CS-UY 4563: Lecture 12
k-Nearest Neighbors, Kernel Methods

NYU Tandon School of Engineering, Prof. Christopher Musco

- Lab 4 due on **Friday, at 11:59pm**. Requires correct solution to HW3, Problem 2. I will post this after class.
- Short lab on Gradient Descent will be released soon and due after break.
- Upcoming labs involve image data and require more programming. Made up with lighter written homework.

Break is a great time to start mulling over ideas for your course project! Details in `project_guidelines.pdf`.

1. Find or collect a data set.
2. Ask a question (or two) about the data set which can possibly be answered with machine learning.
3. Apply tools and techniques learned in the class to answering that question.

- Must work in **groups of 2**. Coordinate over Piazza if looking for a partner.
- Any data set or topic is allowed, but youo should not reproduce an analysis that has already been done! Ask a new question or take a new approach.
- Talk to me or the TA's early if you are stuck on coming up with an idea, or need help narrowing down options.

4/1, Choose Project Partner and Topic. Email me.

4/2,4/6-4/8, Schedule Mandatory Meeting. Claim a time-slot in the Google Doc linked in the project information document.

4/13, Project Proposal Due. 2 Pages. Need to have dataset finalized!

5/6, 5/11, Project Presentations in Class. 5 Minutes.

5/11, Final Report Due 4+ Pages.

Look at your data! Plot features, examine full examples, look for missing data or inconsistencies.

Start small. Test and debug code on a small subset of your data before running on the whole thing.

Start simple. Try the simplest methods first. Linear regression, naive Bayes, etc. Even simpler: for regression, predict using mean($\vec{y}$). For classification predict using max $\vec{y}$ (the most common label). **You need to develop a baseline to compare your methods against.**

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:

majority$(1, 5, 5, 10, 5, 6, 6)$
$= 5$

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

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

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

k-NEAREST NEIGHBOR METHOD

xnew

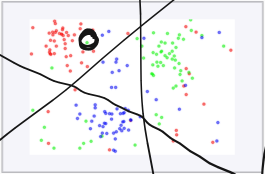


Fig. 1. The dataset.

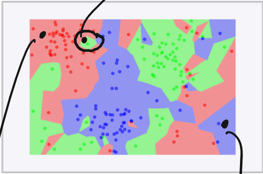


Fig. 2. The 1NN classification map.

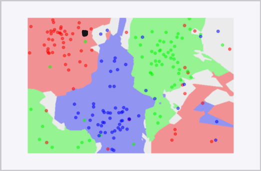


Fig. 3. The 5NN classification map.

Xnew

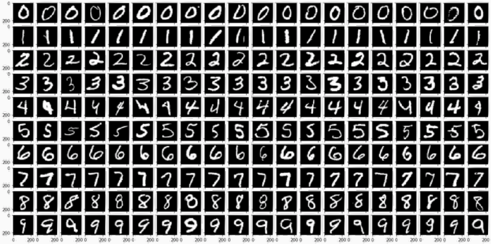
xnew

- 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]

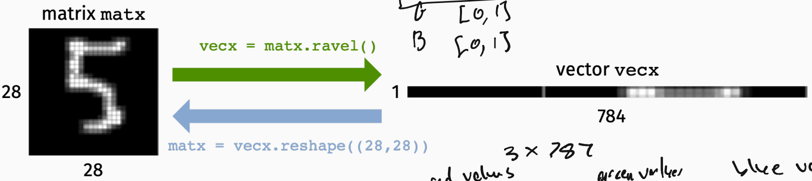
Let's look into this example a bit more...

[2] Can be improved to 99.5% with some simple tricks!

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

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

B [0,1]
G [0,1]
B [0,1]



3 × 787

red values
green values
blue values

I = B + B + G

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

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

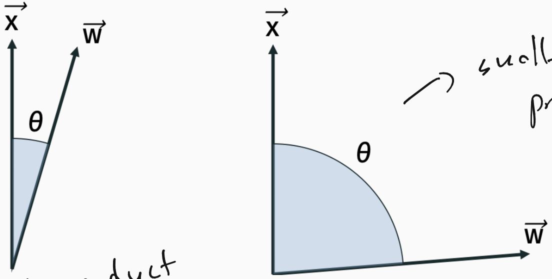
```
xvec = xmat.ravel()
```

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

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

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

smaller dot product



larger dot product

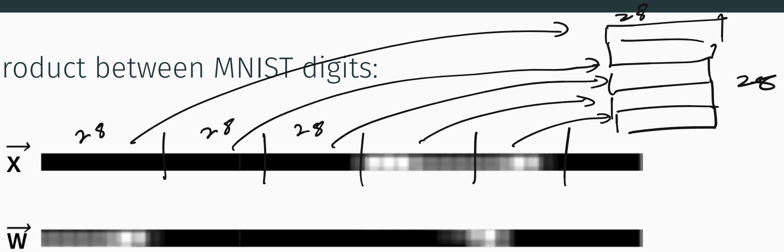
INNER PRODUCT SIMILARITY

Connection to Euclidean (ℓ_2) Distance:

$$\|\vec{x} - \vec{w}\|_2^2 = \overset{1}{\|\vec{x}\|_2^2} + \overset{1}{\|\vec{w}\|_2^2} - 2\langle \vec{x}, \vec{w} \rangle$$

For a set of vectors with the same norm, the pair of vectors with largest inner product is the pair with smallest Euclidean distance.

