Mid Term Project Due October 19, 2020

- Measurement Space $D = \times_{n=1}^{N} L_n$
  - Measurement d has N components $d = (d_1, \ldots, d_N)$
  - Input $|L_n| = M_n,\ n = 1, \ldots, N$;
  - The program should then know that $L_n = \{0, \ldots, M_n - 1\}$
- Given $(d_1, \ldots, d_N)$ compute the linear address
- Input K, The Number of Classes
- Input e the $K \times K$ economic gain matrix
- Input either $P(d \mid c)$, the class conditional probabilities
- Or The data set $(\langle x_1, \ldots, x_Z \rangle, \langle c_1, \ldots c_Z \rangle)$
- Input $P(c)$ the prior probabilities
- Compute $P(c \mid d)$
- Compute the Discrete Bayes Rule $f_d(c)$
- Compute the $K \times K$ Confusion Matrix
- Compute the Expected Gain

Mid Term Project: The Experiments

- Design a generator for synthetic data sets five or six dimensional (The data set generator will appear later in this slide set.)
- Test your software on the synthetic data set
- Make the random number seed something that the user enters so you can do repeatable experiments
- Report the expected gain for the training set
- Report the expected gain from the cross-validation
- Give your data set to one of the other members in the class to run on their software
- Compare your results and if they are different then work together to help find the bugs
- Write everything up in a report; one report for each student.

Constructing a Discrete Bayes Decision Rule

There are three items which are needed for a decision rule to be constructed:

- The class conditional probabilities $P(d \mid c)$
- The prior probabilities $P(c)$
- The economic gain matrix $e(\text{true } c_i, \text{assigned } c_j)$
- This requires knowing
  - The number of classes K
  - The dimensionality N of the mearsurement tuples
  - The number of possible values $M_1, \ldots, M_N$ each component of the N-tuple can have
- The program must have the possibility to input each of these items from a file

Memory Constraints

- Each class conditional table will have $T = \prod_{n=1}^{N} M_n$ entries
- There will be K class conditional tables
- There will be K^2 entries in the economic gain matrix
- And a table of prior probabilities K
- Assume that each table entry is a 32 bit floating point entry
- The memory size is then $K + K^2 + T * K$ floating point entries
- Denote the allowable allocatable memory by Z_{max}
- Z_{max} is an internal parameter of the programs
- Must have $K \prod_{n=1}^{N} M_n < Z_{max}$ floating point linear arrays
  - For each of the K tables, $\prod_{n=1}^{N} M_n$ must be less than $max_{allocatable-size}$

Evaluating Classifier Performance

- Choose the performance criterion – like expected gain
- Choose a data set
  - Minimaly, a data set consists of two sequences
    - A sequence of measurement tuples $\langle x_1, \ldots, x_Z \rangle$
    - A corresponding sequence of the true class tags $\langle c_1, \ldots, c_Z \rangle$
- Then choose the type of classifier and set its tuning parameters (hyperparameters)
  - Setting the tuning parameters must be done *A Priori*
  - Before working with or *peeking* in the data set
- Balance out the order dependencies in data set
  - Randomly Shuffle the data set
- Use the Cross-Validation method to determine Classifier performance

Shuffling the Data

- Sometimes the data set is given in some arranged order
- With correlation between successive data items
- Resulting in different segments having different performance characteristics
  - This leads to expected gains that are biased
    - Low
    - High
  - Because the training data will have an imbalance in number of tuples tagged with each class
  - And the testing data has the reverse imbalance
  - And because the training data may have a dependence between each n-tuple and its neighboring n-tuples
- Random Shuffling the Data
  - Each class tagged tuple has the same probability of being positioned at each position of the sequence

Permuting The Data

- Let the total measurement data sequence be $\langle x_1, \ldots, x_Z \rangle$
- Each x_i is an N-tuple
  - Component 1 has M_1 possible values
  - Component 2 has M_2 possible values
  - Component N has M_N possible values
- Let $\langle c_1, \ldots c_Z \rangle$ be the true class tags, each $c_z \in \{1, \ldots, K\}$
- Let $n_1, \ldots n_Z$ be a random permutation of $1, \ldots, Z$
- Randomly permute the data sequence $\langle x_{n_1}, x_{n_2}, \ldots x_{n_Z} \rangle$
- Use the same permutation to permute the class tags $\langle c_{n_1}, \ldots, c_{n_Z} \rangle$

