**Machine Learning**

Algorithms that can learn from observational data and make predictions based on it

Sci-kit learn requires everything to be numerical

**REGRESSION – PREDICT CONTINUOUS NUMBER**

Is it linear or non-linear?

* Linear: Simple or Multiple Linear Regression
* Non-linear:
  + Polynomial Regression, SVR, Decision Tree or Random Forest
  + k-Fold Cross Validation and then pick model with the best results

1. **Simple Linear Regression - - LINEAR**
   1. **Pros:** Works on any size of data set, gives information about relevance of features
   2. **Cons**: Linear regression assumptions
   3. Fit straight line to dataset of observations and use this line to predict unobserved values
      1. Usually using ‘least squares’
         1. Minimizes the squared-error between each point and the line
            1. Error is just the distance between each point and the line
            2. Sum up all those squared errors
            3. Measuring the variance of the data points from that line
            4. By minimizing the variance, find the line that fits best
         2. Sometimes called ‘Maximum likelihood estimation’
      2. Gradient Descent is an alternate method to least squares
         1. Works best with 3D data
         2. Iterates to find line that best follows the contours defined by the data
         3. Easy to try in python and compare to least squares (usually least squares is a perfectly good choice)
   4. R-squared (coefficient of determination)
      1. % of the total variation in dependent variable y is captured by the model
      2. Ranges from 0 to 1
2. **Multiple Linear Regression - -LINEAR**
   1. **Pros:** Works on any size of data set, gives information about relevance of features
   2. **Cons:** Linear regression assumptions
   3. Still uses least squares
   4. Can still measure fit with r-squared
   5. Create coefficients in front of numerical values that represent each feature
      1. Can’t really use categorical data (ordinal data you can)
   6. Normalize features so you can compare coefficients in a meaningful way
3. **Polynomial Regression - - NON LINEAR\***
   1. **Pros:** Works on any size of dataset, works very well on non-linear problems
   2. **Cons**: Need to choose the right polynomial degree for a good bias/variance tradeoff
   3. Linear formula y = mx + b is a first degree polynomial
      1. Higher orders produce more complex curves
   4. Can still measure fit with r-squared
4. **Support Vector Regression- - NON LINEAR**
   1. **Pros:** Easily adaptable, works very well on non-linear problems, not biased by outliers
   2. **Cons:** Compulsory to apply feature scaling, not well known, more difficult to understand
5. **Decision Tree Regression- - NON LINEAR**
   1. **Pros:** Interpretability, no need for feature scaling, works on linear / nonlinear problems
   2. **Cons:** Poor results on too small datasets, overfitting can easily occur
6. **Random Forest Regression- - NON LINEAR**
   1. **Pros:** Powerful and accurate, good performance on many problems, incl. non linear
   2. **Cons:** No interpretability, overfitting can easily occur, need to choose number of trees

**CLASSIFICATION – PREDICT CATEGORY (SUPERVISED)**

Independent variables x & dependent variable y

Is it linear or non-linear?

1. **Logistic Regression - - LINEAR**
   1. **Pros:** Probabilistic approach, gives information about statistical significance of features
   2. **Cons:** The Logistic Regression Assumptions
   3. Use Logistic Regression (or Naive Bayes if problem is non-linear) when you want to rank your predictions by their probability. For example if you want to rank your customers from the highest probability that they buy a certain product, to the lowest probability. Eventually that allows you to target your marketing campaigns. And of course for this type of business problem, you should use Logistic Regression if your problem is linear
2. **Support Vector Machine (SVM) -- LINEAR**
   1. **Pros:** Performant, not biased by outliers, not sensitive to overfitting
   2. **Cons:** Not appropriate for non linear problems, not the best choice for large # of features
   3. SVM when you want to predict to which segment your customers belong to. Segments can be any kind of segments, for example some market segments you identified earlier with clustering.
   4. Need to scale features
   5. Works well for classifying higher dimensional data (lots of features)
   6. Use SVC to classify data using SVM
   7. Can use different ‘kernels’ (linear, rbf, polynomial) . some will work better than others for a given data set
3. **K-Nearest Neighbors (K-NN) – NON LINEAR**
   1. **Pros:** Simple to understand, fast and efficient
   2. **Cons:** Need to choose the number of neighbours k
   3. Used to classify new data points based on ‘distance’ to known data on a scatter plot
      1. Find K nearest neighbors, based on your distance metric
      2. Let them all vote on the classification
4. **Kernel SVM – NON LINEAR**
   1. **Pros:** High performance on nonlinear problems, not biased by outliers, not sensitive to overfitting
   2. **Cons:** Not the best choice for large number of features, more complex
   3. Need to scale features
5. **Naive Bayes – NON LINEAR**
   1. **Pros:** Efficient, not biased by outliers, works on nonlinear problems, probabilistic approach
   2. **Cons:** Based on the assumption that features have same statistical relevance
   3. Use Naive Bayes (if problem is non-linear) when you want to rank your predictions by their probability. For example if you want to rank your customers from the highest probability that they buy a certain product, to the lowest probability. Eventually that allows you to target your marketing campaigns. And of course for this type of business problem, Naive Bayes if your problem is non linear (Logistic Regression if your problem is linear).
6. **Decision Tree Classification – NON LINEAR**
   1. **Pros:** Interpretability, no need for feature scaling, works on both linear / nonlinear problems
   2. **Cons:** Poor results on too small datasets, overfitting can easily occur
   3. Use when you want to have clear interpretation of your model results
   4. Scaling optional
   5. Susceptible to overfitting
7. **Random Forest Classification – NON LINEAR**
   1. **Pros:** Powerful and accurate, good performance on many problems, including non linear
   2. **Cons:** No interpretability, overfitting can easily occur, need to choose the number of trees
   3. Use when you are just looking for high performance with less need for interpretation.
   4. Construct several alternate decision trees and let them “vote” on the final classification
      1. Randomly re-sample the input data for each tree
         1. Called “bootstrap aggregating” or “bagging”
      2. Randomize a subset of the attributes each step is allowed to choose from

