## Introduction to Machine Learning

**High Level data science process**

* “big data” coined from 1997. Coined from CCC big data pipeline from 2012
* Data mining are in an analysis step
* Steps are (for the big data pipeline)
  + Acquisition
  + Extraction/cleaning (get rid of noise to something usable)
  + Aggregation (Directly condsive for datamining)
  + Analysis/modeling
  + Interpretation
* KDD process (Knowledge discovery in databases) is the same steps as above(which is from 1996) which wasn’t cited from the CCC paper
* Data mining is only a part of the process
* CRSIP-DM (Cross industry standard process for data mining)
  + Made a 77 page manual
  + Includes business and data understanding
  + Lots of iteration
  + Includes a business understanding step and deployment for business purposes
* Stages are similar for all steps.
* Processes is iterative process

Ex. Predict power failures in NYC

* + How to prioritize manhole inspection to prevent manhole events
* In CRISP-DM
  + Opportunity assessment and business understanding
    - What do you want to accomplish? Risks? How to evaluate results?
    - Goals to predict accuracy of manhole events 1yr in events, and create cost benefit analysis for inspection policies that account the cost of inspections
  + Data understanding and acquisition
    - Data given is trouble tickets by dispatchers, also has information about manholes and cable data (not matched with manholes)
    - Also had inspection reports and vented cover data
  + Data cleaning and transformation
    - 99% of the work? Lol
    - Had to turn freetext into structured information
  + Model building
    - Predictive modeling (is ml or stats modeling)
    - If the goal is to answer Y/N then this is a classification problem
    - Numerical value is a regression problem
    - If want to group into similar groups then its clustering problem
    - If want to recommend something then it’s a recommender system
    - There are many other potential problems that can be made
  + Policy construction
    - How should model be used the change policy
    - This is a prescriptive problem not a predictive
  + Evaluation, residuals and metrics
    - Blind test to make sure model works
  + Deployment
    - Turing the project into being implemented
    - Will need to update and improve the thing
* Four Vs of big data
  + Velocity, variety, volume, veracity (trustworthiness)
* Keep a realistic timeline, then add a few months to it

**Overview of Machine Learning**

* Basics of ML
  + Train a model to produce a useful response to the input
  + Response is typically a prediction of what we want to do with the data
* For f(x) = y, when training we know what y is. Y is called the labels
* Supervised machine learning is when someone has marked what results Y which is used to train the model
* Unsupervised learning only relies on feature values
* Ml isn’t a linear thing to follow, its exploration process
* Need to explore and prepare the data (cleaning, feature engineering or splitting dataset)
* As well need to construct and evaluate ML model. But evaluation may not show performance we want
* KNN algorithim
  + Does not require training, but needs to know lables
  + Outlier of other ML algorithims
* New case is defined and then we find the nearest neighbors to the new case which is how you classify or predict the new case
  + Need distance measure between cases.
  + Can use Euclidian or weighted distances (it depends on the # of variables/features to make prediction)
  + K = number of neighbors to use to make the prediction. The result is done by a majority vote
* KNN can be effective but suffers from curse of dimensionality
  + To get the same density of a 1d case, each additional feature/dimension requires an additional rasiing of the power for the number of samples present in the 1D case
    - Eg if 1d requires 10 samples, 2dd requires 100, 3d requires 1000, 4d 10^4 ect.
* Demo
  + Class = # of groups we are trying to break the data into
  + May need to worry about class imbalances.
  + Need to ensure data is properly prepared in the dataset.
  + Note that in the lab these values were scaled, which menas is mean, demeaned and variance scaled data. No bias in model training due to changing numerical range
    - Scale the numeric values of the features. It is important that numeric features used to train machine learning models have a similar range of values. Otherwise, features which happen to have large numeric values may dominate model training, even if other features with smaller numeric values are more informative. In this case Zscore normalization is used. This normalization process scales each feature so that the mean is 0 and the variance is 1.0
  + Note scikit learn can only deal with np arrays numeric
  + Don’t want bias in evaluation for the model so need to split the model
    - Has train\_test\_split, which can create training set of features, labels and test set of features and labels
  + Have to use np.ravel since we need a 1d numpy array
  + For the learning import KNeighborsCalssifier
  + Need to do the following when training
    - First step is to define *a model object*
      * Where you give arguments or hyperparameters of the model object
    - Model is then fit with a fit method, which you also feed the numpy array for training features and labels
    - Output (if not assigned) gives information about the metrics used
  + Test the data
    - Add a dataframe with test features and then add a column of predicted
      * Predict method from the model which should predict what the scores should be based on the model training.
      * Then only need to determine is if that is correct or not
    - Then you can use zip to compute if there is a match between the predicted and the labels
    - Will later go into detail for evaluating models and then can compute accuracy
  + Knn does not really have a formal training step as its quite simple mathematically
  + Note the following inline code (For reference)
    - iris\_test['correct'] = [1 if x == z else 0 for x, z in zip(iris\_test['predicted'], iris\_test\_labels)] #Thats intresting inline

