Decision tree, kNN and model selection

Yifeng Tao
School of Computer Science
Carnegie Mellon University

Slides adapted from Ziv Bar-Joseph, Tom Mitchell, Eric Xing
Outline

- Decision tree and random forest
- kNN
- Model selection and overfitting
Decision trees

- One of the most intuitive classifiers
- Easy to understand and construct
- Surprisingly, also works very well
The ‘best’ classifier

Gradient boosted machines and deep neural nets have dominated recent Kaggle competitions

<table>
<thead>
<tr>
<th>Competition</th>
<th>Type</th>
<th>Winning ML Library/Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liberty Mutual</td>
<td>Regression</td>
<td>XGBoost</td>
</tr>
<tr>
<td>Caterpillar Tubes</td>
<td>Regression</td>
<td>Keras + XGBoost + Reg. Forest</td>
</tr>
<tr>
<td>Diabetic Retinopathy</td>
<td>Image</td>
<td>SparseConvNet + RF</td>
</tr>
<tr>
<td>Avito</td>
<td>CTR</td>
<td>XGBoost</td>
</tr>
<tr>
<td>Taxi Trajectory 2</td>
<td>Geostats</td>
<td>Classic neural net</td>
</tr>
<tr>
<td>Grasp and Lift</td>
<td>EEG</td>
<td>Keras + XGBoost + other CNN</td>
</tr>
<tr>
<td>Otto Group</td>
<td>Classification</td>
<td>Stacked ensemble of 35 models</td>
</tr>
<tr>
<td>Facebook IV</td>
<td>Classification</td>
<td>sklearn GBM</td>
</tr>
</tbody>
</table>

- They are quite robust, intuitive and, surprisingly, very accurate
- XGBoost: an ensemble method of decision trees

[Slide from https://www.quora.com/What-machine-learning-approaches-have-won-most-Kaggle-competitions]
Structure of a decision tree

- An advertising example:
  - Internal nodes correspond to attributes (features)
  - Leaf nodes correspond to classification outcomes
  - Edges denote assignment

[Diagram of a decision tree with examples of attribute assignments like age > 26, income > 40K, citizen, and female.]
## Dataset

<table>
<thead>
<tr>
<th>Movie</th>
<th>Type</th>
<th>Length</th>
<th>Director</th>
<th>Famous actors</th>
<th>Liked?</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>Comedy</td>
<td>Short</td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m2</td>
<td>Animated</td>
<td>Short</td>
<td>Lasseter</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>m3</td>
<td>Drama</td>
<td>Medium</td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m4</td>
<td>animated</td>
<td>long</td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m5</td>
<td>Comedy</td>
<td>Long</td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m6</td>
<td>Drama</td>
<td>Medium</td>
<td>Singer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>m7</td>
<td>animated</td>
<td>Short</td>
<td>Singer</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m8</td>
<td>Comedy</td>
<td>Long</td>
<td>Adamson</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>m9</td>
<td>Drama</td>
<td>Medium</td>
<td>Lasseter</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Building a decision tree

Function BuildTree(n,A)  // n: samples (rows), A: attributes
    If empty(A) or all n(L) are the same
        status = leaf
        class = most common class in n(L)
    else
        status = internal
        a \= bestAttribute(n,A)
        LeftNode = BuildTree(n(a=1), A \ {a})
        RightNode = BuildTree(n(a=0), A \ {a})
    end
end

n(L): Labels for samples in this set
We will discuss this function next
Recursive calls to create left and right subtrees, n(a=1) is the set of samples in n for which the attribute a is 1

[Slide from Ziv Bar-Joseph]
Identifying ‘bestAttribute’

- There are many possible ways to select the best attribute for a given set.
- We will discuss one possible way which is based on information theory and generalizes well to non binary variables.
Entropy

- Quantifies the amount of uncertainty associated with a specific probability distribution
- The higher the entropy, the less confident we are in the outcome
- Definition

\[ H(X) = \sum_c - p(X = c) \log_2 p(X = c) \]