Advantages of Cross-Validation

- Validation is an essential technique in Machine Learning and Data Science
- Allows using more of the data sequence for training
- Overcomes the problem that the testing set is more complex than the training set
- It produces an unbiased estimate of the expected gain

Cross-Validation

- Allows using more of the data set for training
- Each N-tuple gets to be in the testing sequence once
- Partition the data into V-Folds
  - Train on all but one Fold
  - Test on the Fold left out
  - Round Robin the Fold left out
- If the Fold confusion matrix is expressed in counts,
  - The combined confusion matrix is the sum of the fold confusion matrices
  - Normalize the final confusion matrix so that its entries are probabilities and sum to 1
  - Use it and the economic gain matrix to estimate the expected gain.

V-Fold Cross Validation

- Partition $< n_1, \ldots, n_Z >$ into a V-block partition
- Making the size of each block be approximately the same
  - J_1 items in the first block
  - J_2 items in the second block
  - J_V items in the V^{th}
  - $\sum_{v=1}^{V} J_v = Z$
  - For $v \in \{1, \ldots, V\}$
    - $Q_v = \sum_{i=1}^{v} J_i$

The partitioned permutation then is

$$
\begin{array}{ll}
\text{Block 1:} & \langle n_1, \ldots, n_{Q_1} \rangle \\
\text{Block 2:} & \langle n_{Q_1+1}, \ldots, n_{Q_2} \rangle \\
\text{Block 3:} & \langle n_{Q_2+1}, \ldots, n_{Q_3} \rangle \\
\vdots & \vdots \\
\text{Block I:} & \langle n_{Q_{V-1}+1}, \ldots, n_{Q_V} \rangle
\end{array}
$$

V-Folds

Fold#	Measurements	Class Tags
1	$\langle x_{n_1}, \ldots, x_{n_{Q_1}} \rangle$	$\langle c_{n_1}, \ldots, c_{n_{Q_1}} \rangle$
2	$\langle x_{n_{Q_1+1}}, \ldots, x_{n_{Q_2}} \rangle$	$\langle c_{n_{Q_1+1}}, \ldots, c_{n_{Q_2}} \rangle$
$\vdots$	$\vdots$	$\vdots$
V	$\langle x_{n_{Q_{V-1}+1}}, \ldots, x_{n_{Q_V}} \rangle$	$\langle c_{n_{Q_{V-1}+1}}, \ldots, c_{n_{Q_V}} \rangle$

Folds

After doing all the permutations and partitioning, we overload:
The new x's are the permuted old x's

Fold#	Measurements	Class Tags
1	$\langle x_1, \ldots, x_{Q_1} \rangle$	$\langle c_1, \ldots, c_{Q_1} \rangle$
2	$\langle x_{Q_1+1}, \ldots, x_{Q_2} \rangle$	$\langle c_{Q_1+1}, \ldots, c_{Q_2} \rangle$
$\vdots$	$\vdots$	$\vdots$
V	$\langle x_{Q_{V-1}+1}, \ldots, x_{Q_V} \rangle$	$\langle c_{Q_{V-1}+1}, \ldots, c_{Q_V} \rangle$

Round Robin

Training Folds	Testing Folds
$\{1, \ldots, V-1\}$	$\{V\}$
$\{1, \ldots, V-2, V\}$	$\{V-1\}$
$\{1, \ldots, V-3, V-1, V\}$	$\{V-2\}$
$\vdots$	$\vdots$
$\{2, \ldots, V\}$	$\{1\}$