## Exploring Data

**Exploring data for regression**

* Need to understand the content and relationships with data in the datasets
* More time to explore data, waste less time to build datasets
* What will help with the label and what will just be noise? Want to avoid problems
* What do you want to look for
  + General characteristics of the data (size, data types)
  + Summary statistics (means, quantiles)
  + Want to look for errors and outliers
  + For numeric features, distributions of features and label
  + For categorical, frequencies of data
  + Also want to understand relationship between similar correlated features
* Process is iterative process
  + Once you correct errors, need to visualize data
  + Expect to repeat different steps
* Guidance is to try a lot of ideas and have some work
* Visualizations
  + 1- have distribution and frequency tables
  + 2d – can have combinations of numeric and categorical variables.
* Problem with visualizations, there are only 2d computer screens where you can visualize datas
* Aesthetics for visualization
  + Shape, Size, color, Position and length between objects
  + Some of these items can be used for numeric and categorical variables
* Can use arrays of multiple axis.
  + Can use multi axis plots,
  + Or can do conditioned plots. Array of plots of certain dimensions, but just doing a group by operation to show a subset of the plots
* Visualizing distribution/frequencies
  + Bar chart
    - Example
    - For each column, define the plot area, look at leach axis (gca)
    - Then compute a pd series and apply the method wante
    - Apply .plot.ar to plot the counts
    - Then give titles and show the plot
    - Plotting is from Pandas
    - Bar charts are good for two categorical variables but can be done for a lot more
  + Histogram
    - Numerical values are binned into each different bins
    - Make sure to note if plots have any skews or outliers in the data
    - Tool to look at distribtions of numeric data
  + Kernal density estimate plots
    - Use sns package
    - Create a grid on the sns package to create a histogram to include the rug
    - Information is the same but the view is completely different
    - Rug shows frequency of occurrence in the cases
    - Note that the vertical axis is a normalized axis, where sum of area below is 1.
  + Combine histogram and kernel density estimate
    - Can combine both together to see different things on the same view
* Visualizing data relationships
  + Looking at two numeric variables and their relationship
  + Main tool to look at two numeric variables is a scatterplot
  + Can do .plot.scatter in pandas to get the plots from scatters
  + Problem with overplotting -> too many points within a certain narrow range on the plot. Cant tell density within the plots
  + To deal with overplotting, can set a transparency parameter so you can see all the points
  + For Very large samples, can do a kernel estimate density plot in sns for 2d version
    - Essentially returns a contour plot, along with marginal distributions
  + Can also do a hex bin plot
    - Like a 2d equilvanent of a histogram. Less computational intensive than a contour plot
    - Marginal densities show in histograms, and each of the bins are separated into hexagons.
    - In contour there are some lost outliers, but in hex plot there are all shown in some degree
* Visualizing categorical relationships
  + For if one of the variables is categorical
  + Boxplots
    - Way to summarize the distribution of a numeric variable that can be arranged by category
    - Use sns to get the boxplot method
    - Box contains the range of the 25-75% quartile of the data, where the dark line is the 50% quartile
    - The whiskers are the long as the most extreme value of 1.5 time the inner quartile range for whatever the lowest value is
    - Outliers are shown as points outside of the normal range
    - Gives a quick view for overlap between two distributions
  + Violin Plot
    - It’s a kernel density estimation plot.
    - It’s a boxplot within the violins. Just show the box and whiskers inside the violin
    - Note that the area under the violin curve is the same. So if want to lump due to feature engineering, its possible to see this using boxplots
* Using aesthetics to visualize higher dimensions
  + Problem with the plots previously described is that all the plots are in two dimensions
  + Will look at plot aesthetics to add dimensions to plots
  + Shape
    - Pick distinctive shapes so people can see them for a small number
    - Loop over each unique category to plot a temporary dataframe
    - Can also pass values to matplotlib in scatter\_kws
  + Marker size
    - Use a numeric or categorical variable with natural order.
    - Need to make sure to scale size so it’s not overlapping.
  + Color
    - Another addable dimension to the plot.
* Visualizing high dimensional relationships
  + Look at different ways to use a multi axis plot. Don’t need to just have two spatial dimensions
  + Can use pairwise scatterplot (sns.pairplot)
    - Can also put two d plots on the matrix
    - Note the plots will be in matrix form
    - Vertical axis and horizontal axis are the 5 variables. Every possible pairwise scatterplot, but on the diagonals you get two kde plots
  + Another type of plot is a conditional plot
    - Create a sns.FacetGrid with a grid column
    - Bascially a group by operation that gives histograms (due to plot methods)
    - Three histograms for direct comparisions
    - Can directly compare the differences for where the pileups occur at certain locations
    - Called conditioned variable because its conditioned on a specific variable.
    - Can also build a facet grid on 2 dimensions

