CSCI567 Machine Learning (Fall 2014)

Drs. Sha & Liu

{feisha,yanliu.cs}@usc.edu

September 11, 2014
Outline

1. Administration
2. Review of last lecture
3. Some practical sides of NNC
4. Decision tree
D-clearance

- All but a few cases are finalized
- Will no longer accept requests from this point
Outline

1. Administration

2. Review of last lecture
   - General setup for classification
   - Nearest neighbor classifier
   - Understanding learning algorithm

3. Some practical sides of NNC

4. Decision tree
Multi-class classification

**Classify data into one of the multiple categories**
- Input (feature vectors): $\mathbf{x} \in \mathbb{R}^D$
- Output (label): $y \in [C] = \{1, 2, \cdots, C\}$
- 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: $D^{\text{TRAIN}} \cap D^{\text{TEST}} = \emptyset$
Algorithm

Nearest neighbor

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

where \( nn(x) \in [N] = \{1, 2, \cdots, N\} \),
\[ nn(x) = \arg \min_{n \in [N]} \| x - x_n \|_2^2 \]

Classification rule

\[ y = f(x) = y_{nn}(x) \]
Extend to K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use

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

Classification rule

- Every neighbor votes 1 for its class
- Aggregate everyone’s vote

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

- Label with the majority

$$y = f(\mathbf{x}) = \arg \max_{c \in [C]} v_c$$

Other forms of classification rules are also possible.
Choosing optimal $K$ can be difficult

Decision boundaries vary when $K$ is changed

When $K$ increases, the decision boundary becomes smooth.
Measuring whether an algorithm is good or bad

**Performance metric**

- **accuracy**: the percentage of data points being correctly classified
- **error rate**: the percentage of data points being incorrectly classified.

**Empirical quantities**

- Computing the performance metrics on training and evaluation datasets
- The metric on the evaluation dataset reflects more faithfully the algorithm’s performance when applied to *real* data
- Easy to compute
Theoretical measures

Setup

- Assume our data \((x, y)\) is drawn from the joint and *unknown* distribution \(p(x, y)\).
- A *0/1 loss function* to measure 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 risk:

\[
R(f) = \mathbb{E}_{(x, y) \sim p(x, y)} L(f(x), y)
\]
Bayes optimal classifier

For binary classification, consider the following classifier, using the posterior probability $\eta(x) = p(y = 1|x)$

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

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 $x$. 
Comparing NNC to Bayes optimal classifier

How well does our NNC do?

Theorem

For the NNC rule $f^{\text{NNC}}$ for binary classification, we have,

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

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
Things you need to know

**NNC**

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

- **Disadvantages**
  - 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.

**Crucial theoretical concepts** loss function, expected risk, empirical risk, Bayes optimal
Outline

1. Administration

2. Review of last lecture

3. Some practical sides of NNC
   - How to tune to get the best out of it?
   - Preprocessing data

4. Decision tree
Hypeparameters in NNC

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.*
Some practical sides of NNC

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 $K = 1, 3, \ldots, 100$)
  - 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{itemize}
  \item run 1
  \item run 2
  \item run 3
  \item run 4
  \item run 5
\end{itemize}

\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 $D_{s}^{\text{TRAIN}}$ for each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  
  - for every $s \in [1, S]$
    - Train a model using $D_{s}^{\text{TRAIN}} = D^{\text{TRAIN}} - D_{s}^{\text{TRAIN}}$
    - Evaluate the performance of the model on $D_{s}^{\text{TRAIN}}$
  
  - 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 $D^{\text{TRAIN}}$

- Evaluate the model on $D^{\text{TEST}}$
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. Review of last lecture
3. Some practical sides of NNC
4. Decision tree
   - Examples
   - Algorithm
Many decisions are tree structures

**Medical treatment**

- Fever
  - $T > 100$
  - $T < 100$
    - Treatment #1
    - Muscle Pain
      - High
      - Low
        - Treatment #2
        - Treatment #3

**Salary in a company**

- Degree
  - High School
  - College
  - Graduate
    - Work Experience
      - $< \text{yr}$
      - $> \text{yr}$
        - $\text{Salary}_1$
        - $\text{Salary}_2$
      - $> \text{yr}$
        - $\text{Salary}_3$
        - $\text{Salary}_4$
      - $< \text{yr}$
        - $\text{Salary}_5$
        - $\text{Salary}_6$
What is a Tree?