Inner product between MNIST digits:



np.sum (np.sum (matx * matw))

Not matrix mult

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

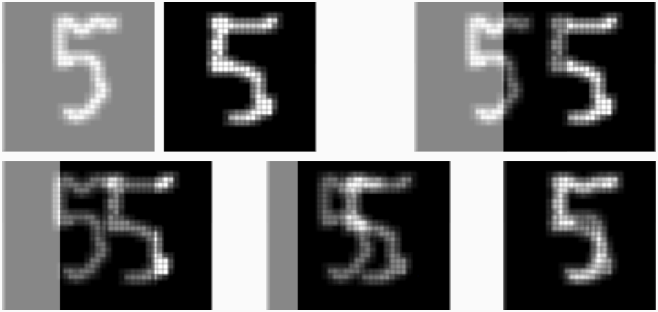
@

$$= \sum_{j=1}^{784} x[j]\, w[j]$$

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:

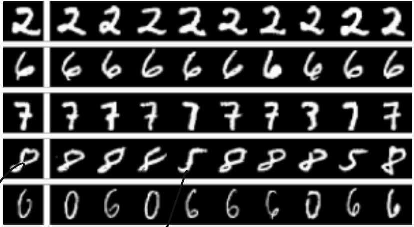
mat w $\longrightarrow$ mat x



Images with high inner product have a lot of overlap.

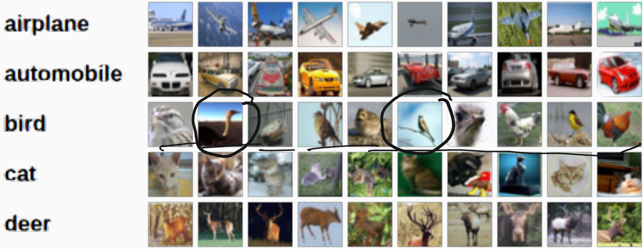
Most similar images during k-nn search, $k = 9$:

$\vec{x}_{new}$ x_{j_1} x_{j_9}



maybe most similar to

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.

computed with logistic regression to minimize train loss

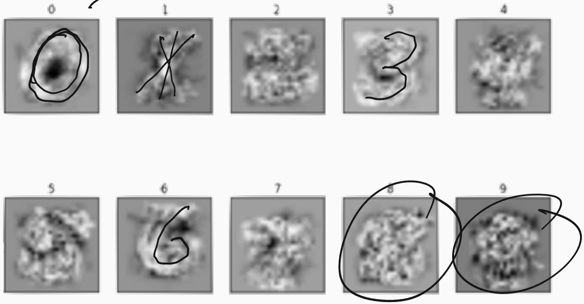
One-vs.-all Classification with Logistic Regression:

- Learn q classifiers with parameters $\vec{\beta}_1, \vec{\beta}_2, \ldots, \vec{\beta}_q$.
- Given $\vec{x}_{new}$ compute $\langle \vec{x}_{new}, \vec{\beta}_1 \rangle, \ldots, \langle \vec{x}_{new}, \vec{\beta}_q \rangle$
- 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 $\vec{\beta}_1, \ldots, \vec{\beta}_q$: parameter vector for separating 0s from dataset



For an input image  Xnew, 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.

$X \quad \beta \quad = \quad \rightarrow \langle \vec{x}_i, \vec{\beta} \rangle$

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:

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

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

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

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:

$X^T\alpha$

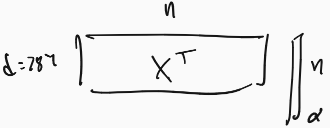
$$\vec{y} \qquad \text{and} \qquad h(\mathbf{X}\vec{\beta})$$

$X\beta$

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:

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



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:

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

Can still be minimized via gradient descent:

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

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

What does classification for a new point $\vec{x}_{new}$ look like?

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

$$x_{new}^T (X^T \alpha) = \langle x_{new}, X^T \alpha \rangle$$

$$(x_{new}^T X^T)\alpha = \langle X x_{new}, \alpha \rangle$$

$$= (X x_{new})^T \alpha =$$

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

Similar to $k-NN$ classifier but we learn a weight α_i for every $\vec{x}_i$ in our training set – can be positive or negative.

KERNEL FUNCTIONS

KERNEL FUNCTIONS

KERNEL FUNCTIONS