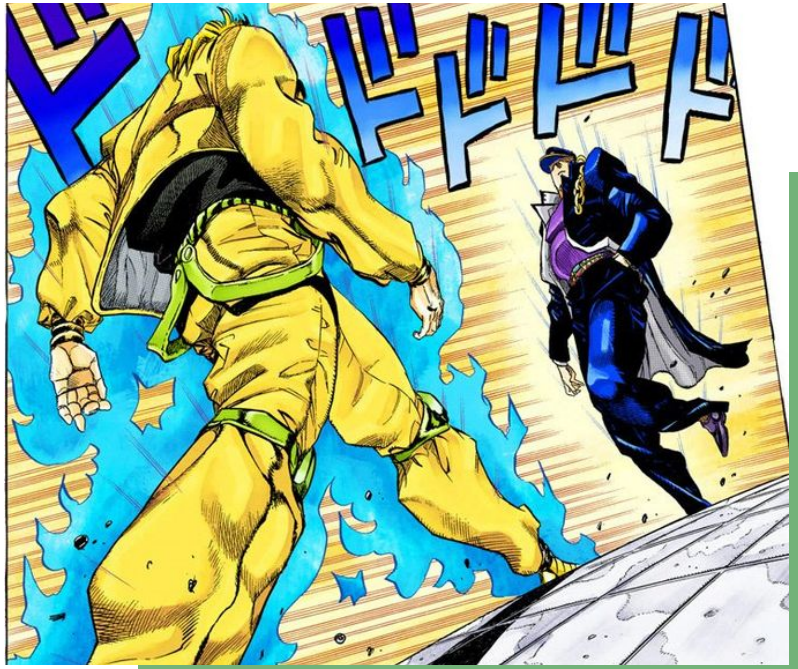
Week 9 lab

COGS 108, 9:00-9:50AM (B01)

https://github.com/alexavndra/cogs108-b01

Reminders!!

- EDA due TODAY at 11:59PM
  - SUBMIT ON GITHUB
- D8 is due Friday, December 1st at 11:59PM
- Office hours still running!
  - https://calendly.com/alexandrarh/office-hours



Evaluate me :O

Also...

LAST LAB SECTION TODAY**!!!!!

**I will be available for project help on Week 10 and 11

D8: UC Social Drinking (jk)

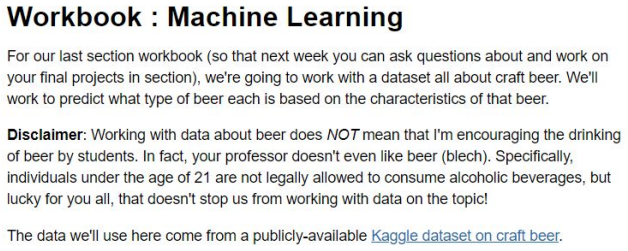
Lab Overview

We will be working on a dataset all about craft beer, to predict what type of beer each is based on the characteristics of that beer.

Part I: Data Wrangling
Read in the `'breweries.csv'` and `beers.csv` from the `data/` directory and clean up the data.

Part II: Prediction Model
Build the model with training and testing set by splitting the dataset. Use the `train_SVM` function to train your model.

Part III: Model Assessment:
Generate a classification report for the predictions generated for your training data relative to the truth value.
To check the performance of the model, generate a confusion matrix for the training data predictions as well as the ground truth.



Workbook : Machine Learning

For our last section workbook (so that next week you can ask questions about and work on your final projects in section), we're going to work with a dataset all about craft beer. We'll work to predict what type of beer each is based on the characteristics of that beer.

Disclaimer: Working with data about beer does *NOT* mean that I'm encouraging the drinking of beer by students. In fact, your professor doesn't even like beer (blech). Specifically, individuals under the age of 21 are not legally allowed to consume alcoholic beverages, but lucky for you all, that doesn't stop us from working with data on the topic!

The data we'll use here come from a publicly-available Kaggle dataset on craft beer.

Part I : Data & Wrangling

beers =beers.dropna(subset=['style','abv','ibu'])
beers.isnull().sum(axis = 0)

Definition and Usage

The `isnull()` method returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False.

Syntax

```
dataframe.isnull()
```

Parameters

This method takes no parameters.

Return Value

Syntax

```
dataframe.dropna(axis, how, thresh, subset, inplace)
```

Parameters

The `axis`, `how`, `thresh`, `subset`, `inplace`, parameters are keyword arguments.

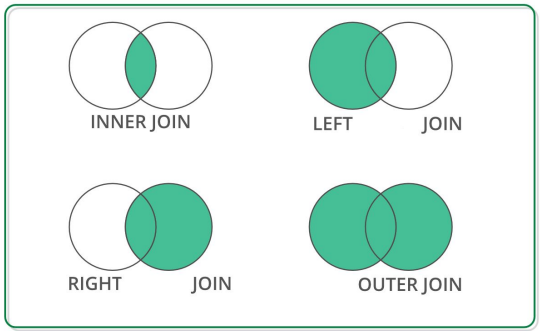
Parameter	Value	Description

axis	`0`
	`1`
	`'index'`
	`'columns'`
		Optional, default 0.
		0 and 'index'removes ROWS that contains NULL values
		1 and 'columns' removes COLUMNS that contains NULL values

how	`'all'` `'any'`	Optional, default 'any'. Specifies whether to remove the row or column when ALL values are NULL, or if ANY value is NULL.
thresh	*Number*	Optional, Specifies the number of NOT NULL values required to keep the row.
subset	*List*	Optional, specifies where to look for NULL values
inplace	`True` `False`	Optional, default False. If True: the removing is done on the current DataFrame. If False: returns a copy where the removing is done.