**Exploring Data analysis for classification**

* Classification problems, want to classify the label
* Want to see if Features are distinct for label categories
* Can do boxplot of each feature for each variable to see the relationship for classification
* Can also do bar charts for categorical data (aka counts)
  + Essentially look for differences between the categorical data values
* Want to know how frequent categories show up to see if they are significant in the model
* Also want to know how balanced the categories of labels are
* Looking for frequencies tables
  + Basically, just how frequent an factor in a category occurs in the data
* Imbalances are significant in statistical modeling depending on counts
* If there is a different between two categories of data in a classifier, then it’s a class imbalance which can easily affect the model

## Cleaning and Preparing Data

Data Preparation and Cleaning

* Data prep is important in ML pipeline
* Want to make sure that ML algorithms work in the optimal way and data prep is needed for good ML performance
* Good preparation allows simple machine learning algorithms to work well.
* If you throw in data that hasn’t been well prepared, will get poor results even at best algos
* Prep steps
  + Explore and understand data problems
  + Remove duplicates
  + Treat missing value
  + Treat outliers and errors
  + Scale features
  + Split dataset
  + Visualization to check results
* Duplicates
  + Show up in databases accidently.
  + Don’t want to overweight to bias ML models
  + Identify duplicates by some sort of Unique ID, or may need to do it by value
  + Also need removal strategy
    - Can keep most recent data/oldest depending on type
    - Or can just keep first/last
* Missing Values
  + Can be detected in data exploration
  + Algos can fail on Missing Values
  + Need to figure out how missing values are coded,
    - Sometimes coded in odd ways such as -9999, 0, “NA”, “?”, ect.
  + Treatment strategy
    - Won’t have much info unless you can fix the missing value, usually safe to remove them than to introduce noise
    - May want to delete, or you can forwards/backwards fill
    - Can also impute missing values or interpolation
* Errors and Outliers
  + Can just get wrong values identified in exploration (errors)
  + Outliers can be very useful in information (so will want to keep them!)
  + To identify cases
    - Use summary Statistics
    - Or can use visualizations.
  + Treatment strategies
    - Just set a range of credible value range
    - Can treat as the same as missing values
* Scaling
  + Improper scaling biases machine learning
  + Don’t want feature with large numeric range to dominate the training
  + There will be large skew with model parameters
  + Common method is Z score
    - Standardization, mean = 0 , stdev = 1
    - Good for approx. normal distribution
  + Min-max scailing
    - To a transformation of range of values from 0 to 1.
      * Fails with outliers cause skew the ranges
  + Treat errors and missing values before scaling.
* Splitting
  + Split the data before training of supervised learning
  + Want independent (statistically) training and evaluation subsets
    - If evaluating and testing model on training data then its called information leakage since model is getting more accurate at the data provided
    - May not fit general case in production
  + Common method to split is Bernoulli sampling for random sampling.
  + Other method is Cross validation, which is a resampling method.
* Need to prep data for success in ML. Problem may be found after training ML model.
  + 80% with data prep, 20%to deal with data prep problems
* Practical approach to dealing with missing values
  + Can use .any() method, which makes a table for missing values for Any column
  + Maybe because of missing values, characters may have been changed to character strings when should be numeric
  + Inplace prevents using extra memory in pandas
* Practical Treating duplicates
  + Biases ML training, so good to track down
  + .shape (rows and column #) can be compared to the .unique values of he shape
  + Pandas can just .drop\_duplicates by a subset argument
* Practical Scaling Data
  + Can be done with tools in scikit learn
  + Need to split the dataset, since you want ot use the scale of training data to be applied to training features as well.
    - Use train\_test\_split to split the data
  + Only scale the training numpy arrays
  + Can use preprocessing.StandardScaler().fit() [fit fits to the training data]
  + Make sure to use the scaler on the test data
  + Its quite easy scale, but need to scale on training data then apply the scale to evaluation data

