Outline

1. Administration
2. First learning algorithm: Nearest neighbor classifier
3. More deep understanding about NNC
4. Some practical sides of NNC
5. What we have learned
Entrance exam

- All were graded
  - about 87% has passed
  - 96 students in Prof. Sha’s section and 48 in Prof. Liu’s section
- Those who have passed the threshold were granted D-clearance – please enroll asap by Friday noon.
- In several cases, advisors had sent out inquiries – please respond by Thursday noon per their instructions.
- If not being granted D-clearance, or being contacted by advisors, you can assume that you are not permitted to enroll.
- You can ask TAs to check up your grades, only if you strongly believe you did well.
Outline

1. Administration

2. First learning algorithm: Nearest neighbor classifier
   - Intuitive example
   - General setup for classification
   - Algorithm

3. More deep understanding about NNC

4. Some practical sides of NNC

5. What we have learned
Recognizing flowers

Types of Iris: setosa, versicolor, and virginica
Measuring the properties of the flowers

Features and attributes: the widths and lengths of sepal and petal
Pairwise scatter plots of 131 flower specimens

Visualization of data helps to identify the right learning model to use

Each colored point is a flower specimen: setosa, versicolor, virginica
Different types seem well-clustered and separable

Using two features: petal width and sepal length
Labeling an unknown flower type

Closer to red cluster: so labeling it as setosa
Multi-class classification

Classify data into one of the multiple categories

- Input (feature vectors): $\mathbf{x} \in \mathbb{R}^D$
- Output (label): $y \in [C] = \{1, 2, \ldots, C\}$
- Learning goal: $y = f(\mathbf{x})$

Special case: binary classification

- Number of classes: $C = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$
More terminology

Training data (set)
- N samples/instances: \( D^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\} \)
- They are used for learning \( f(\cdot) \)

Test (evaluation) data
- M samples/instances: \( D^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_M, y_M)\} \)
- They are used for assessing how well \( f(\cdot) \) will do in predicting an unseen \( x \notin D^{\text{TRAIN}} \)

Training data and test data should not overlap: \( D^{\text{TRAIN}} \cap D^{\text{TEST}} = \emptyset \)
Often, data is conveniently organized as a table

**Ex: Iris data (click here for all data)**
- 4 features
- 3 classes

<table>
<thead>
<tr>
<th>Species</th>
<th>Sepal length</th>
<th>Sepal width</th>
<th>Petal length</th>
<th>Petal width</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>I. setosa</em></td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td><em>I. setosa</em></td>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
</tr>
</tbody>
</table>
First learning algorithm: Nearest neighbor classifier

Nearest neighbor classification (NNC)

Nearest neighbor

\[ x(1) = x_{\text{nn}(x)} \]

where \( \text{nn}(x) \in [N] = \{1, 2, \ldots, N\} \), i.e., the index to one of the training instances,

\[ \text{nn}(x) = \arg \min_{n \in [N]} \| x - x_n \|_2^2 = \arg \min_{n \in [N]} \sum_{d=1}^{D} (x_d - x_{nd})^2 \]

Classification rule

\[ y = f(x) = y_{\text{nn}(x)} \]
Visual example

In this 2-dimensional example, the nearest point to $x$ is a training instance, thus, $x$ will be labeled as red.
**Example: classify Iris with two features**

**Training data**

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category ($y$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setoas</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Flower with unknown category**

petal width = 1.8 and sepal width = 6.4

Calculating distance = $\sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

<table>
<thead>
<tr>
<th>ID</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.75</td>
</tr>
<tr>
<td>2</td>
<td>0.72</td>
</tr>
<tr>
<td>3</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Thus, the category is *versicolor* (the real category is *virginica*)
First learning algorithm: Nearest neighbor classifier

Decision boundary

For every point in the space, we can determine its label using the NNC rule. This gives rise to a *decision boundary* that partitions the space into different regions.

