Real World Database Pre-Processing for Machine Learning/Data Mining

What to expect

We are going to learn about preparing data from databases for downstream machine learning / data mining. In particular these 3 core topics:

- Data cleaning from single tables
- Common table formats and transformations
- Feature engineering from relational tables

We will take a problem scenario focused approach to learning and will largely revolve around solving common scenarios or issues experienced in practice.

What is data pre-processing?

There are many definitions of pre-processing, but in general it refers to the tasks that create a transformed dataset that machine learning or data mining algorithms are applied to [4, 2].

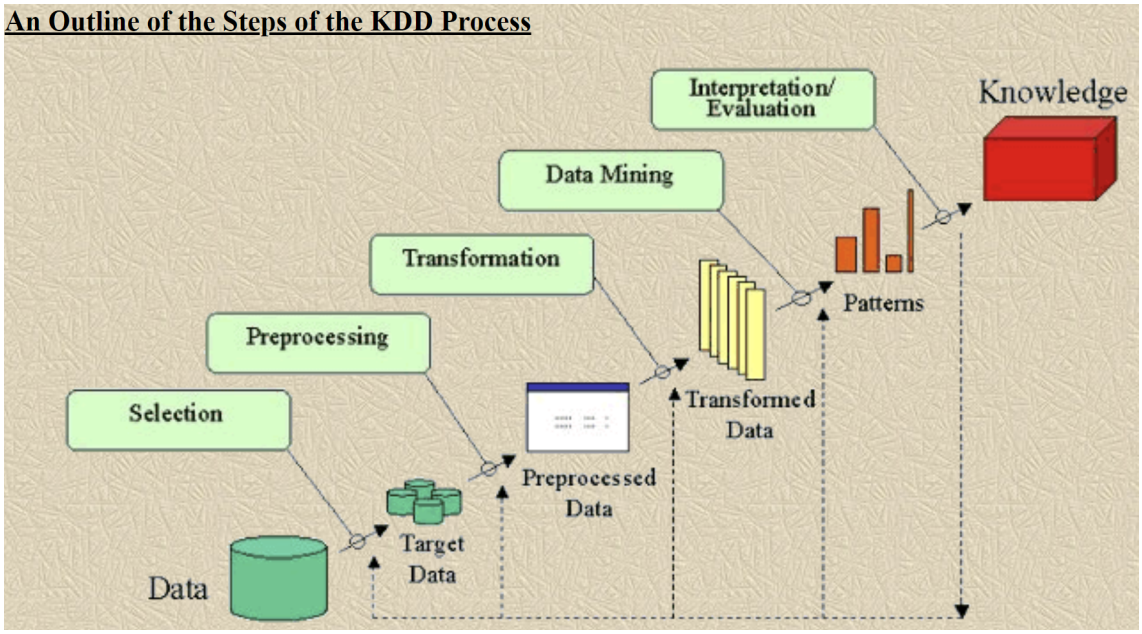

Figure 1: A Data Mining view of pre-processing [2]

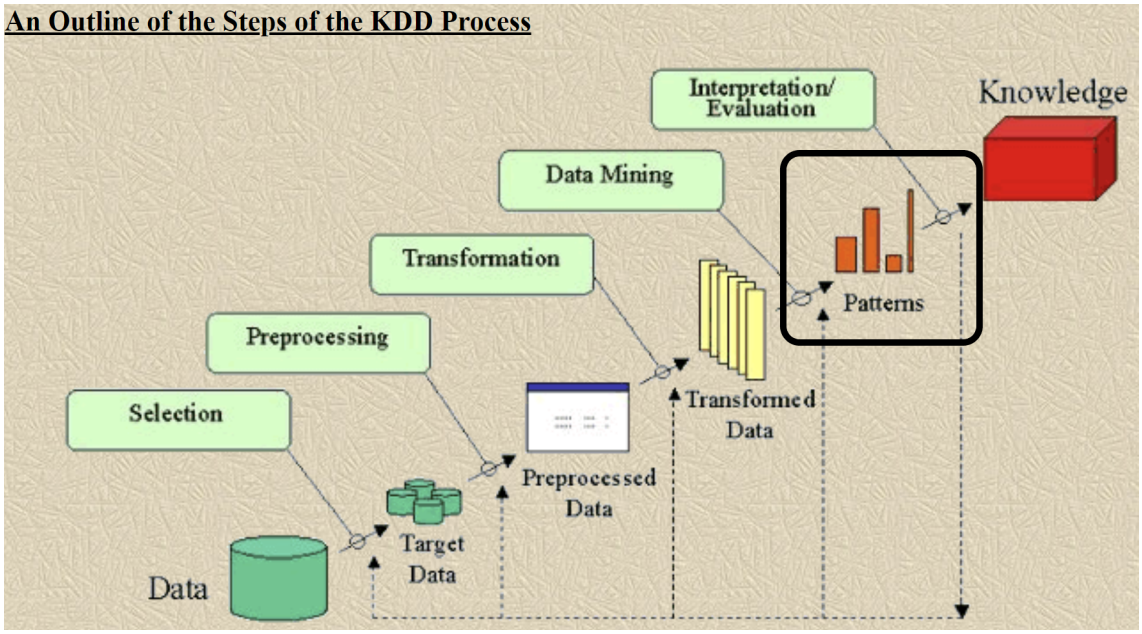

Figure 2: Typically we learn about the algorithms to find patterns from clean data sources

