GENERAL ASSEMBLY

DAT SCIENCE

CLASS 9: DATA CLEANING AND MANIPULATION

I. ENCODING & BINNING CATEGORICAL DATA
II. FINDING FEATURES WITH PERFECT CORRELATION / NO VARIATION
III. FEATURE STANDARDIZATION & IMPUTATION
IV. RECURSIVE FEATURE ELIMINATION

I. ENCODING & BINNING CATEGORICAL DATA

ENCODING & BINNING CATEGORICAL DATA

- Recall that in our last class, we discussed how all models avaliable in scikit-learn only accept numerical features as input.

- To get around this, we create binary variables to reflect the different labels of our categorical feature.

- But, what if we're dealing with data with multiple categories? What if there are 100+ categories in a feature? Creating binary variables for each feature would be annoying and exhausting.

    - The answer: we use scikit-learn's **label encoding methods (otherwise called Binarization).**

- But, what about categories that appear just once or twice in the dataset? Do we need a label for them?
  - **No!** We need to ‘bin’ these labels into an ‘other’ category. Otherwise, your model **will break** when doing cross validation and prediction on new data.

- How do we decide which categories to ‘bin’ and which to retain?
  - There’s no one right answer, but we typically use heuristics like their overall percentage prevalence in the dataset, or a minimum number of observations needed for inclusion.

- What happens if the new data has a label that didn’t appear in our test or training set?
  - **We also bin it into our ‘other’ category.**

- Let's start making a list of what we need to do to get our data ready for use by our predictive model, and what to do to it on its way back out of the model for use by our consumer. So far, we have:

1. Split your data into categorical and numeric data
2. Fill numeric NaNs through imputation or a simple number (more on this later)
3. Fill categorical NaNs with 'Nothing'
4. **Detect low-frequency levels in categorical features and bin them under 'other'**
5. **Encode each categorical variable into a sequence of binary variables.**
6. Merge your encoded categorical data with your numeric data
7. Train and test your model on the data.
8. Re-clean and encode incoming unlabeled data.
9. Input your new data into the model.
10. Retrieve and relay the model's results to a database, message queue, or other service.

II. FINDING FEATURES WITH PERFECT CORRELATION AND / OR NO VARIATION

FINDING FEATURES WITH NO VARIATION

- Recall from class 5 that machine learning models provide a **representation** of the data.

- How do you think the model uses the features to create this representation?
  - Each feature needs **variation** so that it can be helpful in predicting your dependent variable.

- If a feature has no variation, either before or after it is cleaned (through methods such as label binning and encoding), it has **no use for the model.**

- Luckily pandas has a nice way for us to find features with no variation using its pandas.DataFrame.value_counts() method.

- Features with perfect correlation present a similar problem – if more than one feature moves in lock step with all others, then each feature (beyond the first feature identified) provides no additional information to the model.

- As such, you will need to filter these perfectly correlated variables out.

- A **heat map** is a handy way to view correlation between variables (and especially what variables are best correlated with your response feature).

- However, the usefulness of heat maps quickly begins to falter as you add additional features.

- As such, **automated means** to filter out perfectly correlated features are of paramount importance.

- Let's go back to our data cleaning pipeline. We're going to now add a few new items to it:
  1. Split your data into categorical and numeric data
  2. Fill numeric NaNs through imputation or a simple number (more on this later)
  3. Fill categorical NaNs with 'Nothing'
  4. Detect low-frequency levels in categorical features and bin them under 'other'
  5. Encode each categorical variable into a sequence of binary variables.
  6. Merge your encoded categorical data with your numeric data
  7. **Remove features with no variation.**
  8. **Remove perfectly correlated features.**
  9. Train and test your model on the data.
  10. Re-clean and encode incoming unlabeled data.
  11. Input your new data into the model.
  12. Retrieve and relay the model's results to a database, message queue, or other service.

IN CLASS EXERCISE: ENCODING AND BINNING CATEGORICAL DATA

- Using the same query we used in the last class to predict hall of fame induction, lets:
  1. Create a new feature showing the team each player played on the most in his career.
  2. Split the data into string and numeric columns
  3. Bin low-frequency categorical data
  4. Encode string data
  5. Find and drop features with no variance
  6. Find and drop features with perfect correlation to others.

III. FEATURE STANDARDIZATION & IMPUTATION

- Many machine learning models (such as support vector machines) require data to be normally distributed, with a mean of 0 and a standard deviation of 1.
- Others simply become more accurate when you transform the data in this manner.
- Scikit-learn has a handy .scale() method that lets you automatically perform this on your data.
- Be aware, however, that you must apply the same standardization to your new data, or else your model will interpret your raw data as already standardized!
- You may also scale features to a range, although this is more time-intensive and sometimes more arbitrary than standard scaling.

