Lecture 2:
– Nearest Neighbour Classifier
Your 1st Classifier: Nearest Neighbor Classifier
Concept Learning

- **Definition:** Acquire an operational definition of a general category of objects given *positive* and *negative* training examples.

- Also called *binary classification*, *binary supervised learning*
Concept Learning Example

- **Instance Space** $X$: Set of all possible objects describable by attributes (often called *features*).

- **Concept** $c$: Subset of objects from $X$ ($c$ is unknown).

- **Target Function** $f$: Characteristic function indicating membership in $c$ based on attributes (i.e. *label*) ($f$ is unknown).

- **Training Data** $S$: Set of instances labeled with target function.
Concept Learning as Learning
A Binary Function

• **Task**
  – Learn (to imitate) a function $f : X \rightarrow \{+1,-1\}$

• **Training Examples**
  – Learning algorithm is given the correct value of the function for particular inputs $\rightarrow$ training examples
  – An example is a pair $(x, y)$, where $x$ is the input and $y = f(x)$ is the output of the target function applied to $x$.

• **Goal**
  – Find a function $h : X \rightarrow \{+1,-1\}$
    that approximates $f : X \rightarrow \{+1,-1\}$
    as well as possible.
Supervised Learning

• **Task**
  – Learn (to imitate) a function \( f: X \rightarrow Y \)

• **Training Examples**
  – Learning algorithm is given the correct value of the function for particular inputs \( \rightarrow \) training examples
  – An example is a pair \((x, f(x))\), where \( x \) is the input and \( y = f(x) \) is the output of the target function applied to \( x \).

• **Goal**
  – Find a function \( h: X \rightarrow Y \) that approximates \( f: X \rightarrow Y \) as well as possible.
Supervised / Inductive Learning

- **Given**
  - examples of a function \((x, f(x))\)

- **Predict function** \(f(x)\) for new examples \(x\)
  - Discrete \(f(x)\): Classification
  - Continuous \(f(x)\): Regression
  - \(f(x) = \text{Probability}(x)\): Probability estimation
Appropriate Applications for Supervised Learning

- Situations where there is no human expert
  \( x \): Bond graph for a new molecule.
  \( f(x) \): Predicted binding strength to AIDS protease molecule.

- Situations where humans can perform the task but can’t describe how they do it.
  \( x \): Bitmap picture of hand-written character
  \( f(x) \): Ascii code of the character

- Situations where the desired function is changing frequently
  \( x \): Description of stock prices and trades for last 10 days.
  \( f(x) \): Recommended stock transactions

- Situations where each user needs a customized function \( f \)
  \( x \): Incoming email message.
  \( f(x) \): Importance score for presenting to user (or deleting without presenting).
Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→

cat
The problem: semantic gap

Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)
Challenges: Viewpoint Variation
Challenges: Illumination
Challenges: Deformation
Challenges: Occlusion
Challenges: Background clutter
Challenges: Intraclass variation
An image classifier

```python
def predict(image):
    # ?????
    return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.
Attempts have been made
Data-driven approach:
1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```python
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```
First classifier: **Nearest Neighbor Classifier**

```python
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```

- Remember all training images and their labels
- Predict the label of the most similar training image
Example dataset: **CIFAR-10**

10 labels

50,000 training images, each image is tiny: 32x32

10,000 test images.
Example dataset: **CIFAR-10**

- **10 labels**
- **50,000** training images
- **10,000** test images.

For every test image (first column), examples of nearest neighbors in rows.
How do we compare the images? What is the **distance metric**?

**L1 distance:**

\[ d_1(I_1, I_2) = \sum_p |I_{1p}^p - I_{2p}^p| \]

<table>
<thead>
<tr>
<th>test image</th>
<th>training image</th>
<th>pixel-wise absolute value differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 32 10 18</td>
<td>10 20 24 17</td>
<td>46 12 14 1</td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td>82 13 39 33</td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td>12 10 0 30</td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td>2 32 22 108</td>
</tr>
</tbody>
</table>

**add** 456
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
        # find the nearest training image to the i'th test image
        # using the L1 distance (sum of absolute value differences)
        distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
        min_index = np.argmin(distances) # get the index with smallest distance
        Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred

Nearest Neighbor classifier
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """X is N x D where each row is an example. Y is 1-dimensional of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier

remember the training data
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in range(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier

for every test image:
- find nearest train image with L1 distance
- predict the label of nearest training image
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # let's make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Q: how does the classification speed depend on the size of the training data?
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        num_test = X.shape[0]
        # let's make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i, :]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Q: how does the classification speed depend on the size of the training data?
linearly :(
Aside: Approximate Nearest Neighbor

find approximate nearest neighbors quickly

ANN: A Library for Approximate Nearest Neighbor Searching
David M. Mount and Sunil Arya
Version 1.1.2
Release Date: Jan 27, 2010

What is ANN?
ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.

FLANN - Fast Library for Approximate Nearest Neighbors

What is FLANN?
FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

FLANN is written in C++ and contains bindings for the following languages: C, MATLAB and Python.