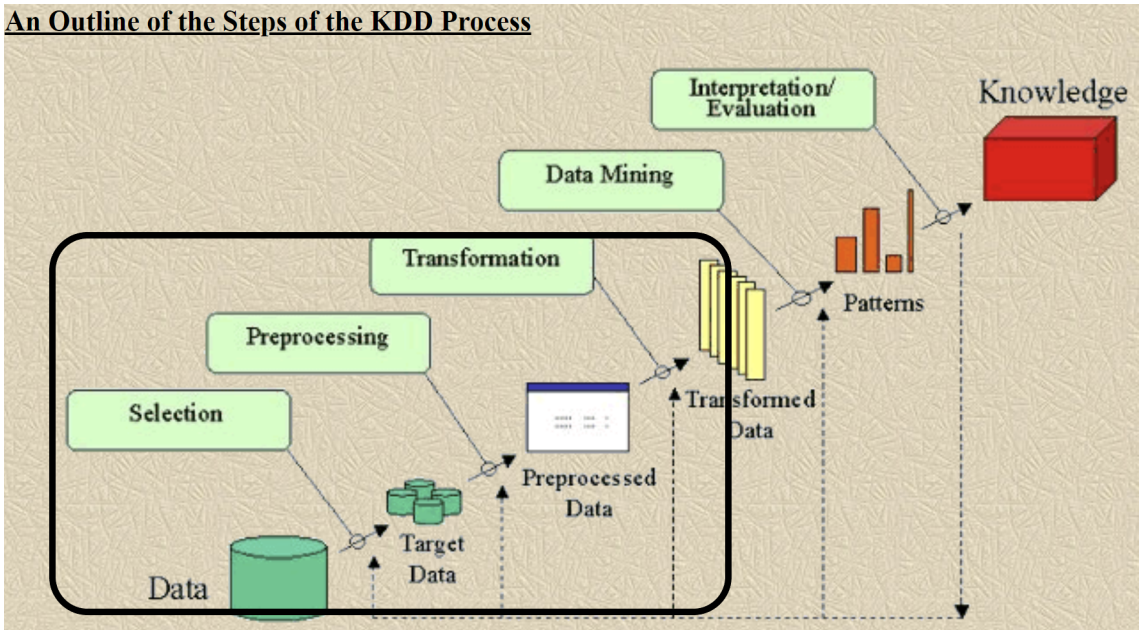

Figure 3: Most of the time spent in real projects is in data pre-processing (including transformation)

Depending how good/clean the data is inside an organisation, pre-processing could even take up to 95% of a projects time. The typical split that gets quoted is 80% spent on data preparation and 20% spent modelling [1]. Therefore in terms of time spent, it is useful to learn how to deal different qualities of raw data.

Missing Data Situations

Types of missing data

There are more types of missing data but these two cases are useful to consider. For a more complete background: `http://www.stat.columbia.edu/~gelman/arm/missing.pdf`

- MCAR: Missing Completely at Random. The process behind whether a value is missing or not is independent of any values in our dataset.
- MAR: Missing at Random. The probability of a value becoming missing is related to the other values in our dataset and can therefore be predicted or inferred.

Simple Imputation

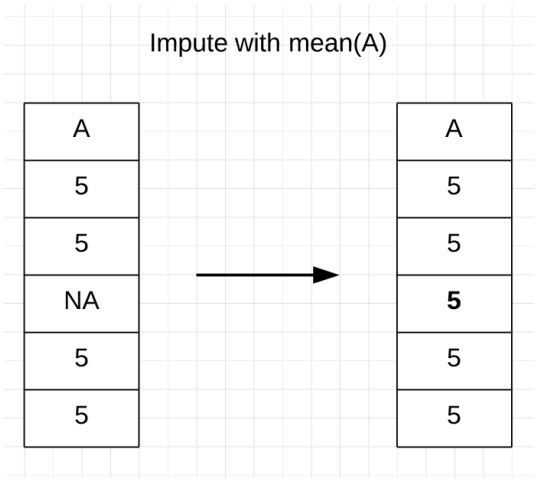

Figure 4: A simple imputation method. We impute the missing value in A by calculating the mean of the non-missing values of A

Example: Competitor Price Scraping

Company A is a rental car company and wants to use the pricing behaviour of Company B to help set their own prices. Company A sets up a web scraper to gather prices on Company B's website.

