE TRANSFORMATION

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

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

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

KERNEL FUNCTIONS AND FEATURE TRANSFORMATION

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

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

$$\begin{aligned}(\mathbf{x}^T\mathbf{w} + 1)^2 &= (x_1y_1 + x_2y_2 + x_3y_3 + 1)^2 \\ &= 1 + 2x_1w_1 + 2x_2w_2 + 2x_3w_3 + x_1^2w_1^2 + x_2^2w_2^2 + x_3^2w_3^2 \\ &\quad + 2x_1w_1x_2w_2 + 2x_1w_1x_3w_3 + 2x_2w_2x_3w_3 \\ &= \phi(\mathbf{x})^T\phi(\mathbf{w}).\end{aligned}$$

KERNEL FUNCTIONS AND FEATURE TRANSFORMATION

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

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

And there are many more...

KERNEL FUNCTIONS AND FEATURE TRANSFORMATION

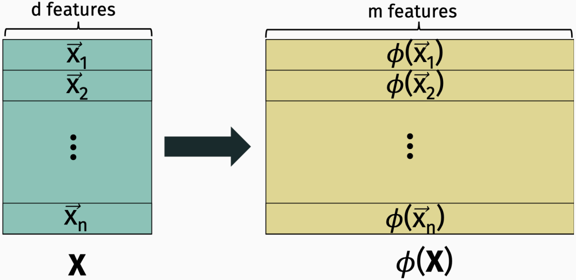
Sometimes $\phi(\vec{x})$ is simple and explicit. **More often, it is not.**

As we will discuss shortly, it doesn't necessarily matter – we often don't even need to know ϕ.

KERNEL FUNCTIONS AND FEATURE TRANSFORMATION

Feature transformations $\Longleftrightarrow$ new similarity metrics.

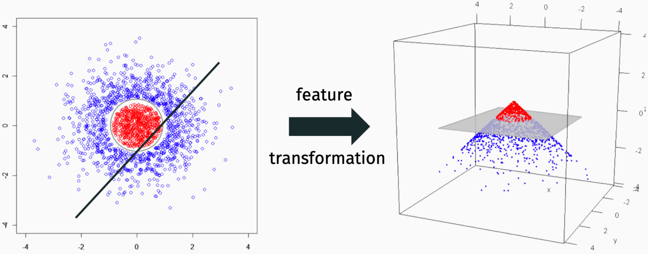
Using $k(\cdot,\cdot)$ in place of the inner product $\langle\cdot,\cdot\rangle$ is equivalent to replacing every data point $\mathbf{x}_1,\ldots,\mathbf{x}_n$ by $\phi(\mathbf{x}_1),\ldots,\phi(\mathbf{x}_n)$.[3]



[3] 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*. Typically you need to use regularization.

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.


feature transformation

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

TAKEAWAY TWO

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.

KERNEL LOGISTIC REGRESSION

Standard logisitic regression

Loss function:

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

Gradient:

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

Prediction:

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

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

Kernel logisitic regression

Loss function:

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

Gradient:

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

Prediction:

$$z = \sum_{j=1}^{n} \alpha_j \langle \phi(\mathbf{x}_{new}), \phi(\mathbf{x}_j) \rangle$$

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

KERNEL REGRESSION

Standard linear regression

Loss function:

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

Gradient:

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

Prediction:

$$y_{new} = \sum_{j=1}^{n} \alpha_j \cdot \langle \mathbf{x}_{new}, \mathbf{x}_j \rangle.$$

Kernel linear regression

Loss function:

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

Gradient:

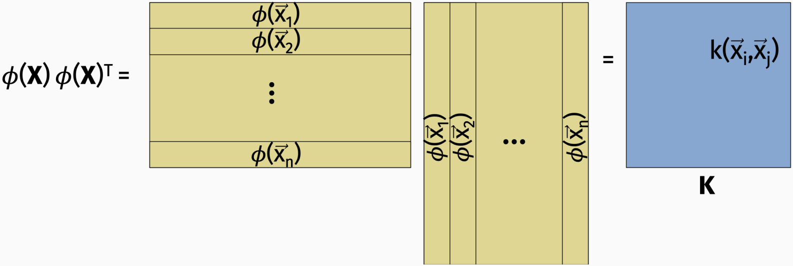
$$\nabla L(\boldsymbol{\alpha}) = 2\phi(\mathbf{X})\phi(\mathbf{X})^T(\phi(\mathbf{X})\phi(\mathbf{X})^T\alpha - \mathbf{y}).$$

Prediction:

$$y_{new} = \sum_{j=1}^{n} \alpha_j \cdot \langle \phi(\mathbf{x}_{new}), \phi(\mathbf{x}_j) \rangle.$$

KERNEL MATRIX

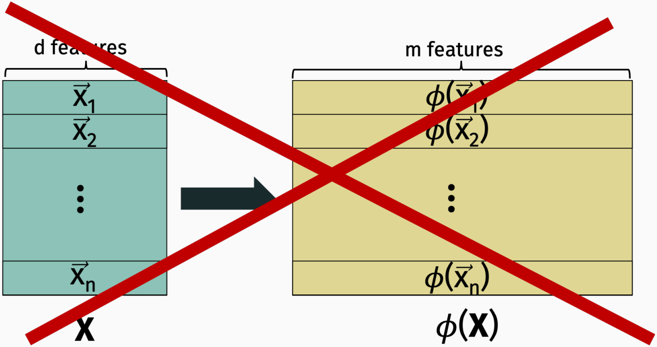
$K = \phi(X)\phi(X)^T$ is called the kernel Gram matrix.



KERNEL TRICK

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

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



KERNEL TRICK

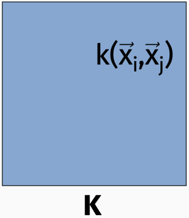
This can lead to significant computational savings!

- 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*. So kernel trick is your only option.

Added benefit: Relatively numerically stable. E.g. is a much better option for performing multivariate or even single variate polynomial regression or classification.

BEYOND THE KERNEL TRICK

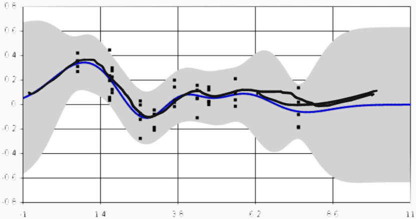
The kernel matrix **K** is still $n \times n$ though which is huge when the size of the training set n is large. Has made the kernel trick less appealing in some modern ML applications.

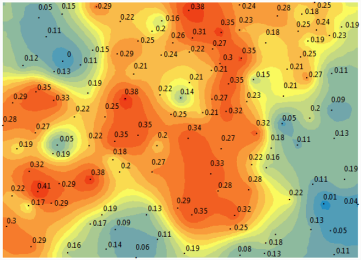


Many algorithmic advances in recent years partially address this computational challenge (random Fourier features methods, Nystrom methods, etc.)[4]

[4] This was a major topic of my research 3-5 years ago.

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.