**Machine Learning**

**Session 5**

1. **Decision trees**:
   1. Contingency tables:
      1. For every combination of attributes, record how frequently it occurs
      2. Check the cube to predict new data
         1. Would be slow and accurate
         2. Decision tree can compress the cube
   2. Model Structure & Test time procedure:
      1. Internal nodes:
         1. Test the value of a particular attribute: Equality / Inequality.
         2. Branch according to the result
      2. Leaf nodes:
         1. Specify the class f(x)
      3. Classify x\* by sending down the tree
   3. Recursive algorithm:
      1. Grow(T)
         1. if All y=0, return Leaf(0)
         2. elseif All y=1, return Leaf(1)
         3. else
            1. x\_j = ChooseBestAttribute(T)
            2. T0 = in T with x\_j=0
            3. T1 = in T with x\_j =1
            4. Return Node(x\_j , Grow(T0), Grow(T1))
      2. j = ChooseBestAttribute(T)
         1. Choose j to minimize
         2. Examples in T0 with y!=0
         3. +
         4. Examples in T1 with y!=1
      3. Actually minimize information gain instead
   4. Training with non-boolean features:
      1. Nominal
         1. Test one value versus all the others (Outlook=Sunny)
         2. Group into disjoint subsets. (Postcode = W1)
      2. Continuous
         1. Threshold inequality x\_j > th
   5. Representation of nominal data:
      1. • Depth 1 tree
         1. Any Boolean function of 1 feature..
      2. Depth 2 tree
         1. Any Boolean function of two features..
      3. DT can represent any boolean function
         1. (But worst case 2^N leaves)
   6. Continuous Data: (A state space [basically a xy graph] separated by a linear prediction line between salmon or cod)
      1. If Length > L1
         1. Then Salmon
      2. Else
         1. If Lightness >L2
         2. Then Cod
      3. Else
         1. Then Salmon
      4. Represent:
         1. Axis parallel cuts.
         2. Can approximate but not exactly represent diagonal boundaries.
         3. Can become arbitrarily complex with enough data
   7. Over-fitting:
      1. Overfitting, formally:
         1. Train Error (known): E(M, D\_{train})
         2. Future Error (unknown): E(M, D\_{all})
         3. Overfit model M if:
            1. If there is some other model M’
            2. E(M, D\_{train}) < E(M’, D\_{all})
      2. Regularization
         1. Grow full tree then prune
            1. How to guide pruning?

Measure performance on train data?

Measure performance on validation data?

* + - 1. Add regularizer to split objective
         1. X\_j = ChooseBestAttribute(T)
         2. If error improvement < lambda^\* #nodes

Then skip

* + - * 1. Determine lambda by validation
  1. Summary:
     1. Decision Tree Classifier:
        1. Representation: Tree
        2. Evaluation: Accuracy
        3. Train: Greedy, Recursive
        4. Test: Traverse tree
        5. Prevent overfit:
           1. Regularize on # of nodes, or
           2. Pruning
     2. Properties:
        1. Good:
           1. Mixed-type data (no 1-of-N encode!)
           2. High dimensions
        2. Classification at test time can take < O(d)!
           1. CF: NN: O(dn), MaxEnt: O(d)
        3. Frequently used in industry
        4. May be Interpretable
        5. Optimal tree is NP-complete
           1. Practical trees are not optimal, but good enough
        6. Some pathological problems can’t be represented as trees

1. **Naïve Bayes**:
   1. Bayes Theorem recap:
      1. Does patient have cancer or not?:
         1. A patient takes a lab test and the result comes back positive. The test returns a correct positive result only 98% of the cases in which the disease is actually present, and a correct negative result in only 97% of the cases in which the disease is not present. Furthermore, 0.008 of the entire population have this cancer.
            1. P(H) = “Prior probability of hypothesis H”, and
            2. P(D|H) “Probability of data D given hypothesis H”
            3. P(H given D) = (p(D given H) \* p(H)) / p(D)