**CLUSTERING - - PREDICT CATEGORY (UNSUPERVISED)**

Independent variables only

1. **K-Means**
   1. **Pros:** Simple to understand, easily adaptable, works well on small or large datasets, fast, efficient and performant
   2. **Cons:** Need to choose the number of clusters (Elbow Method)
   3. **Process**
      1. Randomly pick K centroids (k-means)
      2. Assign each data point to the centroid its closest to
      3. Re-compute the centroids based on the average position of each of it’s points
      4. Iterate until points stop changing assignments to centroids
      5. If you want to predict the cluster for new points, just find the centroid they’re closest to
   4. Does not attempt to assign any meaning to the clusters you find, it’s up to you to determine that!
   5. Must scale data to normalize it (important for good results)
2. **Hierarchical Clustering**
   1. **Pros:** The optimal number of clusters can be obtained by the model itself, practical visualization with the dendrogram
   2. **Cons:** Not appropriate for large datasets

**ASSOCIATION RULE LEARNING**

“People who bought also bought….”

1. **Apriori**
   1. **Support**
   2. **Confidence**
   3. **Lift**
2. **Eclat** 
   1. **Support Only**

**RECOMMENDER SYSTEMS**

**USER-BASED COLLABORATIVE FILTERING**

**Recommend items based on what people similar to you liked**

* Build a matrix of things each user bought/ viewed/rated (signal of interest)
* Compute similarity scores between users
* Find users similar t you
* Recommend stuff they bought/viewed/rated that you haven’t yet
* Problems with User-Based CF
  + People are fickle; tastes change
  + There are usually many more people than things
  + People do bad things

**ITEM BASED COLLABORATIVE FILTERING (AMAZON)**

**Looking at relationships between items**

* A movie will always be the same movie – it doesn’t change
* There are usually fewer things than people (less computation to do)
* Harder to game the system
  + Base behavior on what people actually bought
* Compute similarity of two movies based on the ratings of people who watched both movies
* Sort by movie, then by similarity strength

**REINFORCEMENT LEARNING (aka Online Learning)**

* Used to solve interacting problems where the data observed up to time t is considered to decide which action to take at time t + 1.
* It is also used for Artificial Intelligence when training machines to perform tasks such as walking.
* Desired outcomes provide the AI with reward, undesired with punishment.
* Machines learn through trial and error.
* The Multi-Armed Bandit Problem
  + 5 or more slot machines but need to find out how to bet to max your return
    - Because you don’t know which is the best
    - Max your return as you’re exploring until you find the best one and exploit that one
    - Combine exploration & exploitation & get to optimal results ASAP to max output of efforts
    - Ideal trade-off between exploration and exploitation
  + Exploit the best one while you’re exploring for it
  + Find out which is the best one in the quickest way possible and start exploiting it along the way (as you’re exploring)
  + Upper Confidence Bound Algorithm solves Multi-Armed Bandit Problem
    - Better than AB tests
    - Deterministic algorithm
    - Requires update at every round

