Lesson 7 - Machine Learning

Introduction to Machine Learning

kNN

1-Nearest neighbor

k-Nearest neighbor

Cross validation

Cross validation description

Overfitting

Independent test set

LOOOCV

Gaussian distribution

Formula Properties of Gaussian distribution

Central limit theorem

With tons of data - with enough factors and enough students grades approach Gaussian

Decision boundaries

in 2 d conic section in 3 d hyper quadratics

Error

integral of Gaussian curve for misclassified gives error sometimes the impact of false positive, false negative have different impact so weight the error reduce false pos or false negs.

Bayes classifiers

Bayes rule

Naive Bayes

assume features are conditionally independent on class

Maximum likelihood

assume priors are equally likely

No free Lunch

No 1 algo is good for all ml problems

Generalization

Impact decision boundaries - avoid overfitting

Visualization

for low d plotting

for higher d DTs are useful for seeing which features are important

Decision trees with discrete information

DT use simple tree structure to with nodes being attributes, branches being actual feature an attribute can be, and leaves being nodes

Decision trees with continuous information

Additional steps are needed to discretize the continuous data.

Minimum description length

Smaller trees are better

Entropy

Measures how separated the data is

```
from math import log

def log_2(n):
    if n == 0:
        return 0
    else:
        return log(n, 2)

def b_q(q):
    return -(q * log_2(q) + (1 - q) * log_2(1 - q))

def b(p, n):
    return b_q(p/(p+n))
```

Information Gain

information gain would be 1 bit if the data completely split the classifiers

```
def gain(all, p_n):
    total = all[0] + all[1]
    remainder = sum((x[0] + x[1]) / total * b(x[0], x[1]) for x in p_n)
    return b(all[0], all[1]) - remainder

total = (6,6)
full = [(0,2), (4,0), (2,4)]
type = [(1,1), (1,1), (2,2), (2,2)]

print(gain(total, full))
```

Ensemble Learners

Random Forests

Bagging or Bootstrapping

Input: Data set of size N with M dimensions:

1. Sample n time from Data
2. Sample m times from attributes
3. Learn tree on sampled data and attributes
4. Repeat until k trees

When making a decision k trees vote on the answer. Kind of like a mixture of experts. Compared to using a decision tree, this method tends to avoid overfitting.

Boosting

Decision trees can help to determine what features most important. This can optimize performance (for example in a mobile setting where too many sensors may require too much battery power) or be useful in feature selection or focusing which attributes need further investigation.

Boosting is a machine learning technique. It combines many weak classifiers to create a classifier that is better than the weak classifiers by themselves.

```
def alpha(e):
    return 0.5 * log((1 - e) / e)

e_t = [7/20, 8/20, 9/20, 6/20, 8/20, 3/20]
print([alpha(e) for e in e_t])
```

Neural Nets

Neural nets are a type of machine learning modeled off of biological neurons.

A simple neural net unit takes in inputs biased by a weight and uses an activation function to determine its output. This output can be passed along to other neural net units.

Multilayer nets

Combine neurons into larger layers and networks.

Feed forward - no internal states

Recurrent networks - neurons with delays and internal loops

If every node in a net is linear a network can be reduced to a single neuron but this reduces the flexibility of the neural net

Perceptron learning

Rough algorithm for perceptron learning:

1. Use error metric (square error is common)
2. Perform optimization search by gradient descent
3. Update weights according to gradient descent
4. Repeat cycling through different combinations of inputs and outputs

Expressiveness of perceptron

A single perceptron can only describe a linear decision boundary. This can allow it to handle things like and and or but not xor.

Compared to decision trees perceptron can do something like a majority function really well - an equivalent DT would need 11 layers and would need to see close to the full number of outcomes (2048) to make a good decision.

Meanwhile a perceptron performed relatively poorly on the "dinner" data set.

Multilayer perceptron

Neural nets can be improved by using multiple layers of perceptrons. With 2 layers any continuous function can be modeled. With 3 layers disjoint functions with bumps can be modeled.

Back-propagation

Algorithm for back propagation of error:

1. Output layer rule for updating weights is still the same as a single layer perceptron: delta_i = Err_i x g^1(in_i)
2. For the hidden layer, determine the error attributed to the layer by back propagation of the error from the output layer: delta_j = g^1(in_j) \sum_i {W_{j,i} delta_i}
3. Update the weights in the hidden layer based on the back propagated error: W_{k,j} <- W_{k,j} + alpha x alpha_k x delta_j
4. Iterate until the earliest hidden layer is reached

The process of updating the weights and summing the gradient updates for all the training examples is called an epoch. You can estimate how hard a problem is based on how long it takes for all the weights to converge to get a minimum error on the training set.

Deep Learning

Resurgence in using neural nets with multiple layers with what is called deep learning. Deep learning techniques use hierarchical structures to solve complex problems.

Neural nets powerful, but often require lots of training examples and lots of computation to produce good results. But another big reason for not using neural nets is that it's hard for humans to interpret the results.

Progress is being made to produce results that are understandable at different hidden layer levels however ultimately with various complex problems performance is the main goal not necessarily being able to interpret the various layers.

Unsupervised learning

Unlike supervised learning where the labels are given with the feature set, unsupervised takes data without labels and attempts to determine what classes are in the data and what data belongs to which class. This can be useful when we have large databases that are hard to label or when we have data that doesn't necessarily come with a set of labels i.e. analyzing animal vocalizations to determine if there is a pattern.

k-Means

Algorithm for k-Means:

1. Put k "means" (k based on the number of labels we expect) on the plot.
2. For every point it the database we assign it to the nearest mean. This step is called expectation.
3. Recalculate the means based on the assignment of the points to clusters. This steps is called maximization.

Random restart can be useful if the results don't converge well or to get a better sense of the stability of the convergence.

Visualization is helpful in determining the effectiveness of clustering. For good clustering we want to see data with high inter cluster variance bu low intra cluster variance. In other words we want to see well defined blobs in the data.

EM

Algorithm for EM (similar to k-means):

1. We need to know how many Gaussians we want to use.
2. We assign each point to a cluster.
3. Reestimate the means and variance of the Gaussians.

Because there are more parameters to calculate it will typically take longer to converge than k-means