Claude Shannon (1916 – 2001), most of the work was done in Bell labs
Entropy

○ Definition

\[ H(X) = \sum_i - p(X = i) \log_2 p(X = i) \]

○ So, if \( P(X=1) = 1 \) then

\[ H(X) = -p(x = 1) \log_2 p(X = 1) - p(x = 0) \log_2 p(X = 0) \]

\[ = -1 \log 1 - 0 \log 0 = 0 \]

○ If \( P(X=1) = .5 \) then

\[ H(X) = -p(x = 1) \log_2 p(X = 1) - p(x = 0) \log_2 p(X = 0) \]

\[ = -.5 \log_2 .5 - .5 \log_2 .5 = - \log_2 .5 = 1 \]
Interpreting entropy

- Entropy can be interpreted from an information standpoint.
- Assume both sender and receiver know the distribution. How many bits, on average, would it take to transmit one value?
  - If \( P(X=1) = 1 \) then the answer is 0 (we don’t need to transmit anything).
  - If \( P(X=1) = 0.5 \) then the answer is 1 (either values is equally likely).
  - If \( 0 < P(X=1) < 0.5 \) or \( 0.5 < P(X=1) < 1 \) then the answer is between 0 and 1.
Conditional entropy

- Entropy measures the uncertainty in a specific distribution.
- What if we know something about the transmission?
- For example, say I want to send the label (liked) when the length is known.
- This becomes a conditional entropy problem: $H(\text{Li} \mid \text{Le}=v)$ is the entropy of liked among movies with length $v$.

<table>
<thead>
<tr>
<th>Movie length</th>
<th>Liked?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Yes</td>
</tr>
<tr>
<td>Short</td>
<td>No</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>Long</td>
<td>No</td>
</tr>
<tr>
<td>Long</td>
<td>No</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>Short</td>
<td>Yes</td>
</tr>
<tr>
<td>Long</td>
<td>Yes</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
</tbody>
</table>

[Slide from Ziv Bar-Joseph]
Conditional entropy: Examples for specific values

- Let's compute $H(L_i | L_e = v)$
  - 1. $H(L_i | L_e = S) = .92$
  - 2. $H(L_i | L_e = M) = 0$
  - 3. $H(L_i | L_e = L) = .92$

<table>
<thead>
<tr>
<th>Movie length</th>
<th>Liked?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Yes</td>
</tr>
<tr>
<td>Short</td>
<td>No</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>Long</td>
<td>No</td>
</tr>
<tr>
<td>Long</td>
<td>No</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>Short</td>
<td>Yes</td>
</tr>
<tr>
<td>Long</td>
<td>Yes</td>
</tr>
<tr>
<td>Medium</td>
<td>Yes</td>
</tr>
</tbody>
</table>
We can generalize the conditional entropy idea to determine $H( L_i | L_e)$

Definition: $H(Y | X) = \sum_i P(X = i)H(Y | X = i)$

We explained how to compute this in the previous slides.
Conditional entropy: Example

\[ H(Y \mid X) = \sum_i P(X = i)H(Y \mid X = i) \]

- Let's compute \( H(\text{Li} \mid \text{Le}) \)
  
  \[ H(\text{Li} \mid \text{Le}) = P(\text{Le} = \text{S}) \cdot H(\text{Li} \mid \text{Le} = \text{S}) + \]
  
  \[ P(\text{Le} = \text{M}) \cdot H(\text{Li} \mid \text{Le} = \text{M}) + \]
  
  \[ P(\text{Le} = \text{L}) \cdot H(\text{Li} \mid \text{Le} = \text{L}) = \]
  
  \[ 1/3 \cdot .92 + 1/3 \cdot 0 + 1/3 \cdot .92 = \]
  
  0.61

---

we already computed:

\[ H(\text{Li} \mid \text{Le} = \text{S}) = .92 \]
\[ H(\text{Li} \mid \text{Le} = \text{M}) = 0 \]
\[ H(\text{Li} \mid \text{Le} = \text{L}) = .92 \]
Information gain

○ How much do we gain (in terms of reduction in entropy) from knowing one of the attributes

