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

NYU Tandon School of Engineering, Prof. Christopher Musco

- Previous methods studied (regression, logistic regression) are considered <u>linear</u> 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 <u>non-linear</u> methods. Specifically, kernel methods and neural networks.
- Both are very closely related to feature transformations!

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

Training data: $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_1, \ldots, y_n \in \{1, \ldots, q\}$.

Classification algorithm:

Given new input $x_{new}$,

- Compute $sim(x_{new}, x_1), \ldots, sim(x_{new}, x_n)$.[1]
- Let $x_{j_1}, \ldots, x_{j_k}$ be the training data vectors with highest similarity to $x_{new}$.
- Predict $y_{new}$ as $majority(y_{j_1}, \ldots, y_{j_k})$.

--------------------------------

[1]$sim(x_{new}, x_i)$ is any chosen <u>similarity function</u>, like $1 - \|x_{new} - x_i\|_2$.
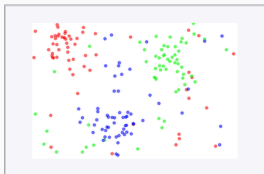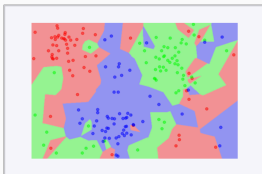
Fig. 1. The dataset.
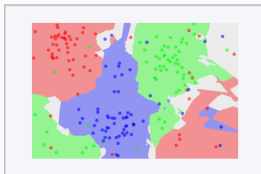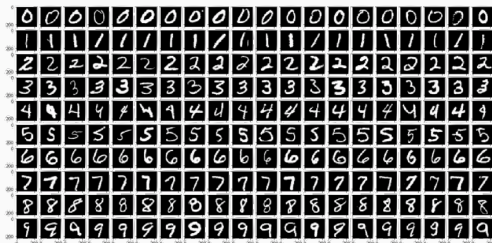
Fig. 2. The 1NN classification map.

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.

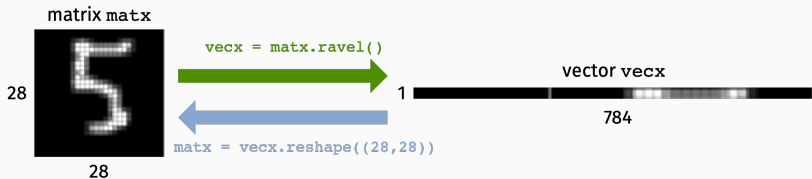Especially good for large datasets with lots of repetition. Works well on MNIST for example:



$\approx 95\%$ **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!

Each pixel is number from $[0, 1]$. 0 is black, 1 is white.
Represent $28 \times 28$ matrix of pixel values as a flattened vector.

matrix `matx`

```
vecx = matx.ravel()
```

vector `vecx`

28

1

784

```
matx = vecx.reshape((28,28))
```
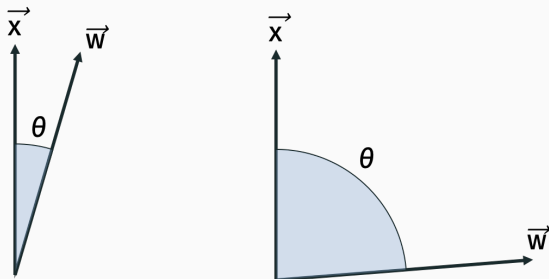
28

```
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])
```

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".

Connection to Euclidean ($\ell_2$) 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 between MNIST digits:
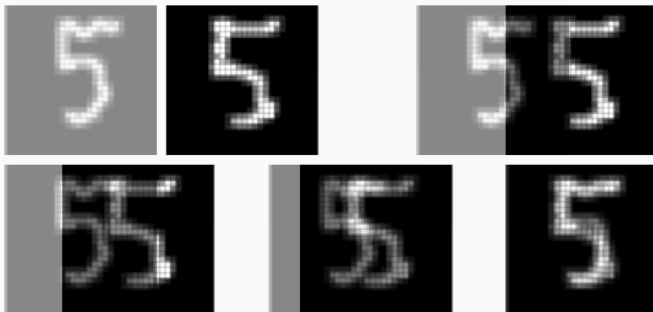


$$\langle x, w \rangle = \sum_{i=1}^{28} \sum_{j=1}^{28} \mathtt{matx}[i,j] \cdot \mathtt{matw}[i,j].$$

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.
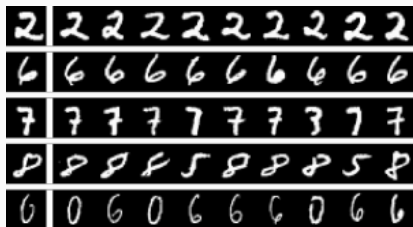
Visualizing the inner product between two images:



Images with high inner product have a lot of overlap.