FEATURE IMPUTATION

- There are many common ways to dealing with NaNs:
  - Dropping rows with then – but what if all but one feature has data?
  - Filling with an extreme value or zero – but what if you're using a linear method? Or if you worry you'll create arbitrary tree splits?
  - Filling with the mean value – but what if the mean value is nonsensical? Or if you have extreme values that skew the mean in a certain direction?

- In these situations, most data scientists use the following methods:
    - Last observation carried forward (LOCF): if you have multiple observations for the same individual or item in your sample, you can carry forward the data from a previous observation to the most recent one.
    - KNN: If your NaNed observations has one or more ‘like’ observations, you can fill your NANs by interpolating based on nearest neighbors.
    - Tree-based methods: using the k-1 features in your independent feature data set, you can create a best ‘guess’ of the missing feature by creating a predictive model for it.

- You can test out which imputation method creates the best predictive accuracy; however, simpler methods (such as filling with the mean value) are often just as predictive as more complicated ones.
- Much like centering and scaling your data, you need to re-transform new incoming data using the same imputation technique for your predictive model to handle it appropriately.
- Scikit-learn has a great Imputer method that lets you do just that – fit an imputer using a strategy on your training data, and then re-use the imputer on new incoming data.
- However, the method is limited to simple strategies – mean, median, and most frequent, although more advanced methods are on the way in future versions.

- Going back to our data cleaning pipeline:
  1. Split your data into categorical and numeric data
  2. **Fill numeric NaNs through imputation**
  3. Fill categorical NaNs with ‘Nothing’
  4. Detect low-frequency levels in categorical features and bin them under ‘other’
  5. Encode each categorical variable into a sequence of binary variables.
  6. Merge your encoded categorical data with your numeric data
  7. Remove feature with no variation
  8. Remove perfectly correlated features
  9. **Scale your data with zero mean and unit variance**
  10. Train and test your model on the data
  11. Re-clean, encode, and scale incoming unlabeled data
  12. Input your new data into the model
  13. Retrieve and relay the model’s results to a database, message queue, or other service

IV. RECURSIVE FEATURE ELIMINATION

- In many cases, you want to reduce your feature set to only those that truly help predict your dependent variable.

- These include:
  1. Use of a model type (such as KNN) where irrelevant features decrease accuracy;
  2. Use of unsupervised dimensionality reduction techniques that rely on feature distances to extract meaning from the data;
  3. Optimizing your model for speed;
  4. Reducing data overhead in preprocessing the data and re-processing results; and
  5. Identifying the most predictive set of features for further investigation.

- You can accomplish this by recursive feature elimination.
- In it, you start by running your full model with all features, and sort your features by some measure of importance.
- Then, you begin eliminating features starting with those calculated as the least importance.
- At each elimination, you calculate cross-validated accuracy metrics. If cross-validated accuracy is the same or better than your maximal model, you save the narrowed list of features and repeat the process again.
- You continue either (1) you remove all but your most important feature from the model; or (2) cross-validated accuracy of your narrowed model is **worse** than your k+1 features model.

- Most data scientists ‘stop’ the process when all features are exhausted due to local maxima/minima in the data.

- Others (especially those doing RFE on linear models) don’t believe in it at all – they believe it gives an arbitrary representation to the data.

- Instead of eliminating by variable importance, you can also explore all possible combinations of the features, but this can be extremely computationally intensive for large numbers of features (ex: for 36 features, 36! = 3.7x10^41 potential model test runs)
  - Combining this with cross validation, your computational overhead would likely not be worth the additional accuracy gain of this method.

- You can combine RFE with grid search (discussed in the last class) to find both the optimal set of features and the optimal tuning parameters for your model.

- One ‘Gotcha’ with RFE in scikit-learn is that it only works with model objects that have a .coef_ attribute, which most tree-based methods do not have.

- To get around this, create a new class of your tree-based method with a .coef_ attribute equal with the already-existing .feature_importances_ attribute.

- Going back to our data cleaning pipeline:
  1. Split your data into categorical and numeric data
  2. Fill numeric NaNs through imputation
  3. Fill categorical NaNs with ‘Nothing’
  4. Detect low-frequency levels in categorical features and bin them under ‘other’
  5. Encode each categorical variable into a sequence of binary variables.
  6. Merge your encoded categorical data with your numeric data
  7. Remove feature with no variation
  8. Remove perfectly correlated features
  9. Scale your data with zero mean and unit variance
  10. **Perform grid search and RFE on your data to find the optimal estimator for your data.**
  11. Train and test your model on the data (if not done as part of #10)
  12. Re-clean, encode, and scale incoming unlabeled data
  13. Input your new data into the model
  14. Retrieve and relay the model’s results to a database, message queue, or other service

IN CLASS EXERCISE: SCALING, IMPUTATION, AND RFE

- Building on your existing script:
  - Center and scale your data.
  - Change our imputation strategy from ‘mean’ to ‘median’ to help protect against extreme values.
  - Perform recursive feature elimination on your data.
  - Plot out the results.
  - Preform a combination of grid-search and RFE to find the best features for the best tuning parameters.

QUESTIONS?