○ In other words, what is the reduction in entropy from this knowledge

○ Definition: $IG(Y|X) = H(Y) - H(Y|X)$
  ○ $IG(X|Y)$ is always $\geq 0$ Proof: Jensen inequality
Where we are

- We were looking for a good criteria for selecting the best attribute for a node split
- We defined the entropy, conditional entropy and information gain
- We will now use information gain as our criteria for a good split
- That is, bestAttribute will return the attribute that maximizes the information gain at each node
Function BuildTree(n,A)  // n: samples (rows), A: attributes
    If empty(A) or all n(L) are the same
        status = leaf
        class = most common class in n(L)
    else
        status = internal
        a ← bestAttribute(n,A)
        LeftNode = BuildTree(n(a=1), A \ {a})
        RightNode = BuildTree(n(a=0), A \ {a})
    end
end

Based on information gain
Example: Root attribute

- $P(L_i=\text{yes}) = \frac{2}{3}$
- $H(L_i) = .91$

- $H(L_i \mid T) = 0.61$
- $H(L_i \mid \text{Le}) = 0.61$
- $H(L_i \mid D) = 0.36$
- $H(L_i \mid F) = 0.85$

- $IG(L_i \mid T) = .91-.61 = 0.3$
- $IG(L_i \mid \text{Le}) = .91-.61 = 0.3$
- $IG(L_i \mid D) = .91-.36 = 0.55$
- $IG(L_i \mid \text{Le}) = .91-.85 = 0.06$

<table>
<thead>
<tr>
<th>Movie</th>
<th>Type</th>
<th>Length</th>
<th>Director</th>
<th>Famous actors</th>
<th>Liked</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>Comedy</td>
<td>Short</td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m2</td>
<td>Animated</td>
<td>Short</td>
<td>Lasseter</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>m3</td>
<td>Drama</td>
<td>Medium</td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m4</td>
<td>animated</td>
<td>Long</td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m5</td>
<td>Comedy</td>
<td>Long</td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m6</td>
<td>Drama</td>
<td>Medium</td>
<td>Singer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>M7</td>
<td>animated</td>
<td>Short</td>
<td>Singer</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m8</td>
<td>Comedy</td>
<td>Long</td>
<td>Adamson</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>m9</td>
<td>Drama</td>
<td>Medium</td>
<td>Lasseter</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Building a tree

<table>
<thead>
<tr>
<th></th>
<th>Movie</th>
<th>Type</th>
<th>Length</th>
<th>Director</th>
<th>Famous actors</th>
<th>Liked?</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>Comedy</td>
<td>Short</td>
<td></td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m2</td>
<td>Animated</td>
<td>Short</td>
<td></td>
<td>Lasseter</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>m3</td>
<td>Drama</td>
<td>Medium</td>
<td></td>
<td>Adamson</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m4</td>
<td>animated</td>
<td>long</td>
<td></td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m5</td>
<td>Comedy</td>
<td>Long</td>
<td></td>
<td>Lasseter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m6</td>
<td>Drama</td>
<td>Medium</td>
<td></td>
<td>Singer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>M7</td>
<td>animated</td>
<td>Short</td>
<td></td>
<td>Singer</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>m8</td>
<td>Comedy</td>
<td>Long</td>
<td></td>
<td>Adamson</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>m9</td>
<td>Drama</td>
<td>Medium</td>
<td></td>
<td>Lasseter</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

[Slide from Ziv Bar-Joseph]
Building a tree

- We only need to focus on the records (samples) associated with this node.
- We eliminated the ‘director’ attribute. All samples have the same director.
- $P(Li=\text{yes}) = \frac{1}{4}$, $H(Li) = .81$
- $H(Li \mid T) = 0$, $H(Li \mid Le) = 0$, $H(Li \mid F) = 0.5$
- $IG(Li \mid T) = 0.81$, $IG(Li \mid Le) = 0.81$, $IG(Li \mid F) = .31$

[Slide from Ziv Bar-Joseph]
Final tree

- **D**
  - Adamson
  - Lasseter
    - Singer
      - yes
      - no
    - animated
      - drama
      - comedy
      - yes
      - no

