CISC/CMPE 251 Data Analytics (DA) Exam Review

Tips

Number 1 mistake when people make when they answer design questions is: they read the question, then start writing. Take your time! Thinking beats writing, try mind maps.

Before you start a design question, review techniques, then go through a process of using those techniques.

Remember every data set is different; will never be asked a design question where you do all the obvious things and get the answer. There's always some property of the data set that's going to mean some choices are better than others. Design questions are getting at how you make those decisions on how to choose one pathway over another.

Generic Process (can customize for whichever design question)

1. What's the problem — Almost always not obvious what the real problem is that you're trying to address. Given some problem by the organization that owns the data that they want to solve, but have to map that problem into what the data analytic problem is that you'll be working with to solve their bigger problem. A domain expert is helpful because they can map the organization's/the system's view of what the problem is into your view of what the data analytic problem is.
2. Look at the data — What are the sampling issues? Missing data? Errors in the data? Size/shape of the data. Statistics - help us to understand something about what the attributes are like, and therefore, gives hints about which attributes might be important in the process going forward
3. Clustering — Sometimes the problem is a clustering problem; want to understand something about the similarity structure in the data set that we've been given. This step is the major step that everything else is working towards. If we're interested in making a prediction model, still useful to do clustering to see how the clusters agree with the outcome class labels we have, so we can see how difficult the prediction problem is likely to be.
4. Prediction — First time around, try some simple predictors to get some sense of how they work. This is an iterative phase because unlikely to pick best choices the first time.
5. Look at results — Could be lucky and these results are good enough, but probably not. Very often from these results, do things like attribute selection to improve the predictions, and might loop back to prediction a couple times.
6. Serious prediction — Having cleaned up the data set using a simple predictor, maybe got rid of the obvious unhelpful attributes. When doing attribute selection, not about picking the best attributes, it's about removing the worst attributes.
7. Assess the results — Might go back to the beginning of prediction or serious prediction to see if we can tweak the process to get better results.
8. Assess implications for the problem — Difference between saying reporting prediction accuracy and what it means.

Never exactly what you're going to follow but provides a checklist of things you should be thinking about for a problem that involves a complex data set.

Two types of problems:

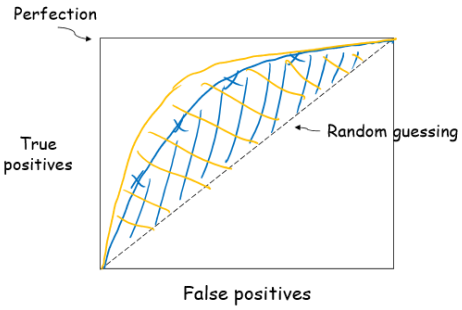
- Prediction: Want to learn the relationship between the properties of an observation and a particular outcome that is associated with that observation. Have outcome attribute (aka target attribute) and have observations and look at cases when an action has occurred. Outcomes usually depend on multiple variables. Want to predict outcome for new records. Prediction is trying to decide what an outcome is likely to be based on an observation. Can predict new, unobserved outcomes
- Exploration (aka clustering): want to understand the structure of observations. Are there groups of similar observations?

Predicting

- **Classification -** there are only a finite number of outcomes
- **Regression -** the target attribute values are continuous. Can subdivides classification problem. Regression is harder to do than classification.
- **Measuring Performance (Classification)**
  - Prediction accuracy – [True Positive (TP) + True Negative (TN)]/ Total outcomes. How many were correct?
  - Precision – TP/[TP+FP (False Positive)]. How many of the "Yes" outcome, did we label correctly?
  - Recall – TP/[TP+FN (False Negative)]. How many of the "Yes" outcome, did we label?

- F-score – 2TP/[2TP + FP (False Positive) + FN). Is the weighted average of Precision and Recall. Much better representation of how the model is doing when the two sets of outcomes are of very different sizes (avoids skewing). For example, when the number of outcome values has more than another such as much more wins than losses.

- **Transparent** – you can tell why the predictor made that decision. I.e. It tells you how it got to that outcome
- **Opaque** – you cannot tell why the predictor made that decision. I.e. It doesn't tell you how it got to that outcome
- **ROC** (Receiver Operating Characteristic) **Curve** – a method to fine tune where the boundary between outcome classes should be. Build the model several times over with the crank in different positions so different numbers of FP and you get several TP that's different in those cases. As you increase the number of FP, the number of TPs changes, join points in a curve. Interested in the quality of the model over all possible ways that we can tweak the boundary to get slightly different answers. Turns out it's really measured by the area under the curve and a dotted line representing random guessing. So, for one model, can make different choices where the boundary should sit. The ROC curve gives us a way to compare one model to another because if we do this with another model it will have a slightly different profile, but the interesting thing is how big is the area between that curve and the dashed line. The larger the area between the ROC curve and the random guessing line, the better that model is.