(b)
How to measure nearness with other distances?

Previously, we use the Euclidean distance

\[ \text{nn}(x) = \arg \min_{n \in [N]} \|x - x_n\|_2^2 \]

We can also use alternative distances
E.g., the following \( L_1 \) distance (i.e., city block distance, or Manhattan distance)

\[ \text{nn}(x) = \arg \min_{n \in [N]} \|x - x_n\|_1 = \arg \min_{n \in [N]} \sum_{d=1}^{D} |x_d - x_{nd}| \]

Green line is Euclidean distance. Red, Blue, and Yellow lines are \( L_1 \) distance
K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $$\text{nn}_1(x) = \arg\min_{n \in [N]} \| x - x_n \|_2^2$$
- 2nd-nearest neighbor: $$\text{nn}_2(x) = \arg\min_{n \in [N] - \text{nn}_1(x)} \| x - x_n \|_2^2$$
- 3rd-nearest neighbor: $$\text{nn}_2(x) = \arg\min_{n \in [N] - \text{nn}_1(x) - \text{nn}_2(x)} \| x - x_n \|_2^2$$

The set of K-nearest neighbor

$$\text{knn}(x) = \{ \text{nn}_1(x), \text{nn}_2(x), \cdots, \text{nn}_K(x) \}$$

Let $$x(k) = x_{\text{nn}_k(x)}$$, then

$$\| x - x(1) \|_2^2 \leq \| x - x(2) \|_2^2 \cdots \leq \| x - x(K) \|_2^2$$
How to classify with $K$ neighbors?

Classification rule

- Every neighbor votes: suppose $y_n$ (the true label) for $x_n$ is $c$, then
  - vote for $c$ is 1
  - vote for $c' \neq c$ is 0

We use the indicator function $\mathbb{I}(y_n == c)$ to represent.

- Aggregate everyone’s vote

$$v_c = \sum_{n \in \text{knn}(x)} \mathbb{I}(y_n == c), \quad \forall \quad c \in [C]$$

- Label with the majority

$$y = f(x) = \arg \max_{c \in [C]} v_c$$
First learning algorithm: Nearest neighbor classifier

Example

K=1, Label: red

K=3, Label: red

K=5, Label: blue
How to choose an optimal $K$?

When $K$ increases, the decision boundary becomes smooth.
Mini-summary

**Advantages of NNC**

- Computationally, simple and easy to implement – just computing the distance
- Theoretically, has strong guarantees “doing the right thing”

**Disadvantages of NNC**

- Computationally intensive for large-scale problems: $O(ND)$ for labeling a data point
- We need to “carry” the training data around. Without it, we cannot do classification. This type of method is called *nonparametric*.
- Choosing the right distance measure and $K$ can be involved.
Outline

1. Administration

2. First learning algorithm: Nearest neighbor classifier

3. More deep understanding about NNC
   - Measuring performance
   - The ideal classifier
   - Comparing NNC to the ideal classifier

4. Some practical sides of NNC

5. What we have learned
Is NNC too simple to do the right thing?

To answer this question, we proceed in 3 steps

1. We define a performance metric for a classifier/algorithm.
2. We then propose an ideal classifier.
3. We then compare our simple NNC classifier to the ideal one and show that it performs nearly as good.
How to measure performance of a classifier?

**Intuition**
We should compute **accuracy** — the percentage of data points being correctly classified, or the **error rate** — the percentage of data points being incorrectly classified.

**Two versions: which one to use?**
- Defined on the training data set
  \[
  A^{\text{TRAIN}} = \frac{1}{N} \sum_n \mathbb{I}[f(x_n) = y_n], \quad \varepsilon^{\text{TRAIN}} = \frac{1}{N} \sum_n \mathbb{I}[f(x_n) \neq y_n]
  \]
- Defined on the test (evaluation) data set
  \[
  A^{\text{TEST}} = \frac{1}{M} \sum_m \mathbb{I}[f(x_m) = y_m], \quad \varepsilon^{\text{TEST}} = \frac{1}{M} \sum_m \mathbb{I}[f(x_m) \neq y_m]
  \]
Example

Training data

What are $A^{\text{TRAIN}}$ and $\varepsilon^{\text{TRAIN}}$?
Example

Training data

What are $A^{\text{TRAIN}}$ and $\varepsilon^{\text{TRAIN}}$?

