Lab 9 – Frequent Patterns

Submission: `lab09.ipynb`, pushed to your Git repo on `master`.
Points: 20
Due: Friday, February 21, 2pm

Objectives

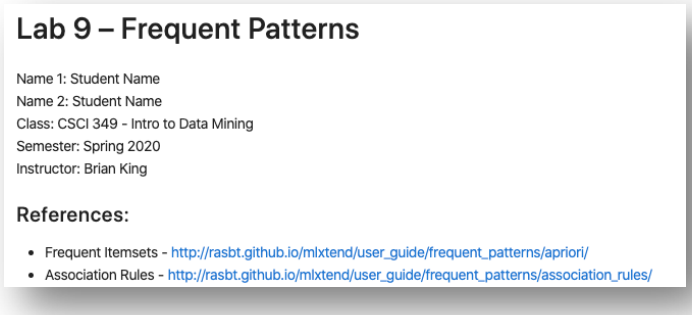
- Because, you just can't get enough pandas
- `mlxtend` – your Python support for transactions, FP mining, and AR generation

Introduction

This lab will introduce you to frequent pattern mining and association rule generation using the `mlxtend` package.

Work with someone!

As a reminder – I strongly recommend you work with someone for the remainder of the work in this course. The work is going to become more challenging, and you and your partners will benefit from the shared, collaborative experience. Be sure to include BOTH partner names in the top header cell. And BOTH members should push up the notebook file to their respective repositories. For example:


Lab 9 – Frequent Patterns

Name 1: Student Name
Name 2: Student Name
Class: CSCI 349 - Intro to Data Mining
Semester: Spring 2020
Instructor: Brian King

References:

- Frequent Itemsets - http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
- Association Rules - http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

Preparing for your lab

Create a **`lab09.ipynb`** file. Create your header cell, then your first cell to set up your imports.

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

Work through this lab, and enter the answers to questions that are scattered throughout this lab. It is quite likely you'll need to add additional imports as you work through the lab.

Installing `mlxtend`

Unfortunately, scikit-learn does not have the packages we need to do association rule and frequent pattern mining. These area available with a few alternative packages. We're going to use the package **`mlxtend`**, a

machine learning set of extensions for python. This library contains many additional features and enhancements to the standard scikit-learn library. See this page for more information: https://github.com/rasbt/mlxtend

The `mlxtend` package is not available from the standard `defaults` channel in conda. We'll install our package from the `conda-forge` channel, which contains an enormous number of packages put together by the conda community. I suggest not adding the channel to your conda installation, as it can substantially slow down future package updates. As usual, we'll do our conda management from the command line. (You can also use Anaconda, but do so at your own risk.)

Follow these steps.

1. Open your terminal, and make your environment for the course active.

2. `$ conda env list`

   This will verify that `csci349` is the current environment (listed with an * next to it.) If it is not, then you didn't do the first step correctly.

3. Just incase you have a problem, I recommend making a backup of your environment:

   `$ conda create --name csci349_backup --clone csci349`
   (Of course, substitute csci349 with whatever name you gave your course environment.

4. `$ conda install mlxtend -c conda-forge`

   This will install the `mlxtend` package directly from the `conda-forge` channel. NOTE: conda goes through a process of "solving your environment" to make sure that any dependencies to `mlxtend` are also installed and/or updated as needed. This may take a minute or so depending on the complexity of your installation.

   NOTE – As of this writing, `mlxtend` is at 0.17, which has not been thoroughly tested with Python 3.7. However, my testing has not revealed any problems. You can accept the warnings that will likely appear.

Exercises

1) [P] Add the import statements:

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

These first exercises are going to allow you to leverage the actual documentation from the mlxtend user guide online. Even though the first few of these are copied from their docs, it's good for you to have these examples in one place. The second part of this lab will use a much more extensive transaction dataset.

2) [P] Go to the page: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/ . Enter the list `dataset` shown on the page. Then, copy the example code that transforms the list to a numpy encoded array, then to a pandas `DataFrame` with the correct column names. Output your data frame. It should look identical to:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

3) [P] Show the result of `describe()` and `info()` on your dataframe.

4) [P] Following along the mlxtend user guide, use the apriori algorithm to final all frequent itemsets with a `min_support` of 0.6. Show the resulting dataframe, and store the result, since you'll have many selection exercises next. All selection exercises must be done from this resulting frame. Set `use_colnames=True`. It'll be much easier to interpret your patterns.

5) [P] Select all frequent itemsets that have support => 0.8

6) [P] Select all frequent itemsets with at least 2 items. In their documentation, they often create additional helper variables to make it easier to select your data. That's entirely up to you. (I tend to be a purist, and reserve additional variables for only very complex selection criteria. Computing the length of an itemset is **not** one of them! It's entirely up to you.)

7) [P] Select the frequent itemsets that contain an `'Onion'` in the itemset.