News
- (14 December 2012) Version 1.8.0 is out bringing incremental addition/removal of points to/from indexes
- (20 December 2011) Version 1.7.0 is out bringing two new index types and several other improvements.
- You can find binary installers for FLANN on the Point Cloud Library project page. Thanks to the PCL developers!
- Mac OS X users can install from MacPorts (thanks to Mark Moir for maintaining the Portfile)
- New release introducing an easier way to use custom distances, kd-tree implementation optimized for low dimensionality and outermost MPI support
- New release introducing new C++ templated API, thread-safe search, save/load of indexes and more.
- The FLANN license was changed from LGPL to BSD.

How fast is it?
In our experiments we have found FLANN to be about one order of magnitude faster on many datasets (in query time), than previously available approximate nearest neighbor search software.

Publications
More information and experimental results can be found in the following papers:
- Markus Muja and David G. Lowe, "Fast Matching of Binary Features", Conference on Computer and Robot Vision (CRV) 2012. [PDF] [BibTeX]
- Markus Muja and David G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration", In International Conference on Computer Vision Theory and Applications (VISAPP19), 2006 [PDF] [BibTeX]
The choice of distance is a hyperparameter common choices:

**L1 (Manhattan) distance**

\[ d_1(I_1, I_2) = \sum_P |I_1^p - I_2^p| \]

**L2 (Euclidean) distance**

\[ d_2(I_1, I_2) = \sqrt{\sum_P (I_1^p - I_2^p)^2} \]
k-Nearest Neighbor

find the k nearest images, have them vote on the label

K-Nearest Neighbor (kNN)

• Given: Training data \( \{(x_1,y_1),\ldots,(x_n,y_n)\} \)
  – Attribute vectors: \( x_i \in X \)
  – Labels: \( y_i \in Y \)

• Parameter:
  – Similarity function: \( K : X \times X \to R \)
  – Number of nearest neighbors to consider: \( k \)

• Prediction rule
  – New example \( x' \)
  – K-nearest neighbors: \( k \) train examples with largest \( K(x_i, x') \)

\[
h(x') = \arg \max_{y \in Y} \left\{ \sum_{i \in \text{knn}(x')} 1_{[y_i = y]} \right\}
\]
1-Nearest Neighbor
4-Nearest Neighbors
4-Nearest Neighbors Sign
Example dataset: CIFAR-10
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows.
What is the best distance to use?
What is the best value of $k$ to use?

i.e. how do we set the hyperparameters?

We will talk about this later!
If we get more data

• 1 Nearest Neighbor
  - Converges to perfect solution if clear separation
  - Twice the minimal error rate $2p(1-p)$ for noisy problems

• k-Nearest Neighbor
  - Converges to perfect solution if clear separation (but needs more data)
  - Converges to minimal error $\min(p, 1-p)$ for noisy problems if $k$ increases
Demo
Weighted K-Nearest Neighbor

• Given: Training data \{(x_1,y_1),\ldots, (x_n,y_n )\}
  – Attribute vectors: \( x_i \in X \)
  – Target attribute \( y_i \in Y \)

• Parameter:
  – Similarity function: \( K : X \times X \rightarrow R \)
  – Number of nearest neighbors to consider: \( k \)

• Prediction rule
  – New example \( x' \)
  – K-nearest neighbors: \( k \) train examples with largest \( K(x_i, x') \)

\[
h(x') = \arg\max_{y \in Y} \left\{ \sum_{i \in knn(x')} 1[y_i = y] K(x_i, x') \right\}
\]
More Nearest Neighbors in Visual Data
Where in the World?  [Hays & Efros, CVPR 2008]

A nearest neighbor recognition example
Where in the World? [Hays & Efros, CVPR 2008]
Where in the World? [Hays & Efros, CVPR 2008]
6+ million geotagged photos by 109,788 photographers

Annotated by Flickr users
Scene Matches
Scene Matches
Scene Matches
The Importance of Data

Percentage of Geolocations within 200km

- First Nearest Neighbor Scene Match
- Chance - Random Scenes

Database size (thousands of images, log scale)
Scene Completion [Hays & Efros, SIGGRAPH07]

Original

Input

Scene Matches

Output
Context Matching
Graph cut + Poisson blending
Weighted K-NN for Regression

• Given: Training data \{(x_1,y_1),\ldots, (x_n,y_n)\}
  – Attribute vectors: \(x_i \in X\)
  – Target attribute \(y_i \in \mathcal{R}\)

• Parameter:
  – Similarity function: \(K : X \times X \rightarrow \mathcal{R}\)
  – Number of nearest neighbors to consider: \(k\)

• Prediction rule
  – New example \(x'\)
  – K-nearest neighbors: \(k\) train examples with largest \(K(x_i,x')\)

\[
h(x') = \frac{\sum_{i \in \text{knn}(x')} y_i K(x_i, x')}{\sum_{i \in \text{knn}(x')} K(x_i, x')}
\]
Collaborative Filtering

<table>
<thead>
<tr>
<th>Rating Matrix</th>
<th>m₁</th>
<th>m₂</th>
<th>m₃</th>
<th>m₄</th>
<th>m₅</th>
<th>m₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>u₁</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>u₂</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>u₃</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>u</td>
<td>?</td>
<td>1</td>
<td>4</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Overview of Nearest Neighbors

• Very simple method

• Retain all training data
  - Can be slow in testing
  - Finding NN in high dimensions is slow

• Metrics are very important

• Good baseline
Next Class:
Linear Regression and Least Squares