Hypothesis = {C, C}, Data = {+, -}

P(C) = 0.008

P(!C) = 0.992

P(+ given C) = 0.98

P(- given C)=0.02

P(+ given !C)=0.03

P(- given !C)=0.97

Therefore, p(C given +) = (0.98 \* 0.008) / (0.98 \* 0.008 + 0.03 \* 0.992) = 0.2

* 1. Visualisation:
     1. “Headaches are rare and flu is rarer, but if you’re coming down with ‘flu there’s a 50/50 chance you’ll have a headache.”
     2. A large rectangle (state space) with two overlapping smaller rectangles (flu and Headaches) enclosed within it.
     3. Set interpretation
        1. H = “Have a headache”
        2. F = “Coming down with Flu”
        3. P(Large rectangle) is size of set A in the world
        4. P(Headache intersect Flu) is the size of the intersection of set F & H – P(Headache given Flu) is the fraction of the space where Flu is true that Headache is also true
           1. P(H) = 1/10
           2. P(F) = 1/40
           3. P(H|F) = ½
     4. Naïve Bayes Theorem:
        1. P(H given (D\_1 intersect D\_2)) = (P(D\_1 given H) \* P(D\_2 given H) \* p(H)) / P(D\_1 intersect D\_2)
           1. P(H) “Prior probability of hypothesis H”
           2. P(D|H) “Probability of data D given hypothesis H”
     5. NB Classifier:
        1. Naïve Bayes spam classification
           1. P(“Viagra” given Spam)=90%
           2. P(“Viagra” given Ham) =5%
           3. P(“Cheap” given Spam)=60%
           4. P(“Cheap” given Ham) =30%
           5. P(Spam)=10%
           6. P(Ham)=90%
        2. P(Spam given Cheap) = p(C given S) \* p(S) / Z = 0.6\*0.1/(0.6\*0.1+0.3\*0.9) = 18%
        3. P(Spam given Viagra) = p(V given S) \* p(S) / Z = 0.9\*0.1/(0.9\*0.1 + 0.05\*0.9) = 67%
        4. P(Spam given (Cheap intersect Viagra)) = p(V given S) \* p(C given S) \* p(S) / Z
           1. = 0.6\*0.9\*0.1/(0.6\*0.9\*0.1+0.3\*0.05\*0.9) = 80%
     6. NB Classifier: Continuous data:
        1. For continuous data, often model p(D given H) as Gaussian
           1. P(S given (x\*\_{col} intersect x\*\_{len})) = p(x\*\_{col} given S) \* p(x\*\_{len} given S) \* p(S)/K
           2. P(C given (x\*\_{col} intersect x\*\_{len})) =p(\*\_{col} given C) \* p(x\*\_{len} given S) \* p(C)/K
        2. Learning:
           1. To learn the NB classifier, need to fit probability distributions ν

Observe a coin with H,H,H,T,T.

p(Heads given Coin)=3/5, 60%, p(Tails given Coin)=2/5, 40%.

Roll Dice 60 times, observe: 12x1, 8x2, 11x3, 9x4, 14x5, 6x6

P(1 given Dice) = 20%, …, p(6|Dice)=10%.

This is called a binomial/multinomial distribution.

Parameter tells you the bias. [0.6, 0.4], [0.2,0.13,0.18,0.15,0.23.0.1]

Find the parameter that maximizes the probability of the data

W\_{coin} = maximise argument of p(D given W\_{coin})

(N\_j is counting number of outcomes of type j)

* + - 1. Pseudocode:
         1. Find the parameter that maximizes the probability of the data
         2. W = maximise argument of p(D given W)

W\_{jk} = N\_{jk} / sum of N\_{jk}

N\_{jk} = sum of I(x\_{ik} = j) \

Foreach attribute k

Foreach Data i

Foreach state j

N(j,k) +=1 if xik = j

Make N(:,k) sum to 1

* + 1. NB Classifier: Continuous:
       1. To learn the NB classifier, independently find the parameter that maximizes the probability of the training data
       2. For Gaussian:

p(x given (mu intersect sigma)) = (1/sqrt(2\* pi)) \* exp(-(1/2\*sigma^2) \* (x – mu)^2)  
  
where,

mu = (1/N) \* (sum of x), and, (sigma)^2 = (1/N-1) \* (sum of (x\_i – mu)^2)