8) [P] Select the frequent itemsets that contain both `'Onion'` and `'Eggs'` in the itemset. (HINT: You should have 2 frequent itemsets selected. And, if you haven't learned about the `set` type in Python and all of the standard set operations, they can really make these types of questions much easier.)

9) [P] Select the frequent itemsets that contain *either* an `'Onion'` or `'Kidney Beans'` (or both) in the itemset. (HINT: You should have 8 frequent itemsets output.)

10) [P] http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/ contains the all the information about the association rule interestingness metrics, as well as giving you the code to generate the association rules. (NOTE – they mention a function called `generate_rules()`. The function is `association_rules()`.) Generate rules with a minimum confidence of 0.7. Store your resulting data frame called `rules`. Show the entire data frame. (You should have 12 rules).

11) [P] Output the top 5 rules in descending order by "lift", with the secondary sort key by "confidence".

12) [P] Select all rules that have a 1.0 support for the antecedent.

13) [P] Select all rules that have at least 3 or more items represented in the rule (i.e. the union of the antecedent and consequent >= 3.)

14) [P] Select the rules that have confidence >= 0.75 and a lift > 1

Chipotle

Now, it's time to think about food… so grab yerself some chips and salsa, and a burrito bowl and have some indigestion and fun… Why? Because you're going to work with a real dataset from **Chipotle**. Mmmmm….. chips.

These data were originally available for public use on Kaggle (https://www.kaggle.com/)

> Get to know Kaggle at some point soon. If you have not yet set up an account on Kaggle, you should. It's amazing repository of shared knowledge from data scientists worldwide, and also a platform for data mining and machine learning competitions

This dataset represents a single day of transactions at a busy Chipotle store. Each observation represents one item purchased as part of an order. The quantity of that item is also indicated. Each order has one or more observations. The variables are:

- **`order_id`** : A unique identifier for one complete order
- **`quantity`** : The quantity of the item ordered
- **`item_name`** : The official name of the item ordered
- **`choice_description`** : A list of options for the item ordered. (For example, a Chicken Bowl, which is an official item, may have Tomatillo-Red Chili Salsa (Hot), Black Beans, Rice, Cheese, and Sour Cream listed in its choice description.
- **`item_price`** : The total price for the quantity of this item

The `'item_name'` variable is a nominal / categorical variable. `'choice_description'` contains a Python list of items that are part of the main item ordered. (For example, a `Chicken Bowl` may contain `Rice`, `Pinto Beans`, `Fresh Tomato Salsa`, etc.) The rest of the columns are self-explanatory.

Let's first review some basics… these should be "no-brainers" just to warm you up, and remind you of the importance of taking the time to check out your data, do some EDA, and preprocess it (before you eat it… ahem, I mean consume it? Process it? Oh, never mind.)

15) [M] Clearly indicate this new section in your notebook:

```
---
# Chipotle Data
```

16) [P] Read in the Chipotle dataset:

```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'
df_chip = pd.read_csv(url, sep = '\t')
```

17) [P] Show the result of `df_chip.info(verbose=True)` You should have five variables.

18) [P] Show the result of `describe(include='all')`

19) [P] Show the first 10 observations

20) [M] Study what you see so far. Minimally, you should notice that you have three variables that need to be transformed into usable types. Which ones, and what do you need to do with them?

21) [P] Let's start doing some preprocessing. Convert the `item_price` field to a floating-point number.

22) [P] Convert the `item_name` to a categorical variable (HINT: Use `pd.Categorical()`)

23) [P] How many unique `item_name` values are there?

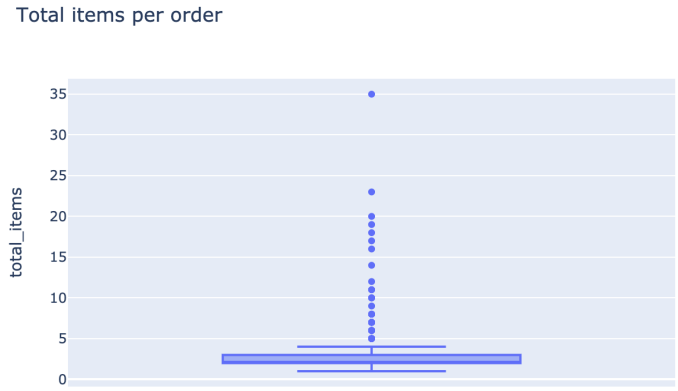
24) [P] Show all of the unique values in `item_name` . Do you see any potential problems? (Leave them! Don't fix them. Just pay close attention. Need a hint? Salsa)

25) [P] How many distinct orders are there?

26) [P] Show a boxplot of the number of line items per order. Do NOT consider the quantity of each item, just the count of line items. (NOTE: Many items appear multiple times in an order. Don't worry about that. Just count the number of lines per order.)

Your result should indicate that the majority of orders only have 1-3 items ordered.