Classifier Optimization

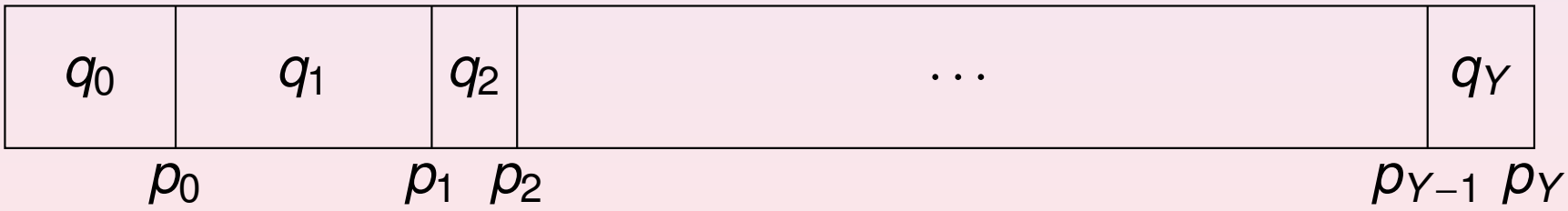
- Almost every Classifier algorithm has Tuning Parameters
- After setting the Tuning Parameters, the Classifier's performance is evaluated
- Improve the Classifier by optimizing the tuning parameters using the same data set
- However, now the principle that the Classifier must be constructed A Priori is violated
  - Meaning the Classifier and its tuning parameters must be chosen before working with the data set
- The optimized Classifier's performance is biased high because the tuning parameters were changed after working with the data set
- This is called *Peeking* and *Peeking* is not allowed!
- *Peeking* is cheating!

Protocol For Optimizing A Classifier

- Optimizing a Classifier must be done out of the loop of cross-validation
- If only one optimization is done, there needs to be two data sets
- Cross-Validation using the first data set establishes the initial Classifier performance
- The optimization can be done using the first data set
- Then Cross-Validation using the second data set establishes the optimized Classifier performance
- Each additional optimization must be done with an additional independent data set

Generate A One Dimensional Cummulative Probability Distribution

- The smallest value is 0
- Input Y, the largest value
- Program needs to generate the probabilities
  - Load an array of size $Y + 1$ with uniform $[0, 1]$ pseudo random numbers
  - Normalize the sum to create the probabilities $\langle q_0, \ldots, q_Y \rangle$
  - Form the Cummulative Distribution: $\langle p_0, \ldots, p_Y \rangle$
    - For $i \in \{0, \ldots, Y\}$
    - $p_i = \sum_{j=0}^{i} q_j$



If x is $U[0, 1]$ then $Prob(p_1 < x \leq p_2) = q_2$

Background

For each class c, the class conditional probabilities are $P(d \mid c)$

- The measurement tuple d has N components
- The range of values for each component can be different
- The Measurement Space is $\mathcal{M} = \times_{n=1}^{N} L_n$
  - $L_n = \{0, \ldots, M_n - 1\}$
- Memory is addressed linearly so a function f is used to map an N-tuple into its linear address
  - $f : \times_{n=1}^{N} L_n \rightarrow \{0, \ldots, \prod_{n=1}^{N} |L_n| - 1\}$
- There is also its inverse g that maps a linear address into its corresponding tuple
  - $g : \{0, \ldots, \prod_{n=1}^{N} M_n - 1\} \rightarrow \times_{n=1}^{N} L_n$
- https://softwareengineering.stackexchange.com/questions/212808/treating-a-1d-data-structure-as-2d-grid
- https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays

Generate The Array Of Class Conditional Probabilities

- Define $S = \prod_{n=1}^{N} M_n \ - 1$
- S is the last address in the measurement space array
- Choose the number of classes K
- Each class c will be an integer in $C = \{1, \ldots, K\}$
- For each class $c \in C$
- Generate a one-dimensional probability function $p : \mathcal{M} \rightarrow [0, 1]$
  - Load an array of size $S + 1$ with uniform $[0, 1]$ pseudo random numbers
  - Normalize the sum to create the probabilities $\langle q_0, \ldots, q_S \rangle$
  - Copy $\langle q_0, \ldots, q_S \rangle$ into the address space for $P(d \mid c)$

Generate Class Assignment Function

This is for the purpose of creating the class tags part of the data set we will be generating

- Load an array of size K with uniform $[0, 1]$ pseudo random numbers
- Normalize the sum to create the probabilities $\langle q_1, \ldots, q_K \rangle$
- Generate the cummulative probability distribution $\langle p_1, \ldots, p_K \rangle$
  - For $i \in C$
    - $p_i = \sum_{j=1}^{i} q_j$
- Generate the class assignment function $h : \mathcal{M} \rightarrow C$
  - For $i \in \mathcal{M}$
  - Generate a uniform $[0, 1]$ number x
  - If $x < p_1$ set $h(i) = 1$
  - If for any $k \in C - \{K\}$, $p_k \leq x < p_{k+1}$ set $h[i] = k + 1$