Node

Edge
Special Names for Nodes in a Tree

- **Node**
- **Root**
- **Edge**
- **Leaf**

Drs. Sha & Liu (feisha,yanliu.cs@usc.edu)
A tree partitions the feature space
Learning a tree model

Three things to learn:
1. The structure of the tree.
2. The threshold values ($\theta_i$).
3. The values for the leaves ($A, B, \ldots$).
A tree model for deciding where to eat

**Choosing a restaurant**
(Example from Russell & Norvig, AIMA)

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alt</td>
<td>Bar</td>
</tr>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Classification of examples is positive (T) or negative (F)
First decision: at the root of the tree

**Which attribute to split?**

Patrons? is a better choice—gives information about the classification

Idea: use information gain to choose which attribute to split
How to measure information gain?

Idea:

Gaining information reduces uncertainty

Use to entropy to measure uncertainty

If a random variable $X$ has $K$ different values, $a_1$, $a_2$, ...$a_K$, it is entropy is given by

$$H[X] = - \sum_{k=1}^{K} P(X = a_k) \log P(X = a_k)$$

the base can be 2, though it is not essential (if the base is 2, the unit of the entropy is called “bit”)
Examples of computing entropy

**Entropy**

\[ H(X) = 0.8360 \]

\[ H(X) = 1.3863 \]

\[ H(X) = 0 \]
Which attribute to split?

Patron vs. Type?

By choosing Patron, we end up with a partition (3 branches) with smaller entropy, ie, smaller uncertainty (0.45 bit)

By choosing Type, we end up with uncertainty of 1 bit.

Thus, we choose Patron over Type.

Patrons? is a better choice—gives information about the classification
Uncertainty if we go with “Patron”

For “None” branch
\[ - \left( \frac{0}{0+2} \log \frac{0}{0+2} + \frac{2}{0+2} \log \frac{2}{0+2} \right) = 0 \]

For “Some” branch
\[ - \left( \frac{4}{4+0} \log \frac{4}{4+0} + \frac{4}{4+0} \log \frac{4}{4+0} \right) = 0 \]

For “Full” branch
\[ - \left( \frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4} \right) \approx 0.9 \]

For choosing “Patrons”

weighted average of each branch: this quantity is called conditional entropy

\[ \frac{2}{12} \times 0 + \frac{4}{12} \times 0 + \frac{6}{12} \times 0.9 = 0.45 \]
**Conditional entropy**

**Definition.** Given two random variables $X$ and $Y$

$$H[Y|X] = \sum_k P(X = a_k) H[Y|X = a_k]$$

**In our example**

- $X$: the attribute to be split
- $Y$: Wait or not

**Relation to information gain**

$$\text{GAIN} = H[Y] - H[Y|X]$$

When $H[Y]$ is fixed, we need only to compare conditional entropy.
Conditional entropy for Type

For “French” branch
\[-\left( \frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1} \right) = 1\]

For “Italian” branch
\[-\left( \frac{1}{1+1} \log \frac{1}{1+1} + \frac{1}{1+1} \log \frac{1}{1+1} \right) = 1\]

For “Thai” and “Burger” branches
\[-\left( \frac{2}{2+2} \log \frac{2}{2+2} + \frac{2}{2+2} \log \frac{2}{2+2} \right) = 1\]

For choosing “Type”

weighted average of each branch:
\[
\frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1
\]
next split?

We will look only at the 6 instances with Patrons == Full

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$ $$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$ $$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$ $$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$ $$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$ $$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
</tr>
</tbody>
</table>

Classification of examples is positive (T) or negative (F)
Do we split on “Non” or “Some”?

No, we do not

The decision is deterministic, as seen from the training data
Greedily we build the tree and get this
How deep should we continue to split?

We should be very careful about this

Eventually, we can get all training examples right. But is that what we want?

The maximum depth of the tree is a hyperparameter and should not be tuned by training data — this is to prevent overfitting (we will discuss later)
Control the size of the tree

**We would prune to have a smaller one**

If we stop here, not all training sample would be classified correctly.

More importantly, how do we classify a new instance?

We label the leaves of this smaller tree with the majority of training samples’ labels.
Example

We stop after the root (first node)
Splitting and Stopping Criteria

For every leaf $m$, define the node impurity $Q(m)$ as:

| Misclassification error | $\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk}$. |
| Gini Index | $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk} (1 - \hat{p}_{mk})$. |
| Cross-entropy | $- \sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$. |

The **Misclassification Error** is less sensitive to changes in class probability:

⇒ Use **Gini Index** or **Cross-entropy** for growing $T_0$,
⇒ Use **Misclassification Error** for pruning $T_0$ and finding $T$. 
Summary of learning trees

Other ideas in learning trees
- There are other ways of splitting attributes, such as Gini index.
- There are other fast ways of learning tree models.
- There are approaches of learning an ensemble of tree models (more on this later)

Advantages of using trees
- The models are transparent: easily interpretable by human (as long as the tree is not too big)
- It is parametric thus compact: unlike NNC, we do not have to carry our training instances around