$A^{\text{TRAIN}} = 100\%$, $\varepsilon^{\text{TRAIN}} = 0\%$
Example

Training data

Test data

What are $A_{\text{TRAIN}}$ and $\varepsilon_{\text{TRAIN}}$?

$A_{\text{TRAIN}} = 100\%$, $\varepsilon_{\text{TRAIN}} = 0\%$

What are $A_{\text{TEST}}$ and $\varepsilon_{\text{TEST}}$?
Example

What are $A^{\text{TRAIN}}$ and $\varepsilon^{\text{TRAIN}}$?

$A^{\text{TRAIN}} = 100\%$, $\varepsilon^{\text{TRAIN}} = 0\%$

What are $A^{\text{TEST}}$ and $\varepsilon^{\text{TEST}}$?

$A^{\text{TEST}} = 0\%$, $\varepsilon^{\text{TEST}} = 100\%$
More deep understanding about NNC

Measuring performance

Leave-one-out (LOO)

**Idea**

- For each training instance $x_n$, take it out of the training set and then label it.
- For NNC, $x_n$’s nearest neighbor will not be itself. So the error rate would not become 0 necessarily.

![Training data]

What are the LOO-version of $A_{\text{TRAIN}}$ and $\varepsilon_{\text{TRAIN}}$?
Leave-one-out (LOO)

Idea

- For each training instance $x_n$, take it out of the training set and then label it.
- For NNC, $x_n$’s nearest neighbor will not be itself. So the error rate would not become 0 necessarily.

What are the LOO-version of $A^{\text{TRAIN}}$ and $\varepsilon^{\text{TRAIN}}$?

$A^{\text{TRAIN}} = 66.67\% (\text{i.e., } 4/6)$

$\varepsilon^{\text{TRAIN}} = 33.33\% (\text{i.e., } 2/6)$
Drawback of the metrics

They are dataset-specific

- Given a different training (or test) dataset, $A^{\text{TRAIN}}$ (or $A^{\text{TEST}}$) will change.
- Thus, if we get a dataset “randomly”, these variables would be random quantities.

$$A^{\text{TEST}}_{D_1}, A^{\text{TEST}}_{D_2}, \ldots, A^{\text{TEST}}_{D_q}, \ldots$$
Drawback of the metrics

They are dataset-specific

- Given a different training (or test) dataset, $A^{TRAIN}$ (or $A^{TEST}$) will change.
- Thus, if we get a dataset “randomly”, these variables would be random quantities.

$$A^{TEST}_{D_1}, A^{TEST}_{D_2}, \ldots, A^{TEST}_{D_q}, \ldots$$

These are called “empirical” accuracies (or errors).
More deep understanding about NNC

Measuring performance

Drawback of the metrics

They are dataset-specific

- Given a different training (or test) dataset, $A^{\text{TRAIN}}$ (or $A^{\text{TEST}}$) will change.
- Thus, if we get a dataset “randomly”, these variables would be random quantities.

$$A_{D_1}^{\text{TEST}}, A_{D_2}^{\text{TEST}}, \cdots, A_{D_q}^{\text{TEST}}, \cdots$$

These are called “empirical” accuracies (or errors).