Company B's Pricing Data			
Date	bigCar	mediumCar	smallCar
2018-01-01	100	80	50
2018-02-01	90	70	40
2018-03-01	110	NA	50
2018-04-01	NA	60	40
2018-05-01	NA	NA	NA
2018-06-01	80	60	70

Figure 5: Some values have gone missing as the scraper didn't work as expected

There are a lot of packages and libraries out there to help with imputation

in R or Python which make it easier to implement more advanced methods. However, there are advantages to simple methods to begin with. Often you are just interested in exploring the data before you try and maximise accuracy, or your favourite algorithm does not handle missing values (NA) natively so you must clean them.

Given the scenario above and the types of missing data:

1. Discuss in groups a simple solution to clean the data so that you can proceed onto analysis.

2. (we said that the data is time series and we assume todays value is related to yesterdays value. Therefore we can replace NA with the previous days value. Even better, we can average Tomorrow's value and Yesterday's value.)

3. Come up with a MAR situation where your simple algorithm can potentially lead to biased data

4. (Holidays and Special events '2018-05-01' may have higher chance for values to go missing, therefore leading to biased results. If bigCar's NA increases the probability of other NA's to be present and will affect results if bigCar is NA when the price ¿110)

Part 2

Unofficial "best practice" for imputation

You were given the dataset with very little information other than there were missing values you need to fix. It's often difficult to remember everything you've learned, so the ability to search for information is an invaluable skill.

Therefore best practice tends to be:

1. google 'missing values'
2. find reputable sources and learn about the possible solutions are (missing values lecture notes from Columbia)
3. settle on a generally accepted best method (multiple imputation is considered the best method)
4. find packages or libraries that implement these methods (dont always exist)

Figure 6: Our search on multiple imputation leads us to MICE

In our specific case, MICE is a popular method as determined by the number of citations. More citations is generally correlated with the validity of the method as it used in peer reviewed research. Some of the time it is not true since bad methods can be cited many times as a benchmark to beat, however MICE seems to be a fairly recent development (2011).

Data Cleaning with Strings

Regular Expressions

For a practical introduction in Python visit: `https://www.datacamp.com/community/tutorials/python-regular-expression-tutorial`

Regular expressions are a language that is useful in manipulating and searching strings.

For this class we will use the substitution function and is very useful in cleaning strings. **re** is the default regular expression library in python and **sub** is a function that takes in arguments (pattern, replacement, string).

```
import re

filename = '123.csv'
#find the pattern .csv and replace with''
clean_filename=re.sub(pattern='.cs',replace='',string=filename)
print(clean_filename)
# '123'
```

More complex regular expressions can be created but for now a simple find and replace is useful enough.

Example: 'State' snapshots

There are many examples of companies who wish to record the current state of a system. For example, what is the state of our rental car fleet? What is the status of each vehicle in our fleet at this point in time?

2018-08-01 snapshot .csv			
carID	carType	status	Mileage
1	big	rented	100,000
2	medium	maintenance	16,000
3	small	idle	500
4	small	idle	'Not Avail.'
5	medium	idle	'Not Avail.'
6	big	rented	1,800

Figure 7: A snapshot file

This particular company has trouble using databases and loves excel (the most common tool in any corporate). They decide to give you a folder with the snapshots they store daily for analysis.

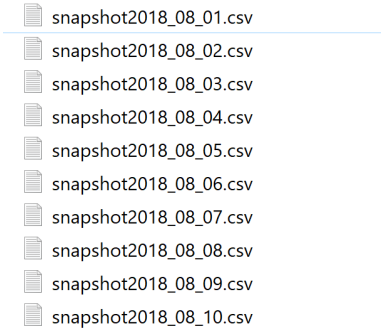

Figure 8: A folder filled with snapshot files, there are hundreds of them

As the 'data guru' in the team, the company asks if you could first visualise how mileage changes over time, but all the snapshots are in different files.

- In groups, come up with a solution to creating a single table from multiple snapshot files. Try to avoid manual input eg. manually encoding snapshot dates in code.
  - We discussed that we would in general:
  - read the first file into memory
  - add a new column 'snapshort_date' by extracting the date from the filename using regular expressions
  - repeat for all files and append them together

After creating a single table with snapshot_date added, you may have noticed an error trying to visualise mileage over time. The Mileage column is not numeric.

- Given the approach we took in the previous question, how can we clean Mileage so that it is numeric.
  - First we replace all instances of ',' as commas are not allowed in numeric types in many languages
  - Second we replace 'Not Avail.' with NA
  - Third, we can optionally use missing value imputation to replace NA with a suitable value.