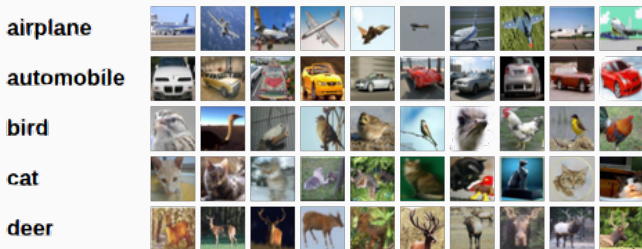Most similar images during $k$-nn search, $k = 9$:

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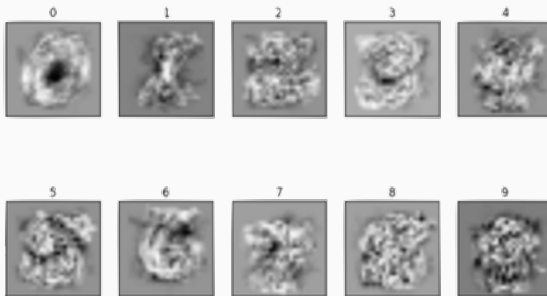
One-vs.-all or Multiclass Cross-entropy Classification with Logistic Regression:

- Learn $q$ classifiers with parameters $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \dots, \boldsymbol{\beta}^{(q)}$.
- Given $\mathbf{x}_{new}$ compute $\langle \mathbf{x}_{new}, \boldsymbol{\beta}^{(1)} \rangle, \dots, \langle \mathbf{x}_{new}, \boldsymbol{\beta}^{(q)} \rangle$
- 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 $28 \times 28$ image.

Visualizing $\beta_1, \ldots, \beta_q$:



For an input image , compute <u>inner product</u> 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 $n$.

### Logistic Regression Model:

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

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

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

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

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

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

$$\boldsymbol{\beta} = \mathsf{X}^T \boldsymbol{\alpha}.$$
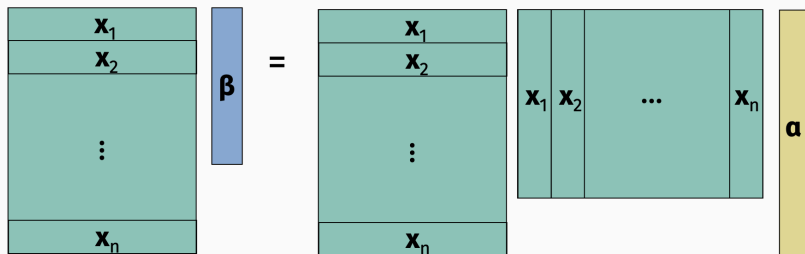
**Logistic Regression Equivalent Formulation:**

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

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

Can still be minimized via gradient descent:

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

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

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

Score for class $i$:

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

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

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

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

$\vec{x}_1$     $\vec{x}_2$     $\vec{x}_3$     $\vec{x}_4$     $\vec{x}_5$     $\vec{x}_6$

$\vec{\alpha_0}[1] = .1$   $\vec{\alpha_0}[2] = .9$   $\vec{\alpha_0}[3] = -.2$   $\vec{\alpha_0}[4] = -.1$   $\vec{\alpha_0}[5] = .8$   $\vec{\alpha_0}[6] = -.05$ •••

$\vec{\alpha_1}[1] = -.05$   $\vec{\alpha_1}[2] = .1$   $\vec{\alpha_1}[3] = -.2$   $\vec{\alpha_1}[4] = -.02$   $\vec{\alpha_1}[5] = -.1$   $\vec{\alpha_1}[6] = .95$ •••

$\vdots$    $\vdots$    $\vdots$    $\vdots$    $\vdots$    $\vdots$

$\vec{\alpha_5}[1] = .1$   $\vec{\alpha_5}[2] = -.2$   $\vec{\alpha_5}[3] = .85$   $\vec{\alpha_5}[4] = .75$   $\vec{\alpha_5}[5] = .1$   $\vec{\alpha_5}[6] = -.05$ •••

$\vdots$    $\vdots$    $\vdots$    $\vdots$    $\vdots$    $\vdots$

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

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

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

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

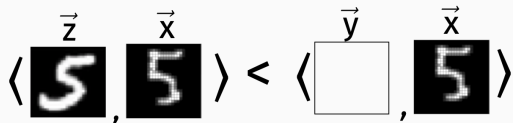Classification looks similar to *k*-NN: we compute the similarity between $x_{new}$ and every other vector in our training data set. A weighted sum of the similarities leads to scores for each class.

Assign $x_{new}$ to the class with highest score.

22

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.

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 $\sigma$.

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.

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 $\langle \mathbf{x}_{new}, \mathbf{X}^T\boldsymbol{\alpha}\rangle$.

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, $k(\mathbf{x}, \mathbf{w}) = (\mathbf{x}^T\mathbf{w} + 1)^2$.