Can we understand the algorithm itself in a “more certain” nature, by removing the uncertainty caused by the datasets?
Expected mistakes

Setup

• Assume our data \((x, y)\) is drawn from the joint and *unknown* distribution \(p(x, y)\)

• Classification mistake on a single data point \(x\) with the ground-truth label \(y\)

\[
L(f(x), y) = \begin{cases} 
0 & \text{if } f(x) = y \\
1 & \text{if } f(x) \neq y 
\end{cases}
\]
Expected mistakes

Setup

- Assume our data \((x, y)\) is drawn from the joint and \textit{unknown} distribution \(p(x, y)\).
- Classification mistake on a single data point \(x\) with the ground-truth label \(y\)
  \[
  L(f(x), y) = \begin{cases} 
  0 & \text{if } f(x) = y \\
  1 & \text{if } f(x) \neq y
  \end{cases}
  \]
- Expected classification mistake on a single data point \(x\)
  \[
  R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)
  \]
Expected mistakes

Setup

- Assume our data \((x, y)\) is drawn from the joint and unknown distribution \(p(x, y)\).
- Classification mistake on a single data point \(x\) with the ground-truth label \(y\)
  
  \[
  L(f(x), y) = \begin{cases} 
  0 & \text{if } f(x) = y \\
  1 & \text{if } f(x) \neq y
  \end{cases}
  \]

- Expected classification mistake on a single data point \(x\)
  
  \[
  R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)
  \]

- The average classification mistake by the classifier itself
  
  \[
  R(f) = \mathbb{E}_{x \sim p(x)} R(f, x) = \mathbb{E}_{(x,y) \sim p(x,y)} L(f(x), y)
  \]
Jargons

- $L(f(x), y)$ is called *0/1 loss function* — many other forms of loss functions exist for different learning problems.

$$
\text{Expected conditional risk} \quad R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)
$$

$$
\text{Expected risk} \quad R(f) = \mathbb{E}_{(x, y) \sim p(x, y)} L(f(x), y)
$$

$$
\text{Empirical risk} \quad R_D(f) = \frac{1}{N} \sum_{n} L(f(x_n), y_n)
$$

Obviously, this is our empirical error (rates).

Drs. Sha & Liu ({feisha, yanliu.cs}@usc.edu)

CSCI567 Machine Learning (Fall 2014)
Jargons

- $L(f(\mathbf{x}), y)$ is called 0/1 loss function — many other forms of loss functions exist for different learning problems.
- Expected conditional risk

$$R(f, \mathbf{x}) = \mathbb{E}_{y \sim p(y|\mathbf{x})} L(f(\mathbf{x}), y)$$
Jargons

- $L(f(x), y)$ is called 0/1 loss function — many other forms of loss functions exist for different learning problems.
- Expected conditional risk
  
  $$R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)$$

- Expected risk
  
  $$R(f) = \mathbb{E}_{(x, y) \sim p(x, y)} L(f(x), y)$$
More deep understanding about NNC

Measuring performance

Jargons

- $L(f(x), y)$ is called **0/1 loss function** — many other forms of loss functions exist for different learning problems.

- Expected conditional risk
  
  $$R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)$$

- Expected risk
  
  $$R(f) = \mathbb{E}_{(x, y) \sim p(x, y)} L(f(x), y)$$

- Empirical risk
  
  $$R_D(f) = \frac{1}{N} \sum_{n} L(f(x_n), y_n)$$

  Obviously, this is our empirical error (rates).
Expected conditional risk of a single data point $x$

\[
R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y) \\
= P(y = 1 | x) \mathbb{I}[f(x) = 0] + P(y = 0 | x) \mathbb{I}[f(x) = 1]
\]
**Ex: binary classification**

**Expected conditional risk of a single data point** \( x \)

\[
R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y) \\
= P(y = 1|x) \mathbb{I}[f(x) = 0] + P(y = 0|x) \mathbb{I}[f(x) = 1]
\]

Let \( \eta(x) = P(y = 1|x) \), we have

\[
R(f, x) = \eta(x) \mathbb{I}[f(x) = 0] + (1 - \eta(x)) \mathbb{I}[f(x) = 1] \\
= 1 - \left\{ \eta(x) \mathbb{I}[f(x) = 1] + (1 - \eta(x)) \mathbb{I}[f(x) = 0] \right\}
\]