Part 3

In the first 2 parts we covered simple cases of pre-processing data in single table formats. Part 3 will focus on how we create single table data from multi table formats.

The single table / flat file / dataset

Machine learning or data mining needs clean single table data to apply algorithms to. It needs a clean table where each row of our data set represents an instance of the entity we are interested in.

In this section we are a supermarket chain looking to predict whether or not our current customers will still be shopping with us in 30 days. We have multiple tables of data, from various sources such as demographics from loyalty card membership, surveys we sent out and emails. The task is to predict churn with the highest accuracy possible.

CustomerDemoChurn			
customerID	Gender	Age	Churn
1	M	40	1
2	M	20	0
3	F	21	0
4	F	45	1
5	F	30	1

Figure 9: Customer demographic data, however this table seems a bit empty. Do customer demographics strongly predict churn?

Introducing Joins

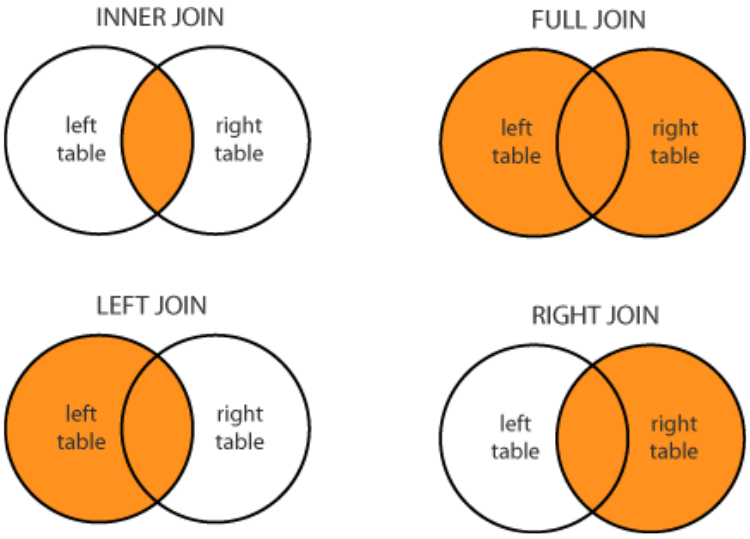

Figure 10: The different types of joins. A join involves 2 tables A and B and a common set of keys or ID's

Surveys				
SurveyID	CustomerID	Income	NPS	FavCategory
1	2	low	10	Wine
2	3	med	10	Wine
3	5	med	3	Seafood
4	4	high	1	Seafood
5	6	low	5	Bakery

Figure 11: Survey Data. Note that we did not send a survey to customer 1. Net Promoter Score(NPS) is a common metric used to measure customer satisfaction. It generally involves a question asking on a scale of 1-10 how likely are you to recommend this product/service to a friend/colleague. FavCategory represents the answer to *w hich category do you care about most?*

CustomerDemoChurn						
customerID	Gender	Age	Churn	Income	NPS	FavCategory
1	M	40	1	NA	NA	NA
2	M	20	0	low	10	Wine
3	F	21	0	med	10	Wine
4	F	45	1	high	1	Seafood
5	F	30	1	med	3	Seafood

Figure 12: Customer demographics table joined with surveys

Wide or Long Data

EmailOpen					
EmailID	CustomerID	Jan1	Jan2	Jan3	Jan4
1	1	NA	1	1	NA
1	2	1	NA	NA	NA
1	3	1	NA	NA	NA
1	4	NA	NA	1	NA
1	5	NA	1	1	1

Figure 13: Wide Data with respect to date. Dates span multiple columns

EmailOpen		
EmailID	CustomerID	Date
1	1	Jan2
1	1	Jan3
1	2	Jan1
1	3	Jan1
1	4	Jan3
1	5	Jan2
1	5	Jan3
1	5	Jan4

Figure 14: Long Data. Dates span 1 column but now there can be many duplicate Email and CustomerID's

EmailOpens		
EmailID	CustomerID	#Opens
1	1	2
1	2	1
1	3	1
1	4	1
1	5	3

Figure 15: Aggregated table by CustomerID. Opens represents the *count* of EmailOpen. Aggregation is important when joining tables with one-to-many relationships

CustomerDemoChurn							
customerID	Gender	Age	Churn	Income	NPS	FavCategory	#Opens
1	M	40	1	NA	NA	NA	2
2	M	20	0	low	10	Wine	1
3	F	21	0	med	10	Wine	1
4	F	45	1	high	1	Seafood	1
5	F	30	1	med	3	Seafood	3

Figure 16: Single table with joined features

Feature Transformations

It is important to transform features before machine learning, especially since many methods such as neural networks only take binary vectors as input.

customerID	LastVisited	Year	Month	Day
1	2017-01-01	2017	01	01
2	2018-08-17	2018	08	17
3	2018-08-10	2018	08	10
4	2017-07-15	2017	07	15
5	2018-03-04	2018	03	04

Figure 17: Month Day Year extraction from dates.