<table>
<thead>
<tr>
<th>Movie</th>
<th>Type</th>
<th>Length</th>
<th>Famous actors</th>
<th>Liked ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2</td>
<td>Animated</td>
<td>Short</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>m4</td>
<td>animated</td>
<td>long</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m5</td>
<td>Comedy</td>
<td>Long</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>m9</td>
<td>Drama</td>
<td>Medium</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Continuous values

- Either use threshold to turn into binary or discretize
- It’s possible to compute information gain for all possible thresholds (there are a finite number of training samples)
- Harder if we wish to assign more than two values (can be done recursively)
Additional points

- The algorithm we gave reaches homogenous nodes (or runs out of attributes)
- This is dangerous: For datasets with many (non relevant) attributes the algorithm will continue to split nodes
- This will lead to overfitting!

[Slide from Ziv Bar-Joseph]
Overfitting

- Training/empirical error: \[ \frac{1}{n} \sum_{i=1}^{n} I(f(X_i) \neq Y_i) \]

- Error over whole data distribution: \[ R(f) = P(f(X) \neq Y) \]
Avoiding overfitting: Tree pruning

- One possible way in decision tree to avoid overfitting
  - Other ways: Do not grow tree beyond some maximum depth; Do not split if splitting criterion (e.g. mutual information) is below some threshold; Stop growing when the split is not statistically significant

- Split data into train and testing set
- Build tree using training set
- For all internal nodes (starting at the root)
  - remove subtree rooted at node
  - assign class to be the most common among training set
  - check test data error
    - if error is lower, keep change
    - otherwise restore subtree, repeat for all nodes in subtree
Random forest: bootstrap

- A collection of decision trees: ensemble methods
- For each tree we select a subset of the attributes (recommended square root of $|A|$), a subset of samples, and build tree using just these attributes
- An input sample is classified using majority voting
Each document is a vector, one component for each term (= word).

| Word 1 | Word 2 | Word 3 | ...
|--------|--------|--------|--------|
| Doc 1  | 3      | 0      | 0      | ...
| Doc 2  | 0      | 8      | 1      | ...
| Doc 3  | 12     | 1      | 10     | ...
| ...
| ...    | 0      | 0      | 0      | ...
| ...
| ...    | 0      | 1      | 3      | ...

- Normalize to unit length.
- High-dimensional vector space:
  - Terms are axes, 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space
Classes in a Vector Space

[Slide from Eric Xing]
Test Document = ?

[Slide from Eric Xing]
K-Nearest Neighbor (kNN) classifier

[Slide from Eric Xing]
kNN is an instance of Instance-Based Learning

- What makes an Instance-Based Learner?
  - A distance metric
  - How many nearby neighbors to look at?
  - A weighting function (optional)
Euclidean Distance Metric

\[ D(x, x') = \sqrt{\sum_i \sigma_i^2 (x_i - x_i')^2} \]

Or equivalently,

\[ D(x, x') = \sqrt{(x - x')^T \Sigma (x - x')} \]

Other metrics:
- L₁ norm: \(|x-x'|\)
- Lₘ norm: \(\max |x-x'|\) (elementwise …)
- Mahalanobis: where \(\Sigma\) is full, and symmetric
- Correlation
- Angle
- Hamming distance, Manhattan distance
- …
1-Nearest Neighbor (kNN) classifier

[Slide from Eric Xing]
5-Nearest Neighbor (kNN) classifier

(Slide from Eric Xing)
Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in \( D \).
- Testing instance \( x \):
  - Compute similarity between \( x \) and all examples in \( D \).
  - Assign \( x \) the category of the most similar example in \( D \).
- Does not explicitly compute a generalization or category prototypes.
- Also called:
  - Case-based learning
  - Memory-based learning
  - Lazy learning
kNN Is Close to Optimal

- Cover and Hart 1967
- Asymptotically, the error rate of 1-nearest-neighbor classification is less than twice the **Bayes rate** [error rate of classifier knowing model that generated data]
- In particular, asymptotic error rate is 0 if Bayes rate is 0.
- Decision boundary:

[Slide from Eric Xing]
Another example

[Figure from Christopher Bishop]
Now let's look more closely into two sources of errors in an functional approximator:

In the following we show the Bias-variance decomposition using LR as an example.
Loss functions for regression

Let \( t \) be the true (target) output and \( y(x) \) be our estimate. The expected squared loss is

\[
E(L) = \iint L(t, y(x)) p(x, t) dx dt
= \iint (t - y(x))^2 p(x, t) dx dt
\]

Our goal is to choose \( y(x) \) that minimize \( E(L) \):

- Calculus of variations:

\[
\frac{\partial E(L)}{\partial y(x)} = 2 \int (t - y(x)) p(x, t) dt = 0
\]

\[
\int y(x) p(x, t) dt = \int tp(x, t) dt
\]

\[
y^*(x) = \int \frac{tp(x, t)}{p(x)} dt = \int tp(t \mid x) dt = E_{y\mid x}[t] = E[t \mid x]
\]
Expected loss

Let $h(x) = E[t|x]$ be the **optimal** predictor, and $y(x)$ our actual predictor, which will incur the following expected loss

$$E(y(x) - t)^2 = \int (y(x) - h(x) + h(x) - t)^2 p(x,t)dxdt$$

$$= \int (y(x) - h(x))^2 + 2(y(x) - h(x))(h(x) - t) + (h(x) - t)^2 p(x,t)dxdt$$

$$= \int (y(x) - h(x))^2 p(x)dx + \int (h(x) - t)^2 p(x,t)dxdt$$

- $\int (h(x) - t)^2 p(x,t)dxdt$ is a noisy term, and we can do no better than this. Thus it is a lower bound of the expected loss.

- The other part of the error come from $\int (y(x) - h(x))^2 p(x)dx$, and let's take a close look of it.

- We will assume $y(x) = y(x|w)$ is a parametric model and the parameters $w$ are fit to a training set $D$. (thus we write $y(x;D)$ )
Bias-variance decomposition

For one data set $D$ and one test point $x$

- since the predictor $y$ depend on the data training data $D$, write $E_D[y(x,D)]$ for the expected predictor over the ensemble of datasets, then (using the same trick) we have:

$$\left(y(x;D) - h(x)\right)^2 = \left(y(x;D) - E_D[y(x;D)] + E_D[y(x;D)] - h(x)\right)^2$$

$$= \left(y(x;D) - E_D[y(x;D)]\right)^2 + \left(E_D[y(x;D)] - h(x)\right)^2$$

$$+ 2\left(y(x;D) - E_D[y(x;D)]\right)\left(E_D[y(x;D)] - h(x)\right)$$

- Surely this error term depends on the training data, so we take an expectation over them:

$$E_D\left[\left(y(x;D) - h(x)\right)^2\right] = \left(E_D[y(x;D)] - h(x)\right)^2 + E_D\left[\left(y(x;D) - E_D[y(x;D)]\right)^2\right]$$

Putting things together:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$
Bias-variance tradeoff: Regularized Regression

- $\lambda$ is a "regularization" terms in LR, the smaller the $\lambda$, is more complex the model (why?)
  - Simple (highly regularized) models have low variance but high bias.
  - Complex models have low bias but high variance.

- You are inspecting an empirical average over 100 training set.

- The actual $E_D$ can not be computed

[Slide from Eric Xing]
Bias$^2 + $variance vs regularizer

- Bias$^2 + $variance predicts (shape of) test error quite well.
- However, bias and variance cannot be computed since it relies on knowing the true distribution of $x$ and $t$ (and hence $h(x) = E[t|x]$).
Model Selection

- Suppose we are trying select among several different models for a learning problem.