Incorporate value right away to proceed to the next round

* + Thompson Sampling Algorithm
    - Probabilistic Algorithm
      * Can accommodate delayed feedback - Update in a batched manner
    - Better empirical evidence
* You have some sort of agent that “explores” some space
* As it goes, it learns the value of different state changes in different conditions
* Those values inform subsequent behavior of the agent
* Examples: Pac-Man
* Yields fast on-line performance on the space has been explored.
* Q Learning
  + A specific implementation of reinforcement learning
  + You have:
    - A set of environmental states s
    - A set of possible actions in those states
    - A value of each state/action Q
  + Start off with Q values ofo 0
  + Explore the space
  + As bad things happen after a given state/action, reduce its Q
  + As rewards happen after a given state/action, increase its Q
* Markov Decision Process
  + Provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker
  + Describes process above using mathematical notation
* You can make an intelligent Pac-man in a few steps
  + Have it semi-randomly explore different choices of movements (actions) given different conditions (states)
  + Keep track of the reward or penalty associated with each choice for a given state/action (Q)
  + Use those stored Q values to inform its future choices

**NATURAL LANGUAGE PROCESSING**

Applying Machine Learning models to text and language

* + Step 1: Bag of Words very known model
    - Used to preprocess the texts to classify before fitting the classification algorithms
  + Step 2: Fit classification algorithm
* Most NLP algorithms are classification models : Logistic Regression, Naïve Bayes, CART (which is a model based on decision trees), Maximum Entropy again related to Decision Trees, Hidden Markov Models which are models bae on Markov processes.
  + Two Approaches to model selection:
    - Could test each classification model and look at model performance criteria to decide best model
      * Classification Model Metrics
        + Accuracy = TP + TN / TP + TN + FP + FN
        + Precision (exactness) = TP / (TP + FP)
        + Recall (completeness) = TP/ (TP + FN)
        + F1 Score (completeness & recall compromise)= 2\*Precision \* Recall / (Precision + Recall)
    - Go with commonly used models for NLP: Naive Bayes, Decision Tree or Random Forest Classification

**DEEP LEARNING**

Mimic the human brain – neurons

* Input Values
  + independent variables of one observation (1 row in database)
  + need to standardize them (mean of zero and variable of one)
  + or sometimes normalize (subtract min value divide by range to get values between 0-1)
  + basically want all of these values to be similar to make it easier for neural network to process them
* Hidden Layers
* Output Value
  + Continuous (price)
  + Binary (will exit yes/no)
  + Categorical (several output values)
* Weights
  + How neural networks learn
  + Things that get adjusted through the process of learning
* The Activation Function
  + Threshold Function
    - Goes from zero to one
    - If value is less than or more than 0
    - Kinks in its curve
  + Sigmoid Function
    - Goes from zero to one
    - Function used in logistic regression
    - Smooth, gradual progression
    - Useful when trying to predict probability
  + Rectifier
    - One of the most used functions
  + Hyperbolic Tangent function
    - Similar to Sigmoid Function
    - Goes below zero
* How do Neural Networks Learn
  + Sum of squared differences between y-hat and y is back propagated through the neural network
  + Weights are adjusted accordingly
    - Brute Force: try all the different combinations
    - Gradient Descent
      * Convex (bowl shaped)
      * Get to the bottom by understanding which way to go
      * Which way is does it feel its going downward and take steps towards that
      * Descending to the minimum of the cost function
    - Batch Gradient Decent
      * Adjusting weights after you run all the rows in your neural network, iterate and re-run
      * Deterministic algorithm
        + As long as you have the same starting weights, every time you run it get the same iterations and results of the way your weights are being updated
    - Stochastic Gradient Descent
      * Stochastic = randomly determined
      * Doesn’t require cost function to be convex (bowl shaped)
      * Run one row at a time and adjust the weights then repeat
      * Doing one iteration one row at a time
      * Helps avoid problem of finding local minimums rather than the overall global minimum
      * Faster and lighter algorithm
      * Stochastic algorithm:
        + Picking rows in random manner
        + Even if you have same weights at the start, have different process/iterations

**Training the ANN with Stochastic Gradient Descent**

1. Randomly initialize the weights to small numbers close to 0 (but not 0)
2. Input the first observation of your dataset in the input layer, each feature in one input node
3. Forward Propagation
   1. From left to right, the neurons are activated in a way that the impact of each neuron’s activation is limited by the w3eights
   2. Propagate the activations until getting the predicted result y
4. Compare the predicted result to the actual result. Measure the generated error
5. Back-Propagation:
   1. From right to left, the error is back-propagated
   2. Update the weights according to how much they are responsible for the error
   3. The learning rate decides by how much we update the weights
6. Repeat Steps 1-5
   1. update the weights after each observation (Reinforcement Learning)
   2. update the weights only after a batch of observations (Batch Learning)
7. When the whole training set passed through the ANN, that makes an epoch. Redo more epochs