One hot encoding

One hot encoding is a technique which creates binary vectors for each distinct categorical value.

customerID	LastVisited	2017-01-01	2018-08-17	2018-08-10	2017-07-15	2018-03-04
1	2017-01-01	1	0	0	0	0
2	2018-08-17	0	1	0	0	0
3	2018-08-10	0	0	1	0	0
4	2017-07-15	0	0	0	1	0
5	2018-03-04	0	0	0	0	1

Figure 18: A clue to why it is better to transform dates into columns year, month and day

customerID	Year	Month	2017	2018	01	08	07	03
1	2017	01	1		1			
2	2018	08		1		1		
3	2018	08		1		1		
4	2017	07	1				1	
5	2018	03		1				1

Figure 19: Year and month can be treated as categorical values or numeric

Discussion Questions:

- Why should we join tables together?
  - Since machine learning typically learns from single table datasets, the single table contains all the information that can be learned from.
  - While Fig. 9 can possibly use Gender and Age to predict churn, it is likely adding additional features from linked tables such as the Net Promoter Score is a stronger and more informative predictor of churn.
- When should we use long format and when should we use wide format tables?
  - We discussed in class that wide format is good for using with visualisation tools like excel
  - wide is also good if we do not have any missing values
  - long is good, because it will save space when there are many NA values. (it can also be bad if we have many ID columns and few NA's)

- Why is Fig 19. likely better than Fig 18?
  - Think about what information each column represents. In practice, information that is too granular is less likely to be useful for prediction.
  - For example, Fig 18. has columns that represent specific dates, how likely is it that a specific date is can be used to predict churn?.
  - If on exactly 2018-08-17 there was an incident where we sold faulty products, then it can lead to better predictions, but this is rarely the case.
  - Another factor is that specific dates such as 2018-08-17 only happen once. A model that learns based on this feature will likely perform poorly on future data as 2018-08-17 becomes outdated.
  - A more general or less granular feature like Month can represent concepts like seasonal effects which are quite common as well as special month specific information e.g. If a customer first shopped in December (Christmas season) they are likely to churn (one-off or infrequent shopper). Also there are only 12 months in a year and these months stay the same across different years too, so the feature is less likely to become outdated.

Academic Writing Resources for Non-Native English Speakers

The UoA library offers online resources[1] and writing workshops[2]. There is also online resource for academic writing[3], articles about scientific writing[4]. Watching related videos[5] can also be beneficial.

Part 4

Previous week recap

In the last week we learned about a few useful techniques and problems that are commonly faced in industry. In particular we saw:

- Examples of missing values and how we might impute them
- Scenarios where regular expressions would be handy
- How to create a single table from multiple tables
- A little bit of feature engineering/transformations

Feature Engineering

Quick introduction to machine learning (∼6 min) `https://www.youtube.com/watch?v=rwobGhobPzY`

What is feature engineering?

It is the process of generating and selecting features to improve machine learning performance (usually accuracy[6]). It can improve accuracy by giving the algorithm more information to better predict variables such as churn (Fig 16). More features generally increase accuracy, but there is a point where it can reduce accuracy.

Example feature generation:

- Applying a oneHotEncoder function to transform a column with date strings into multiple binary vectors as seen in Fig 18.
- Aggregating EmailOpens with the *count* operator in Fig 15/16

[1] `https://www.coursebuilder.cad.auckland.ac.nz/flexicourses/3196/publish/1/3.html`

[2] `https://www.library.auckland.ac.nz/workshops/index.php?p=courses_workshops`

[3] `http://www.socialresearchmethods.net/kb/writeup.php`

[4] `https://cseweb.ucsd.edu/~swanson/papers/science-of-writing.pdf`

[5] `https://www.youtube.com/channel/UCaJD4vYNav5Zub15IyplZ0g`

[6] Since this course is not a machine learning course, when we talk about accuracy it actually means the test set accuracy or generalisation accuracy. Accuracy refers to the performance of a machine learning model, therefore higher is better

- Extracting Day Month Year from the date string in Figure 17

Typically when generating features, we test to see if they improve accuracy then decide if we want to select the feature we generated. Beyond basic data preparation tasks, this can be the most time consuming task. It is also likely to be the most important [1]. Unfortunately, the accepted best practise is to use experience and dark art expertise (intuition and guessing).