- Examples:
  - polynomial regression
    - Model selection: we wish to automatically and objectively decide if $M$ should be, say, 0, 1, ..., or 10.
  - Etc…

- The Problem:

  Given model family $\mathcal{F} = \{M_1, M_2, \ldots, M_I\}$, find $M_i \in \mathcal{F}$ s.t.

  $$M_i = \arg \max_{M \in \mathcal{F}} J(D, M)$$
Cross Validation

We are given training data $D$ and test data $D_{\text{test}}$, and we would like to fit this data with a model $p_i(x; \theta)$ from the family $\mathcal{F}$ (e.g., an LR), which is indexed by $i$ and parameterized by $\theta$.

*K-fold cross-validation (CV)*

- Set aside $\alpha N$ samples of $D$ (where $N = |D|$). This is known as the held-out data and will be used to evaluate different values of $i$.
- For each candidate model $i$, fit the optimal hypothesis $p_i(x; \theta^*)$ to the remaining $(1-\alpha)N$ samples in $D$ (i.e., hold $i$ fixed and find the best $\theta$).
- Evaluate each model $p_i(x|\theta^*)$ on the held-out data using some pre-specified risk function.
- Repeat the above $K$ times, choosing a different held-out data set each time, and the scores are averaged for each model $p_i(.)$ over all held-out data set. This gives an estimate of the risk curve of models over different $i$.
- For the model with the lowest risk, say $p_{i^*}(.)$, we use all of $D$ to find the parameter values for $p_{i^*}(x; \theta^*)$.

[Slide from Eric Xing]
Cross Validation

\[ D = y^{(1)} \quad x^{(1)} \]
\[ y^{(2)} \quad x^{(2)} \]
\[ y^{(N)} \quad x^{(N)} \]

Fold 1
Fold 2
Fold 3
Fold 4

(Slide from Matt Gormley)
Example

- When $\alpha=1/N$, the algorithm is known as Leave-One-Out Cross-Validation (LOOCV)

\[
\text{MSELOOCV}(M_1) = 2.12 \\
\text{MSELOOCV}(M_2) = 0.962
\]

[Slide from Eric Xing]
Practical issues for CV

- **How to decide the values for $K$ and $\alpha$**
  - Commonly used $K=10$ and $\alpha=0.1$.
  - When data sets are small relative to the number of models that are being evaluated, we need to decrease $\alpha$ and increase $K$.

- **Bias-variance trade-off**
  - Small $\alpha$ usually lead to low bias. In principle, LOOCV provides an almost unbiased estimate of the generalization ability of a classifier, especially when the number of the available training samples is severely limited.
  - Large $\alpha$ can reduce variance, but will lead to under-use of data, and causing high-bias.

- **One important point is that the test data $D_{\text{test}}$ is never used in CV, because doing so would result in overly (indeed dishonest) optimistic accuracy rates during the testing phase.**
Regularization

- More in lecture 1...

The posterior distribution of $\theta$

$$p(\theta|D) \propto \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 \right\} \times \exp \left\{ -\theta^T \theta / 2\tau^2 \right\}$$

This leads to a new objective

$$l_{MAP}(\theta; D) = -\frac{1}{2\sigma^2} \frac{1}{2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 - \frac{1}{\tau^2} \frac{1}{2} \sum_{k=1}^{K} \theta_k^2$$

$$= l(\theta; D) - \lambda \| \theta \|$$

- This is $L_2$ regularized LR! --- a MAP estimation of $\theta$
- What about $L_1$ regularized LR! (homework)

How to choose $\lambda$.

- cross-validation!
AIC and BIC

- Akaike information criterion (AIC):
  \[ \text{AIC} = 2k - 2 \ln(\hat{L}) \]

- Bayesian information criterion (BIC):
  \[ \text{BIC} = \ln(n)k - 2 \ln(\hat{L}) \]
Take home message

- Decision tree is a discriminative classifier
  - Built recursively based on maximizing information gain
  - Prevent overfitting through pruning and bootstrap

- kNN is close to optimal (asymptotically)

- Tradeoff between bias and variance of models
  \[ \text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise} \]

- Methods to reduce overfitting
  - Cross-validation
  - Regularization
  - AIC, BIC
  - Ensemble methods: bootstrap
References

- Christopher M. Bishop. Pattern recognition and Machine Learning
- Wikipedia