Generate 1D Sample from a Cummulative Probability Distribution

- The Cummulative Proability Distribution is $\langle p_0, \ldots, p_Y \rangle$
- Each value in the sample will be in the set $J = \{0, \ldots Y\}$
- The sample size is Z
- For $i \in \{1, \ldots, Z\}$
  - Generate x a uniform $[0, 1]$ pseudo random number
  - If $x \leq p_0$, generate a sample value $v_i = 0$
  - If for any $j \in J - \{Y\}$ $p_j < x \leq p_{j+1}$ generate a sample value of $v_i = j + 1$

Generate A Measurement Sequence

Generate the measurement sequence $\langle x_1, \ldots x_Z \rangle$

- Size of measurement sequence is Z
- Number of components is N
  - For each $n \in \{1, \ldots, N\}$
  - Generate a cummulative probability distribution defined on range set L_n
  - Use the n^{th} cummulative probability distribution to generate component n of the tuples in the measurement sequence
- The resulting measurement sequence is $\langle x_1, \ldots x_Z \rangle$
- Each $x_z \in \mathcal{M} = \bigtimes_{n=1}^{N} L_n$

Generate The Class Tag Sequence

- The class tag sequence is $\langle c_1, \ldots, c_Z \rangle$
- The class assignment function is $h : \mathcal{M} \rightarrow C$
- $\langle c_1, \ldots, c_Z \rangle = \langle h(x_1), \ldots h(x_Z) \rangle$

Improvements

- We have generated the data set $(\langle x_1, \ldots, x_Z \rangle, \langle c_1, \ldots, c_Z \rangle)$
- A discrete Bayes Rule will perform badly on this data set because the class conditional distributions overlap
- This is alright because we will be able to gradually change the class conditional probabilities to increase the expected gain
- We will observe this in our experiments

Modifying The Class Conditional Probabilities

- Choose $0 < \Delta < .05$
- For $d \in \mathcal{M}$
  - By the Bayes decision Rule, d's assigned class is c
  - This is because the economic gain for this assignment c is greater than any other assignment
  - But the true class is $c' \neq c$
  - In this case the conditional probability $P(d \mid c')$ needs to be increased
  - Increase $P(d \mid c')$ by Δ
- Renormalize so that For each $c \in C$, $\sum_{d \in \mathcal{M}} P(d \mid c) = 1$
- Using the modified class conditionals, regenerate the measurement sequence and the class tag sequence
- Make a graph of the expected economic gain as a function of the number of iterations that Δ modified the class conditional probabilities
- You should see that the expected gain increases monotonically as a function of iteration number

Interim Report Due on September 21

The period of time from the first class to September 21 is for programming as specified for the midterm project. Make a list of the subroutines you will need to program. For each subroutine provide a mathematical description of what the subroutine is suppose to accomplish. You can copy any of the information on the slides for your description. Indicate in the report and in the subroutine comment lines the type and data structure for each argument. Each subroutine should be in its own file. Following the unit testing protocol, associated with each subroutine should be a mainline that tests the subroutine to verify that to some reasonable extent, it is working correctly.

Interim Report, Continued

If you have difficulty with any of the material in the report, you can explain the difficulty in the report. For example is there information missing that prevents you from completing a subroutine that you want to write. Is there any ambiguity in the slides that needs to clarified? Is there any assumption that you need to make to write the code? It is OK to have difficulty. After writing out what the difficulty is, go back to the slides and find the slide that has the information you need. If there is no slide that provides the information, then you can indicate that in your report. And I will respond with a slide that clarifies that problem, put a revised file on the website that you and the others in the class can go through.

Interim Report, Continued

Besides learning about Machine Learning, the work that you do in the course is to give you concrete opportunities to think and be conscious about what you are doing: what may be your hidden assumptions and the reasons why you may need something to be clarified. Should you discover that when rethinking about a difficulty you get the full picture and can complete the work, you have accomplished something.

Interim Report, Continued

If the difficulty remains, I will work with you individually to help you solve the difficulty. I will help you to find the place in your thinking that stopped you or where there was an assumption in your thinking that you did not realize is either wrong or inappropriate. Many times when I help you I will be able to lead you to point that you realize that everything that you needed to know, you in fact knew. But you did not access that information. So you got blocked. Thinking is about transcending being blocked.

The appendix of the interim report should have a listing of the code you have completed. And the results of your unit testing

The amount of time I expect it will take you to do the programming and writeup for what you include in the interim report is about 25 solid hours.