EmailOpen		
EmailID	CustomerID	Date
1	1	Jan2
1	1	Jan3
1	2	Jan1
1	3	Jan1
1	4	Jan3
1	5	Jan2
1	5	Jan3
1	5	Jan4

Figure 20: Long form email data. There are one or many CustomerID's

CustomerDemoChurn							
customerID	Gender	Age	Churn	Income	NPS	FavCategory	#Opens
1	M	40	1	NA	NA	NA	2
2	M	20	0	low	10	Wine	1
3	F	21	0	med	10	Wine	1
4	F	45	1	high	1	Seafood	1
5	F	30	1	med	3	Seafood	3

Figure 21: Single Table to support churn prediction

Revisting the supermarket case study in the previous week:

- What other features can we create from Fig 20, to include in Fig 21 other than '#Opens'? ('#Opens' represents an aggregation on Customer ID where we calculate the count for each customerID)
  - We can use other aggregation operators such as sum, min, max, mean, percentile, but there are many more types of transformations that are possible.

Assume we have extra tables below:

Transactions		
CustomerID	TransactionID	Amount
1	1	100
2	2	60
3	3	70
4	4	20
1	5	50
2	6	60
5	7	90
4	8	70

Figure 22: A table of transactions. Amount is the total amount of money spent for each transaction

TransactionLineItems			
ProductID	TransactionID	isExpired	Type
1	1	1	Seafood
2	1	0	Flowers
3	2	0	Fruit
4	3	0	Vege
1	3	0	Seafood
1	4	1	Seafood
1	5	1	Seafood
4	6	0	Vege
3	7	0	Fruit
2	7	0	Flowers
4	7	0	Vege
1	8	0	Seafood

Figure 23: A table of transactions with potentially multiple products per transaction. It helps us identify which items were in a particular transaction and also contains additional information about the products

- What other features can we create from Fig 23, to expand our single table (Fig 21)? and what are the steps we need to take to derive it so that it can be joined into the single table?
  - Example of creating a feature based on TransactionLineItems.Type (represents product type)
  - We are trying to construct a feature that represents which type of product a customer purchases the most
  - First we apply oneHotEncoder to TransactionLineItems.Type which results in 4 new columns TransactionLineItems.Type_Seafood, TransactionLineItems.Type_Flowers, TransactionLineItems.Type_Fruit and TransactionLineItems.Type_Vege
  - Second we aggregate (sum) our 4 new features in TransactionLineItems by TransactionID.
  - Third, we join these features from the previous step into the Transactions table

- Fourth, we aggregate (sum) the features from the previous step by CustomerID
- Fifth, the resulting features in the fourth step are joined onto the CustomerDemoChurn table and for simplicity we will name these total_Seafood, total_Flowers, total_Fruit, total_Vege.
- Sixth, to calculate a new feature called FavouriteType we pick the product type which has the highest number of items purchased e.g. if total_Seafood *geq* (total_Flowers, total_Fruit, total_Vege) then FavouriteType = Seafood. Ties can be broken randomly.
- Seventh, we actually have 5 new features in CustomerDemoChurn, but we can optionally remove the 4 features created in step 5 if desired.

part 5

Open Problems in Feature Engineering

In the previous lecture we discussed different features we could derive to include into our single table from related tables.

We also saw that:

- CustomerDemoChurn has a one to many relationship with Transactions via CustomerID
- Transactions has a one to many relationship with TransactionLineItems via TransactionID

So to join a feature isExpired into the CustomerDemoChurn table:

- Let agg_i represent an aggregate function $i \in \{sum, min, max, meanorcount\}$
- $f_1 = agg_i$(TransactionLineItems.isExpired) by TransactionID
- Left join f_1 into Transactions
- $f_1 = agg_i(f_1)$ by CustomerID
- Left join f_1 into CustomerDemoChurn
- The feature in plain English represents the total number of expired items bought by each customer

It is straightforward to derive one type of feature from isExpired. However, as we can see above there are multiple transformations to choose from in agg_i. Should we use the sum or the count or the mean? Most of the time we will choose the combination of transformations that we think make the most sense. It is a very manual process.

Brute Force Aggregation

The most cited paper that relates to relational feature engineering is The Data Science Machine. They advocate for brute force feature generation followed by feature selection, but this can become an issue for both memory and time when successively apply sets of transformations.

For example. If we have 5 transformation functions (sum,min,max,mean,count) we can create 5 features from isExpired to join into Transactions. The intermediate features created then need to be joined into CustomerDemoChurn, which produces another 5 features for each of the 5 features already generated. Imagine having to apply another set of transformations (log, square, cube, sqrt, exp), the number of features generated is 5 * 5 * 5. 125 Features to represent information derived from TransactionLineItems.isExpired.