* + - * 1. {mu, sigma} = argmax p(D given (mu intersect sigma))
      1. D={<x\_i, x\_c, fish>}
         1. ={ <0.1, 0.3, cod>, <0.2, 0.4, cod>, … }
         2. <0.3, 0.2, salm>, <0.4, 0.3, salm>
      2. Then
         1. Cod mu\_{len} = (0.1+0.2+…)/N
         2. Salmon mu\_{len} = (0.3+0.4+….)/N, etc.
    1. NB Classifier: Over-fitting:
       1. What if you had ten spams and no real emails with “viagra”?
          1. Our parameter estimate equation:
          2. P(Viagra|Spam)=10/10+0=100%
          3. P(Viagra|Ham)=0/10+0=0% •
       2. Now you get a long email from a friend that happens to mention Viagra:
          1. The spam evidence from one “Viagra” overrides every other indication of ham from the email. (Multiply by zero)
    2. NB Classifier: Regularisation:
       1. What if you had ten spams and no real emails with “Viagra”?
          1. MLE Learning • P(Viagra given Spam)=10/10+0=100%
          2. P(Viagra given Ham)=0/10+0=0%
       2. Regularized Learning, lambda=1
          1. P(Viagra given Spam)=10+1/11+1=92%
          2. P(Viagra given Ham)=1/11+1=8%
       3. Now, with enough positive evidence, an email could be Ham despite including Viagra
    3. NB Classifier: Overconfidence:
       1. Naïve assumption:
          1. Counting each piece of evidence equally
          2. Not exploiting attribute correlation
    4. NB Classifier: Relation to MaxEnt:
       1. Both classifiers have simple boundaries
       2. For data D={y\_i, x\_i}
       3. MaxEnt: product sum of p(y\_i given (x\_i intersect w))
       4. NB Learning decouples the prior:
          1. You can take your NB cancer classifier to Chernobyl and it will still work…
          2. You can move your NB fish classifier from UK to Norway…
          3. Your MaxEnt cancer classifier will have to re-train from scratch
  1. Online Learning:
     1. Sometimes you want to learn from a data stream instead of from a pre-existing static database.
        1. Because you want to keep your model very up-to-date.
        2. Because your database is too huge to fit in memory, and you don’t want to read it off disk more than once.
           1. Thin task is know as online learning.
        3. Any algorithm can be re-trained from scratch every time a new row is added from the stream.
           1. E.g., MaxEnt you repeat your O(dn) training for each of n data itmes.
           2. Inefficient!! Leads to n \* O(dn)=O(dn^2)
        4. An algorithm that can update the model from the stream in O(1) (i.e., without revisiting the old database) has the Incremental property.
     2. Naïve Bayes is naturally online incremental!
        1. If you want to learn from a continuous stream of observations
           1. Maintain your sufficient statistics N\_j

(i.e., how many times each token j is associated with the current class)

* + - * 1. Add +1 to the appropriate Nj each new observation x\_j
  1. Summary:
     1. Naïve Bayes Classifier:
        1. Representation: Likelihoods Bayes
        2. Evaluation: Likelihood
        3. Train:
           1. Exact, maximum likelihood

(Each attribute independently)

* + - * 1. Set the prior manually or ML
      1. Test: Maximum A-Posteriori
      2. Properties:
         1. Train Complexity: O(dn)
         2. Test Complexity: O(d)
      3. Good in high dimensions
         1. Even d more than n
      4. Good for Big Data
         1. Incremental online
         2. One-pass
      5. Can change priors
         1. Good for mixed-type data

1. **Performance Metrics**:
   1. Accuracy:  
        