**expected conditional accuracies**
Expected conditional risk of a single data point $x$

$$R(f, x) = \mathbb{E}_{y \sim p(y|x)} L(f(x), y)$$

$$= P(y = 1|x) \mathbb{I}[f(x) = 0] + P(y = 0|x) \mathbb{I}[f(x) = 1]$$

Let $\eta(x) = P(y = 1|x)$, we have

$$R(f, x) = \eta(x) \mathbb{I}[f(x) = 0] + (1 - \eta(x)) \mathbb{I}[f(x) = 1]$$

$$= 1 - \{\eta(x) \mathbb{I}[f(x) = 1] + (1 - \eta(x)) \mathbb{I}[f(x) = 0]\} \text{ expected conditional accuracies}$$

Exercise: please verify the last equality.
Bayes optimal classifier

Consider the following classifier, using the posterior probability

\[ \eta(x) = p(y = 1|x) \]

\[ f^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq 1/2 \\ 0 & \text{if } \eta(x) < 1/2 \end{cases} \]

equivalently

\[ f^*(x) = \begin{cases} 1 & \text{if } p(y = 1|x) \geq p(y = 0|x) \\ 0 & \text{if } p(y = 1|x) < p(y = 0|x) \end{cases} \]
Bayes optimal classifier

Consider the following classifier, using the posterior probability

$$\eta(x) = p(y = 1|x)$$

$$f^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq 1/2 \\ 0 & \text{if } \eta(x) < 1/2 \end{cases}$$

equivalently

$$f^*(x) = \begin{cases} 1 & \text{if } p(y = 1|x) \geq p(y = 0|x) \\ 0 & \text{if } p(y = 1|x) < p(y = 0|x) \end{cases}$$

Theorem

For any labeling function $$f(\cdot)$$, $$R(f^*, x) \leq R(f, x)$$. Similarly, $$R(f^*) \leq R(f)$$. Namely, $$f^*(\cdot)$$ is optimal.
Proof (not required)

From definition

\[ R(f, \mathbf{x}) = 1 - \{\eta(\mathbf{x})\mathbb{I}[f(\mathbf{x}) = 1] + (1 - \eta(\mathbf{x}))\mathbb{I}[f(\mathbf{x}) = 0]\} \]

\[ R(f^*, \mathbf{x}) = 1 - \{\eta(\mathbf{x})\mathbb{I}[f^*(\mathbf{x}) = 1] + (1 - \eta(\mathbf{x}))\mathbb{I}[f^*(\mathbf{x}) = 0]\} \]
Proof (not required)

From definition

\[
R(f, x) = 1 - \{\eta(x)\mathbb{I}[f(x) = 1] + (1 - \eta(x))\mathbb{I}[f(x) = 0]\}
\]

\[
R(f^*, x) = 1 - \{\eta(x)\mathbb{I}[f^*(x) = 1] + (1 - \eta(x))\mathbb{I}[f^*(x) = 0]\}
\]

Thus,

\[
R(f, x) - R(f^*, x) = \eta(x) \{\mathbb{I}[f^*(x) = 1] - \mathbb{I}[f(x) = 1]\}
\]

\[
+ (1 - \eta(x)) \{\mathbb{I}[f^*(x) = 0] - \mathbb{I}[f(x) = 0]\}
\]

\[
= \eta(x) \{\mathbb{I}[f^*(x) = 1] - \mathbb{I}[f(x) = 1]\}
\]

\[
+ (1 - \eta(x)) \{1 - \mathbb{I}[f^*(x) = 1] - 1 + \mathbb{I}[f(x) = 1]\}
\]

\[
= (2\eta(x) - 1) \{\mathbb{I}[f^*(x) = 1] - \mathbb{I}[f(x) = 1]\}
\]

\[
\geq 0
\]
Bayes optimal classifier in general form

For multi-class classification problem

\[ f^*(x) = \arg \max_{c \in [C]} p(y = c | x) \]

when \( C = 2 \), this reduces to detecting whether or not
\[ \eta(x) = p(y = 1 | x) \]

is greater than \( 1/2 \). We refer \( p(y = c | x) \) as the posterior probability of \( x \).
Bayes optimal classifier in general form