**ANN in Python**

* Encode categorical variables
  + Only if it’s 3 or more categories
  + Don’t need to for two categories because one will automatically be left out to avoid dummy trap
* Number of nodes in hidden layer
  + If data is linearly separable, don’t need a hidden layer or a neural network at all
  + Choose # of nodes in hidden layer as the # of nodes in input (# independent variables) & output layer/2
  + Or Parameter tuning
    - k – fold cross validation
      * creating set in dataset besides training set and test set “cross validation set”
      * experiment with different paramaters in model in cross validation set
* Initialize weights
  + ‘uniform’:
    - initialize weights randomly and
    - makes sure they are close to zero
* Activation function
  + Hidden layers: rectifier function ‘relu’
  + Output layer: sigmoid function
* Output layer
  + Want to have probabilities of the outcome
  + In order to get probabilities, have to replace activation function from rectifier to sigmoid function
  + Dependent later with more than 2 categories
    - Output\_dim = 3
    - Activation function = softmax (sigmoid function but applied to dependent variable with more than 2 categories)
* Compile

**CONVOLUTIONAL NEURAL NETWORKS**

Input Image > CNN > Output Label (Image Class)

1. Convolution
   1. Go through input image by column (stride) with Feature Detector box and look for matches
      1. Reduces input image but keep important features
   2. Input Image > Feature detector > Feature map
   3. Find features in your in your image using the feature detector and put them into a feature map
   4. ReLu Layer
      1. Apply Rectifier function
      2. Want to increase non-linearity in our CNN
         1. Images themselves are highly non-linear
2. Max Pooling (Down sampling)
   1. Feature Map > Max Pooling > Pooled Feature Map
   2. Take the max of each 4x4 box, stride of 2
   3. Preserve the features and account for possible spatial or textural distortions
      1. Introducing spatial invariance
   4. Also reducing the side
   5. Reducing number of parameters that are going into the final layers of CNN
      1. Preventing overfitting
   6. Less info, but same features
3. Flattening
   1. Flatten into column
   2. Take numbers row by row and put them into a single column
   3. Vector of inputs for CNN
   4. Input image > Convolution into Convolutional Layer > Pooling into Pooling Layer > Flattening into input layer of a future ANN
4. Full Connection
   1. Adding new ANN to CNN
      1. In CNN hidden layers are called Fully Connected Layer

Summary

* Input image to which we apply multiple different feature detectors (filters) to create feature map. This comprises our convolutional layer then on top of that convolutional layer we apply the ReLu to remove linearity In our images.
* Then apply pooling layer to convolutional layer to make sure we have spatial invariance in our images (can still pick up features even if its tilted) and reduce size and avoid over-fitting (max pooling)
* Flatten pooled images into one long vector (column of values) and input that into ANN
* Fully connected ANN all the features are processed through the network and we have this final fully connected layer which performs the voting towards the classes we’re after. All of this is trained through forward and back propogation with lots of iterations and epochs and in the end we have a very well defined NN.
* Not only are the weights trained, feature detectors are trained and adjusted in the same gradient descent process and that allows us to come up with the best feature maps. And in the end, we get a fully trained CNN which can recognize images and classify them.

**DIMENTIONALITY REDUCTION**

Remember in Part 3 - Classification, we worked with datasets composed of only two independent variables. We did for two reasons:

Because we needed two dimensions to visualize better how Machine Learning models worked (by plotting the prediction regions and the prediction boundary for each model).

Because whatever is the original number of our independent variables, we can often end up with two independent variables by applying an appropriate Dimensionality Reduction technique.

There are two types of Dimensionality Reduction techniques:

Feature Selection

Feature Extraction

Feature Selection techniques are Backward Elimination, Forward Selection, Bidirectional Elimination, Score Comparison and more. We covered these techniques in Part 2 - Regression.

In this part we will cover the following Feature Extraction techniques:

1. Principal Component Analysis (PCA)
   1. From the m independent variables of your dataset, PCA extracts p<= m new independent variables that explain the most variance of the dataset regardless of the dependent variable
   2. The fact that DV is not considered makes PCA an unsupervised model
   3. After data pre-processing and scaling but before you fit the model
   4. End up with 2 independent variables that are NEW as opposed to feature selection where we end up with 2 independent variables that are among the original independent variables
2. Linear Discriminant Analysis (LDA)
   1. From the n independent variables of the dataset, LDA extracts p <= n new independent variables that separate the most of the classes of the dependent variable
   2. Considers the dependent variable, makes LDA a supervised model
3. Kernel PCA
   1. Non-linear feature extraction model
4. Quadratic Discriminant Analysis (QDA)

Dimensionality reduction attempts to distill higher-dimensional data down to a smaller number of dimensions, while preserving as much of the variance in the data as possible.