To simplify the problem, we only consider aggregation transformations. The feature space is defines all possible features we can create. Naturally this

means it is defined by the transformation operations we use and the number of times we recursively apply them, in this case the transformations are (sum,min,max,mean,count) and with 2 one-to-many joins we apply them twice. The number of one-to-many transformations (or transformations in general) required to transform a feature (TransactionLineITems.isExpired) into the base table (CustomerDemoChurn) is called depth. The number of features generated for a particular depth d and set of transformations k is k^d.

There is very little research about how to do feature engineering from relational tables in an efficient way.

- When you have limited time to spend on feature engineering, what are some practical strategies to increase your chance of creating a good feature?
  - Use domain expertise to decide which transformations and features may be most promising (more experience required)
  - Use more compute resources (more $$ required)
  - Give up and focus on the machine learning (too common...)
  - Many projects decide not to do much or any feature engineering as it can be a long, painful and costly process. They rely on using more advanced machine learning methods and parameter tuning with the hope that the machine learning algorithms will implicitly learn the features anyway.
- When is brute force appropriate?
  - When the data is small
  - When the total number of features is small
  - When you have little domain expertise
  - Otherwise, it is computationally expensive and practically infeasible
- How can we make feature engineering more efficient?
  - It is an open problem, but we discussed various ideas in groups
  - Use machine learning to identify which features to search first
  - Use traditional search methods like Depth first search, breadth first search to search transformation paths
  - Use less transformations, especially when two transformations are known to generate similar features.

part 6

A Possible Linear Time Solution to Relational Feature Engineering - Under Review

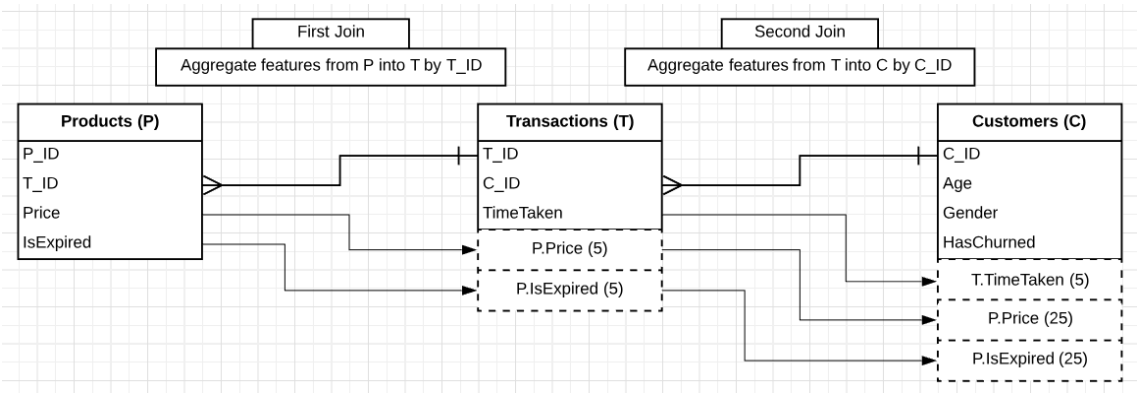

Figure 24: Similar to the example in class but not the same. Assume we have 2 pre-defined join operations which successively aggregate features from the table P into table C. Defining which joins to make is another topic and there is not much research that addresses this problem for feature engineering. So in practice we arbitrarily define which joins we perform and the order we perform them.

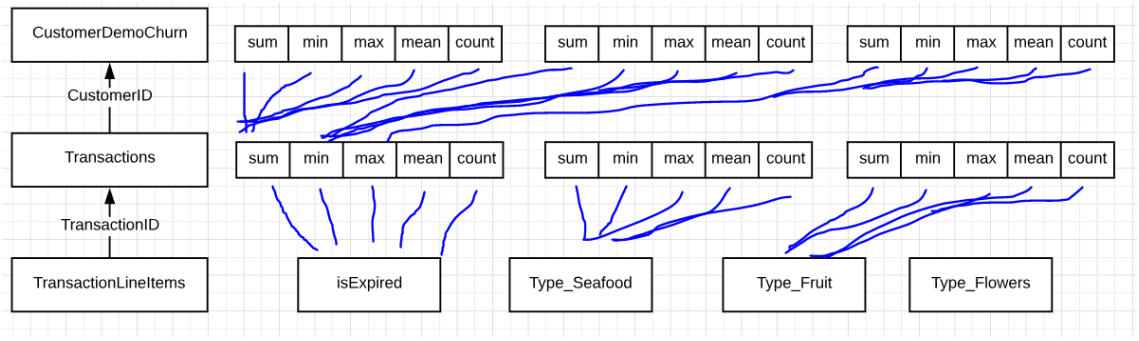

Figure 25: In class example of brute force feature engineering.