For multi-class classification problem

$$f^*(x) = \arg \max_{c \in [C]} p(y = c | x)$$

when $C = 2$, this reduces to detecting whether or not $\eta(x) = p(y = 1 | x)$ is greater than $1/2$. We refer $p(y = c | x)$ as the posterior probability of $x$.

Remarks

- The Bayes optimal classifier is generally not computable as it assumes the knowledge of $p(x, y)$ or $p(y | x)$.

- However, it is useful as a conceptual tool to formalize how well a classifier can do \textit{without} knowing the joint distribution.
More deep understanding about NNC

Comparing NNC to the ideal classifier

Comparing NNC to Bayes optimal classifier

How well does our NNC do?

Theorem

For the NNC rule $f_{\text{NCC}}$ for binary classification, we have,

$$R(f^*) \leq R(f_{\text{NCC}}) \leq 2R(f^*)(1 - R(f^*)) \leq 2R(f^*)$$

Namely, the expected risk by the classifier is at worst twice that of the Bayes optimal classifier.

In short, NNC seems doing a reasonable thing
Proof sketches (not required)

**Step 1** To show that when $N \to +\infty$, $x$ and its nearest neighbor $x(1)$ are similar. In particular $p(y = 1|x)$ and $p(y = 1|x(1))$ are the same.
Proof sketches (not required)

Step 1 To show that when $N \to +\infty$, $x$ and its nearest neighbor $x(1)$ are similar. In particular $p(y = 1|x)$ and $p(y = 1|x(1))$ are the same.

Step 2 To show the mistake made by our classifier on $x$ is attributed to the probability that $x$ and $x(1)$ have different labels.
Proof sketches (not required)

**Step 1** To show that when $N \rightarrow +\infty$, $x$ and its nearest neighbor $x(1)$ are similar. In particular $p(y = 1|x)$ and $p(y = 1|x(1))$ are the same.

**Step 2** To show the mistake made by our classifier on $x$ is attributed to the probability that $x$ and $x(1)$ have different labels.

**Step 3** To show the probability having different labels is

$$R(f, x) = 2p(y = 1|x)[1 - p(y = 1|x)] = 2\eta(x)(1 - \eta(x))$$
**Proof sketches (not required)**

**Step 1** To show that when $N \to +\infty$, $\mathbf{x}$ and its nearest neighbor $\mathbf{x}(1)$ are similar. In particular $p(y = 1|\mathbf{x})$ and $p(y = 1|\mathbf{x}(1))$ are the same.

**Step 2** To show the mistake made by our classifier on $\mathbf{x}$ is attributed to the probability that $\mathbf{x}$ and $\mathbf{x}(1)$ have different labels.

**Step 3** To show the probability having different labels is

$$R(f, \mathbf{x}) = 2p(y = 1|\mathbf{x})[1 - p(y = 1|\mathbf{x})] = 2\eta(\mathbf{x})(1 - \eta(\mathbf{x}))$$

**Step 4** To show the expected risk of the Bayes optimal classifier on $\mathbf{x}$ is

$$R(f^*, \mathbf{x}) = \min\{\eta(\mathbf{x}), 1 - \eta(\mathbf{x})\}$$
Proof sketches (not required)

**Step 1** To show that when $N \rightarrow +\infty$, $x$ and its nearest neighbor $x(1)$ are similar. In particular $p(y = 1|x)$ and $p(y = 1|x(1))$ are the same.