27) [P] Show a boxplot of the TOTAL number of items per order. Now, you must consider the quantity of each item in each order. (The box plot will be similar to the previous, with the exception that there will be more outliers, and the maximum outlier will be 35.)



28) [P] What were the top 5 ordered items by total quantity? Report the item and its total quantity ordered. **Be sure to consider the quantity of each item ordered!**

29) [P] What is the total number of "`Steak Burrito`" ordered?

30) [P] What is mean price for an order (NOTE – This is NOT just a mean of the `item_price` column!)

31) [P] What was total revenue for the day?

32) [P] What was the largest total price for a single order? Show the order number and the total price.

33) [P] Show the entire order to your answer to the previous question

    (NOTE: This should show you how some orders can contain multiple lines of the same item. Not uncommon!)

34) [P] What order had the largest total quantity of items purchased? Show the order number and the total number of items

35) [P] Show the entire order to your answer to the previous question

Frequent Patterns in Chipotle Data

36) [P] Recall that this dataset is a set of transactions, where each observation represents one item purchased as part of an `order_id`. However, the data are not read in this way. You need to transform this dataset to a collection of binary encoded transactions, where each row represents ONE transaction, and the columns are binary encoded variables, with each variable representing ONE item available for purchase at Chipotle. Convert your data. Your resulting data frame should have an index representing the `order_id`, and columns representing each possible item from the `item_name` variable. For now, a transaction will ignore the quantity of item purchased.

    The shape of your resulting data frame should be (1834,50)

37) [P] Show the first 10 observations from your transaction data

38) [P] Too often, many start by considering a minimum support that is arbitrarily large. Go ahead and use the apriori method to generate frequent itemsets with a minsup value of 0.5. What happened?

39) [P] Take a step back. Your previous outcome is why you ALWAYS perform essential EDA tasks before you dive into mining a dataset! Report a table that shows the number of transactions each item occurred in, sorted in order of most frequent to least. NOTE: That number essentially represents the absolute support for 1-itemsets! So, include a column that shows the relative support (i.e. the fraction of total transactions.)

    (HINT: The item with the highest support is 33.5%!)

40) [P] Now, make a smarter decision. Like many large, real-world transaction datasets, data is sparse! You have many variables, and most observations use only a handful of them. This is the definition of a sparse dataset. You need a better minsup value. Regenerate frequent itemsets, but now use a minsup of 0.005. How many frequent itemsets were reported? Report your frequent items sorted by decreasing support order.

41) [M] In the context of association rules, explain the difference between support, confidence, lift, leverage and conviction.

42) [P] Generate all association rules that meet a minimum support of 0.01. How many rules were output in total?

43) [P] Show only the rules that have a **lift** > 2, but sorted in order of decreasing **confidence**. What is your strongest rule?

44) [M] Consider yourself the data scientist hired to help Chipotle understand item purchasing patterns. Interpret the following rule for the non data scientist. Be careful not to say, "if your customers purchase canned soft drinks AND chips, they are also going to buy chicken bowls." Think! What do strong association rules convey?

antecedents	consequents	antecedent support	consequent support	support	confidence	lift
(Canned Soft Drink, Chips)	(Chicken Bowl)	0.031625	0.335333	0.019084	0.603448	1.799551

45) [P] Suppose your boss is interested in what items are most likely related to a purchase of `"Chips and Guacamole"`. Using your rule set generated, first select the rules that have `"Chips and Guacamole"` listed in the consequent itemset. Sort the rules by confidence, then by lift. And interpret your findings. Identify the item(s) that are the most suggestive of including `"Chips and Guacamole"` when purchased, and state why.

46) [P] These data suggest that the `"Chicken Bowl"` is the single most frequent item purchased for this particular location. Great. Let's dive into those chicken bowls more. As you know, Chipotle lets you customize your items. That's what the `choice_description` field is for.

    Process all of the items listed in the `"choice_description"` field by creating a new transaction dataset representing binary encoded transaction data for only Chicken Bowl. Note – this is tricky because the `choice_description` variable is read in as a long string. The string itself represents a list, and quite often, it's a list of lists. You need to process this to be an actual *flattened* list of items. Then, you can easily convert these to transactions. This page will give you an idea on how to do it. https://chrisalbon.com/python/data_wrangling/pandas_expand_cells_containing_lists/ . However, keep in mind that many of these lists contain lists themselves! You may need to just write a python function that takes in a list of lists, and returns a single list of all of the items flattened out. Then, apply that function.

47) [P,M] Use your own knowledge to generate strong frequent patterns and association rules for the choice_description items used with Chicken Bowls. Explain your findings.

Deliverables

Commit and push `lab09.ipynb`. Be sure you have every cell run, and output generated. All plots should have plt.show() as the last command to ensure the plot is generated and viewable on Gitlab. Verify that your file is pushed properly on Gitlab. Be sure each question is properly annotated.