      Accuracy = (1/N) \* (sum of I \* (y\_i^{est} = y\_i^{tru}))  
      1. If classifier makes predictions y\_{est} and the true values are y\_{tru}
      2. Accuracy: Percentage of correct answers
      3. Advantages:
         1. Easy, single number
      4. Limitations:
         1. Doesn’t account for imbalanced data:
            1. E.g., Loans: 90% of people overall pay back their loan
            2. Bank classifies good/bad borrowers to make lending decisions
            3. If classify all as good => 90% “accurate” …but useless!
         2. Doesn’t account for which mistakes are made
         3. Doesn’t account for classifier calibration
   2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Actual | |
| Predicted | True Positive | False Positive |
| False Negative | True Negative |

* + 1. The confusion matrix compares how many instances of each actual category are predicted as each estimated category.
       1. The sum of the confusion matrix diagonal gives the accuracy.

|  |  |  |
| --- | --- | --- |
|  | Actual  G B | |
| Predicted G  B | 3 | 4 |
| 1 | 2 |

|  |  |  |
| --- | --- | --- |
|  | Actual  G B | |
| Predicted G  B | 3 | 1 |
| 4 | 2 |

The confusion matrix compares how many instances of each actual category are predicted as each estimated category.

* + 1. Sometimes which mistakes you make matter more than the total number of mistakes
       1. E.g., Loans. Predicting good/bad credit
    2. Consider two classifier results
       1. Accuracy = 50% in each case
       2. Both classifiers get the bank 3 loans worth of interest payments
       3. But which is more useful?
       4. Classifier A: Lost business: 1, Bad Loans: 4
       5. Classifier B: Lost business: 4, Bad Loans: 1
    3. Accuracy results can mislead if you have imbalanced data
       1. Normalised confusion matrix can reveal this
    4. Different applications care about different parts of the confusion matrix.
       1. E.g., bank cares more about minimizing FPR (bad loans) than FNR (lost business)
       2. E.g., High security system cares more about minimizing FNR (permitted breakins) than FPR (false alarms).
    5. Why? Because each outcome has a different cost.
  1. Expected value:
     1. Which loan classifier is better, and by how much?
        1. A makes more good loans, but B makes less bad loans
        2. Expected Value calculation gives a single number given a confusion matrix and cost matrix   
             
           EV = P(Outcome1)\*Val(Outcome1)+P(Outcome2)\*Val(Outcome2)
     2. Costs:

|  |  |  |
| --- | --- | --- |
|  | Actual  G B | |
| Predicted G  B | $2 | $-4 |
| $-0.1 | $-0.15 |

|  |  |  |
| --- | --- | --- |
|  | Actual  G B | |
| Predicted G  B | 3 | 4 |
| 1 | 2 |

|  |  |  |
| --- | --- | --- |
|  | Actual  G B | |
| Predicted G  B | 2 | 0 |
| 2 | 6 |

EV = (2\*3 - 0.1\*1 - 4\*4 – 0.1\*2)/10 = $-1 per customer

EV = (2\*2 - 0.1\*2 - 0\*4 - 0.1\*6)/10 = $0.32 per customer

* 1. ROC Curve:
     1. All the metrics discussed so far depend on classifier calibration
        1. You are correct if your final estimate is y\_{tru} = y\_{est}
        2. But good binary classifiers can output a confidence as well as a class…
        3. By default, the classifier says: y\_{est} = 1 if p(y) more than 0.5
           1. If you are worried about FPs, you could say: y\_{est}=1 if p(y) > 0.75
           2. If you want to maximize TPs you could say: y\_{est}=1 if p(y) > 0.25
        4. This threshold will change the distribution in the confusion matrix
           1. Since threshold is user/business context dependent….
        5. Is there a way to evaluate a classifier independently of the threshold?
           1. So, we can evaluate independently of the end user
     2. Consider a variety of thresholds
        1. Each threshold defines a TPR and FPR
        2. The ROC curve is a graph with y-axis: TPRs and x-axis: FPRs
           1. Receiver operating characteristic
        3. Better ROC curves approach the top left
        4. Area under the ROC curve is a threshold-independent measure of goodness
           1. AUROC: Perfect: 1, Worst: 0, Random: 0.5