- **Measuring Performance (Regression)**
  - Want minimal error between prediction and actual value
  - Standard way: **Root Mean Squared Error** (RMSE)
    - Problem with this method is when there's one test set error which has a very big error, then when squared it contributes a lot to the sum which doesn't get removed when square rooted. RMSEs come from fitting linear functions, but not so clear they make sense, in context with how well the predictor is fitting the testing.
  - Better way: **Mean Absolute Error** (MAE)
    - Take the absolute value of the difference between real and predicted value for each test record, add them up and divide by the number of test records. This weighs the errors of all records equally. MAE is in the same units as whatever's being predicted, so it's directly interpretable.

Clustering

- Need some idea of **similarity**
  - Metric-based techniques: closeness
  - Distribution-based techniques: frequency
  - Hierarchical techniques
  - Matrix decompositions
- Want to see clusters that are tight (**intracluster**) and well separated (**intercluster**)

Data considerations

- What attributes should be/are collected?
- How big is the data set, both observation-wise, and attribute-wise?
- **Missing data**
  - Not possible to collect all the attributes for all objects/records/observations. I.e. stars and surveys
  - Matters why some values are missing
    - **Random**: some hope of replacing them by appropriate surrogate values
    - **Systematic**: much harder to compensate
  - Some data analytics techniques require all data to be present so missing values must be replaced
  - Others can still build model, but there must still be some hidden assumptions about them, which may need considering
- **Wrong data**
  - Error in transcription when data is collected by humans
  - Sometimes data was collected for some other reason without considering DA
  - Often useful to look at a distribution of the values of each attribute. Errors may show up as outliers, but sometimes outliers are the interesting part, so they should only be removed with caution
- **Sampling issues**
  - Selection process can have major impact on the models that result
  - **Bias** – property that comes from sampling in a way which systematically selects some kind of data in expense to other kinds of data.

- **Variance –** this is the way the model and its performance are because it was built from this sample rather than that sample. Might be more or less equivalent for many kinds of statistical process you might apply to them, but it's still the case where there are some records in this one and not in that one. This suggests, to reduce variance, a bigger sample is better than a smaller one. However, this doesn't work out quite the way it seems.
- Almost all the theory of statistics is based on the gold standard statistical sample, **iid** (**independent and identically distributed**). If you have 2 samples, you can't tell the difference between them by any kind of process that makes sense to measure them by; real world samples are not like that overall. So many theoretical results in statistics technically don't hold, but this doesn't matter. As an empirical fact, most of the results hold pretty well.

Data Manipulation

- Categorical →Numeric
  - Mapping categories into numbers doesn't work because algorithm might say one category is more different than another.
  - **Solution**: create a regular polygon where each vertex is equidistant from another (points on the circumference of a circle) and therefore, categories are mapped into coordinate points of an nth-dimension.
  - Expensive solution because if there are k different values for the categorial attribute then there will be k-1 new attributes.
  - **1-hot encoding** – 1 new attribute for all of the possible values that the categorical attribute can take on, so if we have k values, now we have k new attributes. Instead of complicated numerical transformations, we now just put in the respective category. This is a lot simpler and it preserves the similarity between the different locations because if you draw this in 3D space, you'll see each point is a vertex of the triangle. Expensive again, because if there are k values, you get k new attributes.
- Ordinal Categorical → Numeric
  - Problem becomes more difficult when there's a weak order of the categorical attributes.
  - Mapping the example categories (different stages in life) into 1-8 does work, but domain experts know that you'll be losing information since each category has a range of values that are not uniformly distributed. Hence, the 2nd option, with approximately the average age of each category is a better numerical representation. Notice how any choice embeds assumptions, so be very careful so that the assumption doesn't sneak in anything that isn't true unless you really must.
- Numeric → Categorical (discretization, binning)
  - Use **equal-interval bins**, then give that range of values a name (label). Then replace each observed value with the category's name. Think about how many bins you want to preserve. The more categories there are, the less blurring of values that are the same. Problem: equal-size bins pays no attention to the distribution of the values in the data
  - Use **equal-frequency bins** which is much more dependent on the distribution of the data. Smaller bins when there are lots of points in them, and large when there are fewer points in them. Each bin has about the same number of records allocated to it, so it captures something about the distribution of the numeric values as well. Choose how many bins ahead of time, and then allocate the same number of values to each bin. Bin boundaries become the (min value of the bin to the right minus max value of current bin)/2 added to the max value of the current bin.
- **Normalization** – map values into a specific range, typically [0, 1] or [-1, 1]
- **Standardization** – transforms data to have a mean = 0 and standard deviation = 1

Model Evaluation

- Different predictors produce different accuracies.
  - **Good fit** if different predictors have similar accuracies
  - **Bad fit** if different predictors have different accuracies; could be because the training subset for one predictor is sensitive to a record in one of the training sets which is not found in the others.
- **Cross validation –** don't have different training sets to try it on, so we use the data we have in multiple different ways.
  - Divide the dataset into training and test sets in multiple different ways
  - Build a model from each of the training sets and rate it, using each of the test sets
  - Use the average performance over the different partitions to judge how we're doing
- **Out-of-bag** (OOB) testing (better way) – create a training set by copying objects in the dataset randomly (allow sampling with replacement). Training set will have 2/3 of records, test set will have 1/3, but the test set will act as if it was size n.