Part I : Data & Wrangling

```
df = pd.merge(beers,
              breweries, how="left")
```



Syntax

```
dataframe.merge(right, how, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
```

Parameters

All parameters except `right`, are keyword arguments.

Parameter	Value	Description
right		Required. A DataFrame, a Series to merge with
how	'left' 'right' 'outer' 'inner' 'cross'	Optional. Default 'inner'. Specifies how to merge
on	String List	Optional. Specifies in what level to do the merging
left_on	String List	Optional. Specifies in what level to do the merging on the DataFrame to the left
right_on	String List	Optional. Specifies in what level to do the merging on the DataFrame to the right

Call the answer "result"

We start with the user_usage dataframe. This is the "left" dataset in the merge

We are joining data from the user_device dataframe. This is the "right" dataset in th emerge.

```
In [13]: result = pd.merge(user_usage,
                          user_device[['use_id', 'platform', 'device']],
                          on='use_id')
         result.head()
```

The common column between the right (user_usage) and left (user_device) dataframes is "use_id".

We only want these three columns from the user_device dataset

Part II : Prediction Model

beer_clf = train_SVM(beer_train_X, beer_train_Y)

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, you can then feed some features to your classifier to see what the "predicted" class is.

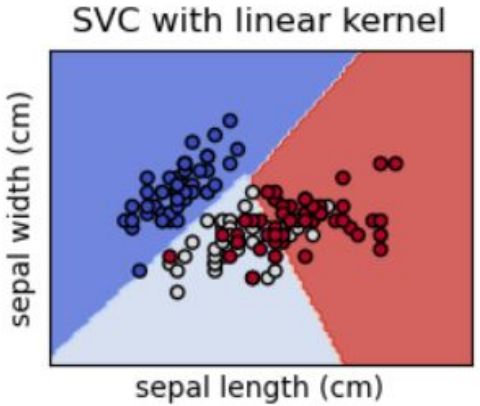
sklearn.svm.SVC

class **sklearn.svm.SVC**(*, *C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **LinearSVC** or **SGDClassifier** instead, possibly after a **Nystroem** transformer or other Kernel Approximation.

The multiclass support is handled according to a one-vs-one scheme.



Ref: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Part III : Model Assessment

```
class_report_train =
classification_report(train_Y,
predicted_train_Y)
```

classification_report() function from the **sklearn** library generates the following matrices:

1. Precision: Percentage of correct positive predictions relative to total positive predictions.

2. Recall: Percentage of correct positive predictions relative to total actual positives.

3. F1 Score: A weighted harmonic mean of precision and recall. The closer to 1, the better the model.

- F1 Score: 2 * (Precision * Recall) / (Precision + Recall)

sklearn.metrics.classification_report

`sklearn.metrics.classification_report`(*y_true, y_pred, *, labels=None, target_names=None, sample_weight=None, digits=2, output_dict=False, zero_division='warn'*) [source]

Build a text report showing the main classification metrics.

Read more in the User Guide.

Parameters:

y_true : ***1d array-like, or label indicator array / sparse matrix***
Ground truth (correct) target values.

y_pred : ***1d array-like, or label indicator array / sparse matrix***
Estimated targets as returned by a classifier.

labels : ***array-like of shape (n_labels,), default=None***
Optional list of label indices to include in the report.

target_names : ***array-like of shape (n_labels,), default=None***
Optional display names matching the labels (same order).

sample_weight : ***array-like of shape (n_samples,), default=None***
Sample weights.

digits : ***int, default=2***
Number of digits for formatting output floating point values. When `output_dict` is `True`, this will be ignored and the returned values will not be rounded.

Ref: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Part III : Model Assessment

conf_mat_train = confusion_matrix(train_Y, predicted_train_Y, sample_weight=None)

Confusion Matrix summarizes the predicted and actual values of a classification model to identify misclassifications, by computing :

- True positives (TP)
- False positives (FP)
- True negatives (TN)
- False negatives (FN)

sklearn.metrics.confusion_matrix

`sklearn.metrics.confusion_matrix(y_true, y_pred, *, labels=None, sample_weight=None, normalize=None)` [source]

Compute confusion matrix to evaluate the accuracy of a classification.

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

Read more in the User Guide.

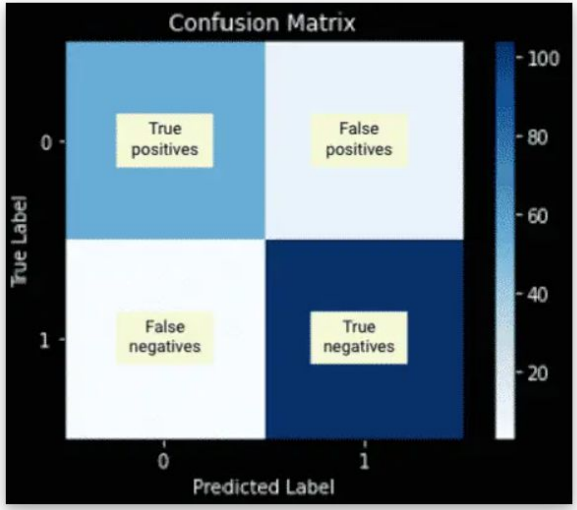
Parameters:

y_true : *array-like of shape (n_samples,)*
Ground truth (correct) target values.

y_pred : *array-like of shape (n_samples,)*
Estimated targets as returned by a classifier.

labels : *array-like of shape (n_classes), default=None*
List of labels to index the matrix. This may be used to reorder or select a subset of labels. If `None` is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order.

sample_weight : *array-like of shape (n_samples,), default=None*
Sample weights.



Ref: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html