**Feature Engineering**

* Good features that are highly predictive of labels we are trying to predict, will get good performance
* Want to develop highly predictive features
* Raw features presented are not always the best
  + May not be the best way to view them
* Good features, allow for simple machine learning algorithms to work well
  + Poor features can result in poor performance even with best algos
* Certain steps will be repead
  + Explore understand data relationships
  + Transform features (single feature by math process)
  + Interaction terms through math on other features
  + Visualizations to check results
  + Test with machine learning model
* Transforming Features
  + Want to improve distribution properties that may be skewed to be more normal
  + May also have the feature covariate more with the label
    - Relationship may be highly nonlinear which may have low correlation but still highly covariate in transformed space
  + Common transformations
    - Log, square and square roots, variances, means ect
    - Could also do differences between features, cumulative sums (common w/ time ordered data)
    - All transformations are nonlinear. So they are not going to have high correlation with previous features.
      * Don’t need to introduce new colinearlity
* Interaction terms
  + When two or more features that interact with each other, interaction term is define d when its one variable by another variable
  + Many way to compute interaction terms
    - Mean, median, ect.
* Finding predicte features is the key sto success
  + If can transform or engineer more predictive features then the process is going to be better
  + Iterative process
  + Can use data exploration to id these opperunties and to test lots of ideas.
* Feature Engineering Examples
  + Aggregating Categories
    - Can build a dictionary of new variables, so we can map list to different categories.
    - Want to aggregate categories that are within the domain you are working in
    - Successful if you go from meaningless categories that are well separated
  + Transforming Variables
    - Can just use a math method for the process, can you can see the distribution to be a bit more symmetric. Basic kind of transformation for a simple feature engineering method

## Getting Started With Supervised Learning

**Regression**

* Simple linear regression (one feature)
* Multiple linear regression (multiple features)
* Regression is for predicting real value outcomes
  + Eg income, sales figures
* To need to be able to measure deviation (squared as error for deviation)
* Sum of squares error is fundamental quality in regression
* Simple linear regression
  + Only one feature to try and estimate the model
* Goal is to minimize sum of squares error, so we choose coeffiecnets to minimize the sum of squares
  + Goal of linear regression
* Multiple Linear Regression
  + Just weighted combination of factors
  + Same as simple regression but can have multiple terms
  + Can use polynomials for variables
  + But too many can run into dimensionality issues, but can include all potentially important factors
* Basics of scikit learn ML model
  + Label array in numpy array
  + Then get an index to use train\_test\_split
  + Create x and y train and test which is training/testing labels
    - Need to use ravel for the right dimensions for labeled numpy array
  + Cant use Numpy arrays for scikit must use numpy array
  + Then you can scale with a scaler, and fit it to training eature
  + Then you build a linear regression model
    - Linear\_model.LinearRegression
    - Defines scikit learn model object
    - Can then apply the fit method to the training features
  + Linear and coefficients can be print by intercept\_ and coef\_
* Evaluation of regression models
  + Can just use SSE or mean square error (MSE).
  + Residuals is the difference between prediction and the truth
    - Don’t want to see pattern in residuals. Ideally all close to zero
* Demo
  + Get all from scikit learn metrics package
  + Can also make a QQ plot with some of the packages (which should be In the code)
* Have to use numpy arrays (numeric) for scikit learn
  + So how to get to numeric numpy array to use with scikit learn
  + Need to create dummy variables from categorical variables
  + Using preprocessing can encode a LabelEncoder()
  + Then can fit the encoded labels to the features which is a pandas series
  + Then you need to make the dummy variables for 0 or 1 if the value appears
  + Can use OneHotEncoder object and fit it to the features.
  + Essentially getting a sparse non-invertible design matrix