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

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

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: $k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^q$.
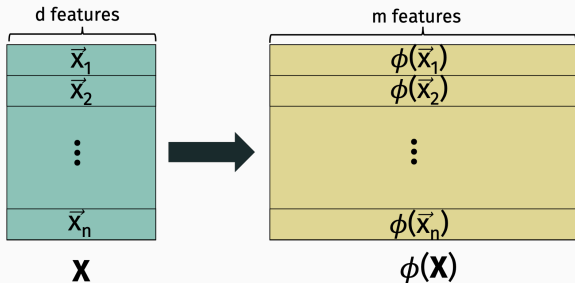
And there are many more...

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 $\phi$.

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 $x_1, \ldots, x_n$ by $\phi(x_1), \ldots, \phi(x_n)$.[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.

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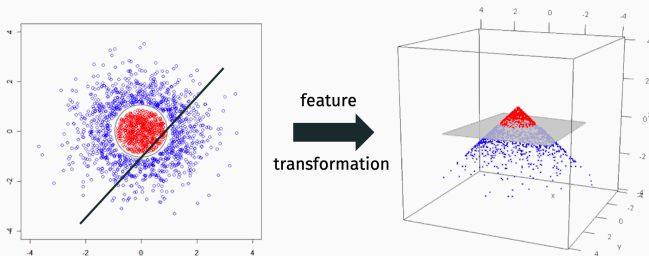


When you add features, it becomes possible to learn more complex decision boundaries (in this case a circle) with a linear classifier.

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   Kernel 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:

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

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

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:

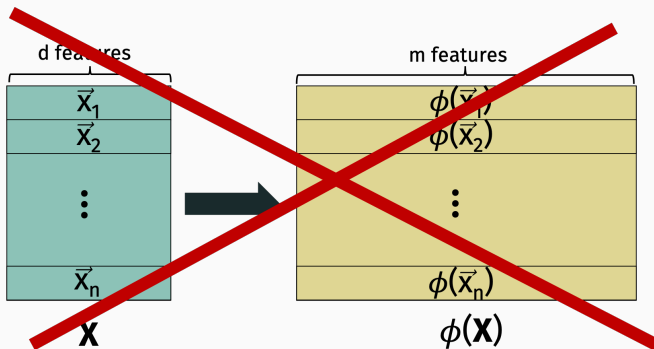$$y_{new} = \sum_{j=1}^{n} \alpha_j \cdot \langle \mathbf{x}_{new}, \mathbf{x}_j \rangle.$$

### 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:

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

$\mathsf{K} = \phi(\mathsf{X})\phi(\mathsf{X})^T$ is called the <u>kernel Gram matrix</u>.

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 $i$.

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.

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.



$k(\vec{x}_i, \vec{x}_j)$
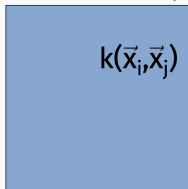
**K**

There is an inherent quadratic dependence on $n$ in the computational and space complexity of kernel methods.

- $10,000$ data points $\rightarrow$ runtime scales as $\sim 100,000,000$, **K** takes 800MB of space.
- $1,000,000$ data points $\rightarrow$ runtime scales as $\sim 10^{12}$, **K** takes 8TB of space.

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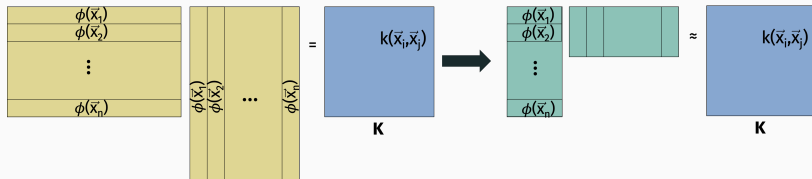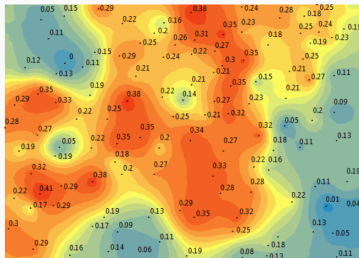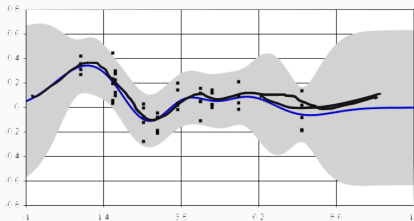


Often based on "reversing" the kernel trick to find a compact feature set that well approximates the kernel.[4]

---

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

We won't study kernel <u>regression</u> 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. Can be justified from a Bayesian modeling perspective.

Reformulation of linear regression:

$$\min_{\boldsymbol{\beta}} \|X\boldsymbol{\beta} - y\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \rightarrow \min_{\boldsymbol{\alpha}} \|XX^T\boldsymbol{\alpha} - y\|_2^2 + \lambda\|X^T\boldsymbol{\alpha}\|_2^2$$

Replace $XX^T$ by kernel matrix $K$ during training.

Prediction:

$$y_{new} = \sum_{i=1}^{n} \alpha_i \cdot k(x_{new}, x_i).$$

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