Prediction Techniques

Predictor	Analysis
Naïve Bayes	Two assumptions: 1. all attributes are equally important, 2. All attributes are statistically independent (knowing the value of one attribute says nothing about the value of another) Good for when attributes are independent of each other. Bad when attributes are better used together; there are relationships between other attributes. Can not have too many attributes. Does not do when there are duplicates in the data. Bad for repeated attributes
K Nearest Neighbour	Good for smaller data sets. Don't need deeper understanding of the data. Good for data that isn't clumpy. Bad when data set is large. Need deeper understanding of the data. Always must consider k.
Decision Trees	Explainable model because each branch is a rule. You can read your way down a branch and figure out how to reach a leaf, the outcome. Limitations: They are moderately effective They are building blocks for more effective models. Good because they are useful for humans A decision tree becomes a logical extension of a checklist. Constructing a decision tree from real data puts the sequence of decisions into a structured form that humans can use so long as it doesn't exceed 10-15 nodes. I.e. diagnosing heart attacks. Bad because computationally expensive **Brittle** - slight training data change creates an entirely different tree with the same accuracy as the other one. Therefore, there could be multiple explainable models. This makes a decision tree not an opaque, but also not a transparent, predictor. It's about halfway in between.
Neural Networks	Good when output is a non-linear function of the inputs, when the dependencies among the attributes are messy and information gain or similar measures don't do well (in short, attributes are related to each other), when there are many attributes that each have some predictive power Bad because it takes long to train. Not good with small data set because there won't be enough nudging to stabilize the weight values, so accuracy will be bad.
Support Vector Machine (SVM)	Great for 2-class classification. Works extremely well for problems that require a higher dimension (i.e. more difficult boundaries). Efficient because of the use of kernel tricks. Bad for classification problems with more than 2 outcomes. Do not use this for regression!
Random Forests	Advantages: Handles multiple classes Fast (worse than 1 decision tree, better than a NN) Low variance, low bias (because of the availability of test sets) No separate test procedure needed (no cross-validation) High accuracy Effective for large number of attributes Helpful for attribute selection Does not overfit no matter how many trees Disadvantages: Opaque predictor (because final decision is going to be based on votes or averaging across many individual predictors; can use attribute selection to know which ones affect outcome the most though)

	A little expensive to deploy (because every time it receives a new record it must be passed to all the component predictors

Clustering Techniques

Clustering	Analysis
K-Means	Good for when not much is known about the data. Don't know how many clusters. Bad if clusters are different sizes, clusters aren't spherical.
Expectation Maximization	Good when there is missing information. Good for when there is a reason and variation. Can use this if making assumptions about the data Technique of choice when enough is known of the data to make plausible guesses. Must consider how many distributions and what their shape should be
Hierarchical Clustering	Useful because gives dendrogram view of clustering and can be used with a wider range of data Expensive – O(n^3)
DBSCAN	Great for outlier detection. Also, good when clusters are not spherical. Again, don't know in advance what good choices for these properties will be, MinPts is probably easier to guess from the domain because we want to pay attention to the big clusters and not the small ones. Epsilon can be harder and may require iteration as usual.
SVD	Turns out to be useful to have S because the sigmas tell us how much variation there is in each of the dimensions in the transformed data. Interpret it as V is the new axes starting from the one that passes through the data cloud in direction with maximal variation, and the orthogonal one captures the most uncorrelated/leftover correlation. U becomes the new coordinates with respect to those new axes. This is why we use 3 matrices and not 2. It's figuring out the maximal variation of the cloud regardless of the way the data was presented. One of the payoffs of doing that is can have a representation of the data in k-dimensions instead of m-dimensions. With z-scores, some entries can be negative, so forces can be both push and pull. If I imagine there is a weak force which forces all of the points away from the origin, and that's being counter-balanced by the pull of the forces between object and attribute, then the SVD results in exactly the stable position of each point. What you see is a representation of both the objects and attributes in a consistent way so that objects are pulled towards attributes for which they have large values, and exactly symmetrically, attributes are pulled towards objects for which they have large values. Can use the fact that there's this mutuality in the process to understand what's going on For example, if know that objects get pulled to attributes, and therefore indirectly to other objects, then if you have very similar objects that are trying to be like every other objects, the net pull end up somewhere near the origin because there's no where else to be since every object is pulling. If you're an object that isn't related strongly to any other objects, then it inhabits a dimension that will collapse when truncated down to k. If you think about an object like this, in the reduced k-dimension space, when it lost its own dimension, it'll also end up near the origin. Also good for visualization and figuring out k-parameter.

Ensembles are good for regression

Best predictor needs to match dataset size, outcome size, attribute relationships