**Classification**

* ML is to teach computers by example
* We don’t tell computers what we want to look at, we just give it a lot of examples. Very close to ML
* Classification problem, where you give a training set of observations and a test set (to be able to make predictions)
  + Should be able to learn if there is not much noise
* Want as much data to train these models
* Each observation is represented by a set of numbers (vector)
* ML is to get the best weights to get the best predictors
* Formally, given a training set, we want to create a classification model to predict a label for a new observation
* Trying to create a function that divides the space so that on one side of the decision boundary there is an event
* Classification is mostly used for answers yes or no questions
* Loss (functions)
  + Measurement of classification error
  + Can use fraction of times the prediction is wrong, but the problem is that you can run into problems for minimization
  + If we get it right, we lose zero points, if we get it wrong then we lose one point
    - Called zero-one loss and it tells us if classifier is right or wrong
    - Is a kind of loss function
    - Problem as its not smooth
  + Note points are on a scale, further away from the decision boundary is very wrong/correct.
  + Use scale of y\*f(x) where y is the result of classification, x is the actual classification
    - F(x) is the misclassification error where it is +1 for one classifcation and -1 for incorrect classification
  + Other loss functions
    - can give large penalties for being very wrong but then less penalties for being more correct
    - three types of loss functions that will be covered: Logistic, adaboost which is exponential and a piecewise one from vector support machines
  + Goal is to choose a function for f that minimizes the average loss
    - Fraction of times sign of f(xi) is not yi = sum(y != sign f(xi))/n
    - <= sum(loss(yi f(xi))/n)
    - Minimization of models to get best model is about optimization
  + Want to predict well on data we haven’t seen before, not just low training error
* Stats theory for supervised learning (may need to review)
  + Note f(x) is the equation for regression
  + Ockhams razor, the best models are the simplest models
  + Don’t want to overfit the models
  + More complex models, more tend to overfit
    - Plot of training error, as more complex more models, training error decreaseas
    - Test error will also increase
    - If under fit then training and test will still have error
  + Need balance between accuracy and simplicity
  + Common ML methods choose function f to minimize error and complexity
    - Curse of dimensionality, overfit but not enough data
    - Want to choose a low complexity and low training error
  + Simplicity is called regularization and has many functions
* Logistic regression
  + Uses log(1+exp(-yf(x)))
  + Just minimizes the average loss
  + F(x) can just be a regression model
  + Logistic regression tries to find weights to minimize the sum of training losses
* MLE perspective
  + Can model the outcome Y for a specific x and beta using the logistic function, which only produces probabilities
  + Basically using plogit to make things into a probability to maximize the probability with the log likelihood
  + Essentially whatever the score is from the model, getting the probability that y=1 (it’s a fit!)
* Can use sum of squares for as the regularization coefficient, aka measuring simplicity of model
  + L2 is sum of squares, l1 is the absolute values
* Evaluating Classifiers,
  + Features (x), labels (y) which are +-1 and ml algo will give a number to each observation for how far it thinks its from decision boundary f(x)
    - Y hat gives the predicted label from the algorithm f(x), which is just +-1
    - If good, then true label will be same as predicted label
    - Type 1 (false positive) y = -1, yhat = 1
    - Type 2 (False negative) y = 1, yhat = -1
  + Quality of classifier
    - Construct confusion matrix which is of +-1 for yhat and y. if its good then +1 or -1 for both has a lot of observations, not too many errors
  + (Mis)classification error, fraction of points where the predicted label is not equal to true label
  + True positive Rate (TPR) is # of( points y = 1 and yhat = 1 ) / # of (y = 1)
    - Sensitivity/recall
  + True negative rate (TNR) is # of( points y = -1 and yhat = -1 ) / # of (y = -1)
  + False positive rate is # of( points y = -1 and yhat = 1 ) / # of (y = -1)
  + Precision is # of( points y = 1 and yhat = 1 ) / # of (yhat = 1)
    - True positive over the number of predicted positives