- Issues with brute force
  - Too many features
  - Time and space complexity is exponential
  - A lot of algorithms assume features are independent.... (Naive Bayes Classifier, Unsupervised clustering/feature selection)

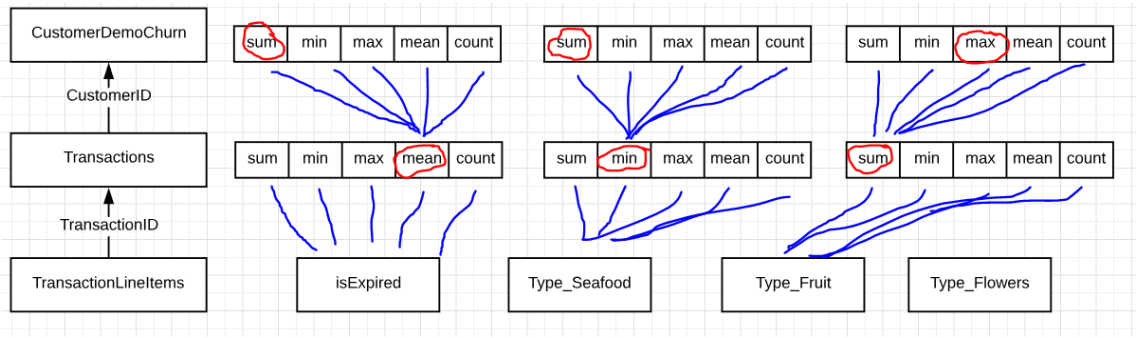

Figure 26: Embedding a feature selection process at each feature generation step to select one best feature. Guarantees linear run time.

A feature generation step is a single step where a set of transformations are applied onto an input feature. eg. isExpired is an input feature and 5 simple aggregation transformations are applied.

If we embed feature selection at each feature generation step, we can reduce or prevent the combinatorial explosion of features. If we only select one feature to survive at each generation step and prune the rest, we can guarantee linear run time.

However, selecting a subset of features results in information loss. Therefore we must decide how to select features that minimise the information loss. A simple method is to measure the pairwise correlation between candidate features. The intuition is that if features are highly correlated, they contain similar information. Correlation between a pair of features also represents the ability for a feature to predict the other feature in the pair. This means that selecting the feature with the highest pairwise correlation is also minimising the linear reconstruction error if we use that single feature to predict all other features in the same candidate feature set.

How good is it?

- Scales much better than brute force
- Preliminary results suggest it has competitive accuracy

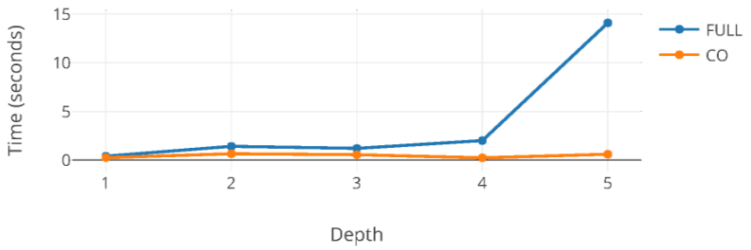

Figure 27: It scales really well vs brute force in terms of run time

Table 1. Number of cases where classification accuracy is within best result. ChooseOne has particularly high performance across all classifier combinations while LS (Laplace Scoring) has the worst performance

Classifier	DT		LR		NB		Max: 84
Method	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Total Cases
CO	**13**	**13**	**12**	**12**	**12**	**12**	**74**
COR	11	9	11	11	8	9	59
FULL	10	10	8	9	8	7	52
LS	6	6	4	4	4	4	28
MCFS	9	9	9	9	7	6	49
PCA	5	5	8	8	11	11	48
SPEC	6	5	7	7	5	5	35

Figure 28: Unpublished results suggest it is better than FULL (Brute force) and all other unsupervised feature selection algorithms that use the full dataset. It was tested on 7 publicly available datasets. CO = ChooseOne

Machine Learning Based Methods

Explorekit[3], Learning Feature Engineering (LFE) [5].

General Concept:

- Gather 100s of publicly available datasets
- Train model to learn what historical patterns relate to 'good features'
- Apply to new unseen datasets and hope it works

Explorekit can reduce classification error by $\sim$ 20%, while LFE produces a new set of features that improves accuracy in 90+% of cases compared with no feature engineering.

However, these papers only explore feature engineering from already preprocessed single table datasets. They don't do relational feature engineering, or prepare data from multiple tables. This is fine, but you're still going to be spending significant amounts of your time in industry engineering features from multiple tables.

References

[1] P. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, Oct. 2012.

[2] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[3] G. Katz, E. C. R. Shin, and D. Song. Explorekit: Automatic feature generation and selection. In *IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[4] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

[5] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga. Learning feature engineering for classification. In *Proceedings of the International Joint Conference on Artificial Intelligence*. IJCAI, 2017.