**Step 2** To show the mistake made by our classifier on $x$ is attributed to the probability that $x$ and $x(1)$ have different labels.

**Step 3** To show the probability having different labels is

$$R(f, x) = 2p(y = 1|x)[1 - p(y = 1|x)] = 2\eta(x)(1 - \eta(x))$$

**Step 4** To show the expected risk of the Bayes optimal classifier on $x$ is

$$R(f^*, x) = \min\{\eta(x), 1 - \eta(x)\}$$

**Step 5** Then tie everything together

$$R(f, x) = 2R(f^*, x)(1 - R(f^*, x))$$

We are also most there. But one more step is needed (and omitted here).
Mini-summary

Advantages of NNC

- Computationally, simple and easy to implement – just computing the distance
- Theoretically, has strong guarantees “doing the right thing”

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(ND)$ for labeling a data point
- We need to “carry” the training data around. Without it, we cannot do classification. This type of method is called nonparametric.
- Choosing the right distance measure and $K$ can be involved.
Outline

1 Administration

2 First learning algorithm: Nearest neighbor classifier

3 More deep understanding about NNC

4 Some practical sides of NNC
   • How to tune to get the best out of it?
   • Preprocessing data

5 What we have learned
Two practical issues about NNC

- Choosing $K$, i.e., the number of nearest neighbors (default is 1)
- Choosing the right distance measure (default is Euclidean distance), for example, from the following generalized distance measure

$$
\| \mathbf{x} - \mathbf{x}_n \|_p = \left( \sum_d |x_d - x_{nd}|^p \right)^{1/p}
$$

for $p \geq 1$.

Those are not specified by the algorithm itself — resolving them requires empirical studies and are task/dataset-specific.
Tuning by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
- They are used for learning $f(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x \notin \mathcal{D}^{\text{TRAIN}}$

Development (or validation) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_L, y_L)\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should not overlap!
Recipe

- for each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$
Some practical sides of NNC

How to tune to get the best out of it?

Cross-validation

What if we do not have validation data?

- We split the training data into $S$ equal parts.
- We use each part \textit{in turn} as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that \textit{on average}, the model performing the best

$S = 5$: 5-fold cross validation

\begin{align*}
\text{run 1} & \\
\text{run 2} & \\
\text{run 3} & \\
\text{run 4} & \\
\text{run 5} & \\
\end{align*}

\textit{Special case:} when $S = N$, this will be leave-one-out.
Recipe

- Split the training data into $S$ equal parts. Denote each part as $\mathcal{D}_{s}^{\text{TRAIN}}$
- for each possible value of the hyperparameter (say $K = 1, 3, \ldots, 100$)
  - for every $s \in [1, S]$
    - Train a model using $\mathcal{D}_{s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_{s}^{\text{TRAIN}}$
    - Evaluate the performance of the model on $\mathcal{D}_{s}^{\text{TRAIN}}$
  - Average the $S$ performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$
Yet, another practical issue with NNC

Distances depend on units of the features!
(Show how proximity can be changed due to change in features’ scale; Draw on screen or blackboard)
Preprocess data

Normalize data so that the data look like from a normal distribution

- Compute the means and standard deviations in each feature

\[
\bar{x}_d = \frac{1}{N} \sum_n x_{nd}, \quad s_d^2 = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2
\]

- Scale the feature accordingly

\[
x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}
\]

Many other ways of normalizing data — you would need/want to try different ones and pick them using (cross)validation
Outline

1. Administration
2. First learning algorithm: Nearest neighbor classifier
3. More deep understanding about NNC
4. Some practical sides of NNC
5. What we have learned
Summary so far

- Described a simple learning algorithm
  - Used intensively in practical applications — you will get a taste of it in your homework
  - Discussed a few practical aspects, such as tuning hyperparameters, with (cross)validation
- Briefly studied its theoretical properties
  - Concepts: loss function, risks, Bayes optimal
  - Theoretical guarantees: explaining why NNC would work