  + F1 score, is 2\*Precision\*recall / (precision\*recall)
    - Good f1 score then usually means good model
    - Very good in information retrevial
  + Precision@N is the # of true positives from N / # of predicted positives at N
  + Recall@N is the # of true positives from N / # of positives predicted (?)
  + Often use Misclassification error to compare algos, but only when errors in positive = errors in negatives
  + For information retrieval, want to use Precision and recall and F1. For doctors, may want to look at TPR and TNR (Sensitivity and Specificity)
* ROC Curves
  + For a particular false positive rate, what is the True positive rate?
  + Idea is to record the true and false positive rates based on the moving decision boundary for results
  + Quality used is area under curve (AUC) or AUROC
  + You want to be better than a f(x) = x line since that is the results you expect when randomly guessing
* ROC Curve Algorithms
  + Can be produced which uses a single real valued classifier to evaluate the classifier
    - For a classifier f(x), can place the decision boundary anywhere we want, and we record the true and false positive rate, and plot the rates as we do it
  + Can also use a single algorithm and sweep the imbalance parameter across the full range to trace out an ROC curve to evaluate the algorithm
    - C parameter to allow different weights for positives and negatives
    - You sweep through values of C parameter, and adjust the ML model each time C is adjusted. Only cares about getting the negatives right.
    - Get True/false positive rates each time
    - Evaluates the algorithm
  + Algorithm optimized for a specific decision point can do better than an algorithm optimized for something else (single classifier)
* Classification Models
  + How to preform classification through scikit-learn
  + Always split the dataset before training and testing and then scale the data values
  + There is logisticRegression in scikit learn, then can fit it on variables
  + Can also get intercept and coefficents. Gives linear regression model values
  + To get the log-likelihoods, use predict\_proba(\*testfeature\*) to predict the probabilities of the test features
    - Can then use the probabilities to see where they lie across the threshold
  + Basics of going from logistic regression, to probabilities then assigning threshold to probabilities to come up with label scores
* Classifier Evaluation demo
  + Scikit learn metrics package
  + Can use precision\_recall\_fscore\_support to get the precision
  + Can also get confusion matrix with confustion\_matrix
  + Also an accuracy method with accuracy\_score
  + There is roc\_curve method, you need to give the probabilities to get the AUC
  + Don’t just look at one metric, look at multiple metrics to see better classifers
* Imbalanced data
  + Sometimes in imbalanced data, true positive may not be the same as true negative
  + Many ways to get this done
  + To weight positives more important than negative functions, can just break the sums for the loss function and to weight the classification over the other
  + All Ml algos allows for adjustment for C
    - Works better for some algorithms (eg Decision Trees, as can swing rapidly)
  + Do not report plain accuracy for imbalanced data. Report the True positive rate and false positive rate. Accuracy is meaningless
  + Can adjust C to get ideal balance between true and false positive rate.
    - Can also use C parameter to evaluate the quality of a whole algorithm
* Approaches for addressing imbalanced data
  + Goal is to find approaches to limit imbalance bias in ML training
  + May also have a bias to get majority case correct when you want minority case
  + Approach is to change sampling or weight of dataset to create a balance
  + May have to test to see what works for your situation
  + Cases
    - Under sample majority case – but you lose a lot of training data, may give less biased result? You random sample majority case to find what you keep. Set p to get desired number of cases back
    - If can’t throw away majority case, might want to oversample minority case (aka giving more weight to the data). Bernouilli sample the minority case and limit bias.
    - Case weights (eg linear models, support vector machine). You set weights inversely proportional to frequency of classes. Weights add up to one, but overweight the minority cases so there is no bias with majority case
    - May also be able to use Synthetic minority over sampling (SMOTE) which may work but they may also fail
  + Imbalanced cases will bias the ml training and often will bias in the wrong way.