Decision Trees

Oct - 7 - 2009
Previously

• We learned two different classifiers
  – Perceptron: LTU
  – KNN: complex decision boundary

• If you are a novice in this field, given a classification application, are these two methods ready-to-use?
Off-The-Shelf Classifier

A method that can be applied directly to data without requiring a great deal of time-consuming data preprocessing or careful tuning of the learning procedure.
More specifically …

• Robustness to outliers
• Ability to deal with irrelevant inputs
• Handling different feature types:
  – Continuous vs discrete (ordered, unordered)
• Computational scalability for large data sets
• Interpretability
• Predictive power
Robustness to Outliers

- Outliers are those that are *different* from the majority of the examples.
- Outliers can be different in various ways.

Points with label noise

Points that distributed differently from the rest
• Perceptron: Outliers can cause the algorithm to loop forever if it causes the data to be linearly inseparable

• KNN: the impact of outliers are localized, decision boundaries far away from the outliers will not be influenced, thus more robust
Remaining Criteria

- **Computational Scaling**: Perceptron scales better than KNN
- **Irrelevant features**:
  - Perceptron will likely assign small weights to irrelevant features.
  - KNN will be misguided by irrelevant features unless we learn the distance function carefully
- **Different feature types**: Both KNN and perceptron assumes that features are continuous, but can be extended to work with other types of features
- **Interpretability**: both are fairly simple and easy to interpret
- **Predictive power**: KNN and perceptron are both on the low end of predictive power, there are much more powerful classifier that we will see soon
- **Bottom line**: both methods are not so “off-the-shelf”!
Decision Tree
One of the most popular off-the-shelf classifiers
Decision Tree for Playing Tennis

- **Outlook**
  - Sunny
  - Overcast
    - Yes
  - Rain
    - Strong
      - No
    - Weak
      - Yes

- **Humidity**
  - High
  - Normal
    - No
    - Yes
Definition

- **Internal nodes**
  - Each tests an attribute
  - Branch according to attribute values
  - Discrete attributes – branching is naturally defined
  - Continuous attributes – branching by comparing to a threshold

- **Leaf nodes**
  - Each assign a class label
(outlook=sunny, wind=strong, humidity=normal, ? )
Decision Tree Decision Boundaries

- For continuous attributes, a decision tree divides the input space into *axis-parallel rectangles* and label each rectangle with one of the K classes.
Characteristics of Decision Trees

- Decision trees have many appealing properties
  - Similar to human decision process, easy to understand
  - Deal with both discrete and continuous features
  - Highly flexible hypothesis space, as the # of nodes (or depth) of the tree increase, decision tree can represent increasingly complex decision boundaries

**Definition: Hypothesis space $H$**

The space of solutions that a learning algorithm can possibly output. For example,

- For Perceptron: the hypothesis space is the space of all straight lines
- For nearest neighbor: the hypothesis space is infinitely complex
- For decision tree: it is a flexible space, as we increase the depth of the tree, the hypothesis space grows larger and larger
DT can represent arbitrarily complex decision boundaries

If needed, the tree can keep on growing until all examples are correctly classified! Although it may not be the best idea.
So far we have looked at what is a decision tree, and what kind of decision boundaries decision trees produce, and its appealing properties. We now need to address:

How to learn decision trees

• Goal: Find a decision tree $h$ that achieves **minimum** misclassification errors on the training data

• As our previous slides suggest, we can always achieve this by using large trees

• In fact, we can achieve this trivially: just create a decision tree with one path from root to leaf for each training example
  – Problem: Such a tree would just memorize the training data. It would not **generalize to new data points** – *i.e.*, capture regularities that are applicable to unseen data

• Alternatively: find the **smallest** tree $h$ that minimizes training error
  – Problem: This is NP-Hard
Greedy Learning For DT

There are different ways to construct trees from data. We will focus on the top-down, greedy search approach. Instead of trying to optimize the whole tree together, we try to find one test at a time.

Basic idea: (assuming discrete features, relax later)

1. Choose the best attribute $a^*$ to place at the root of the tree.
2. Separate training set $S$ into subsets $\{S_1, S_2, \ldots, S_k\}$ where each subset $S_i$ contains examples having the same value for $a^*$
3. Recursively apply the algorithm on each new subset until all examples have the same class or there are few of them.
4. Label the leaf nodes with the dominating class among all examples in that node
Building DT: an example

Training data contains
13 15

If we had to make a decision now, we’d pick 13. But there’s too much uncertainty.

Based on training data, with probability 13/28 I would be wrong

Now if you are allowed to ask one question about your example to help the decision, which question will you ask?
One possible question: is $x < 0.5$?

Now we feel much better because the uncertainty in each leaf node is much reduced!
Building a decision tree

1. Choosing the best attribute $a^*$ to place at the root of the tree.
   – What do we mean by “best” – reduce the most uncertainty about our decision of the class labels
2. Separate the training set $S$ into subsets $\{S_1, S_2, \ldots, S_k\}$ where each subset $S_i$ contains examples having the same value for $a^*$
3. Recursively apply the algorithm on each new subset until all examples have the same class label
4. Label each leaf node by its dominant class
Choosing split: example

If we were to stop here, what labels to give to the leave nodes?
Left branch: $y=1$, right branch: $y=0$
How many mistakes we will be making? 2

We can use the number of training errors as a way of measuring the uncertainty