* lets you distill multi-dimensional data down to fewer dimensions, selecting new dimensions that preserve variance in the data as best it can.
* Dimensions: features
* Explained variance ratio: how much of the variance in the original data was preserved as I reduced it down to two dimensions
* PCA has chosen the remaining two dimensions well enough that we've captured 92% of the variance in our data in a single dimension alone! The second dimension just gives us an additional 5%; altogether we've only really lost less than 3% of the variance in our data by projecting it down to two dimensions.

MODEL SELECTION

Consists of choosing best parameters of machine learning models

Two types of parameters

1. Parameters that the model learned
2. Hyper-parameters
   1. Parameters that we chose ourselves
   2. Grid Search
      1. Find optimal values of these parameters

HOW DO I KNOW WHICH MODEL TO CHOOOSE??

1. Look at your **dependent variable** to decide if it is a Regression, Classification, or Clustering Problem

* Continuous Outcome: Regression Problem
* Categorical Outcome: Classification Problem
* No dependent variable: Clustering Problem

2.Is my problem a linear problem or non-linear Problem?

* Grid-Search

XG BOOST

Most powerful implementation of gradient boosting in terms of model performance and execution speed

1. High performance
2. Fast execution speed
3. Keep interpretation of problem and model (no feature scaling)

K-Fold Cross Validation

* Variance problem: When you test the performance of your model on one test set, you get one accuracy but if you test it again on another test set you can get a very different accuracy
* So judging accuracy on one test set is not the most relevant way to evaluate model performance
* K-Fold Cross Validation: Splitting the training set into 10 folds (most of the time k = 10)
* Train model on 9 folds and test it on the last remaining fold
  + With each iteration can use different combinations of the 9 training folds and 1 test fold
  + Can train and test the model on 10 combinations of training and test sets
  + Take an average of the different accuracies of the 10 evaluations
  + and also compute the standard deviation to look at the variance to get a much better idea of model performance

**Idea is train and evaluate model using different train/test split combinations (Cross-validation) within the training data, tune hyper-parameters accordingly until you get an acceptable mean accuracy. Then apply the very best model on test set to see how it performs on ‘general’ data**

**SIMPLE K-FOLDS CROSS VALIDATION**

Instead of evaluating model on one train/test spit, evaluate model on multiple train/test split combinations and take the overall mean accuracy.

Pass in model and all of your data

Run model 5 times using different 5 train/test split combinations and returns accuracy for each individual result (Fold)

average accuracies together to get an overall error metric

Evaluate our model against the entire dataset spit up 5 different ways and give us back the individual results

In practice, you need to try different variations of your model and measure the mean accuracy using K-Fold Cross Validation until you find a sweet spot

**ENTROPHY**

* A measure of a dataset’s disorder – how same or different it is
* If we classify a dataset into N different classes (ex: a data set of animal attributes and their species)
  + 0 Entropy: all the classes in the data are the same (everyone is an iguana)
  + High Entropy: They’re all different classes

**ENSEMBLE LEARNING**

Using different models to try to solve the same problem and let them vote on the results (Random Forests)

Ensembling techniques take a number of weak learners (classifiers/regressors that are barely better than guessing) combine them (through averaging or max vote) to create a strong learner that can make accurate predictions

* 1. Bagging (Bootstrap aggregating)
     + **Take random subsets (bootstrap samples) of the training data and feed them into different versions of the same model and let them all vote on the final result**
       - Bootstrapping is a type of resampling where large numbers of smaller samples of the same size are repeatedly drawn, with replacement, from a single original sample
     + Random forest uses bagging to implement ensemble learning
  2. Boosting
     + Alternative technique where each subsequent model in the ensemble boost attributes that address data mis-classified by the previous model
     + **Each model in the ensemble boosts (give more weight to) attributes mis-classified in the previous model so that subsequent models give more focus to them during training**
     + Keep refining model based on the weaknesses of the previous one
     + Uses all the data to train each learner but instances that were misclassified by the previous learners are given more weight so that subsequent learners give more focus to them during training
  3. A bucket of models:
     + trains several different models using training data and picks the one that works best with the test data
     + Take entirely different models (for example: Kmeans, decision tree, and regression), run all three models together a on a set of training data and let them all vote on a final classification result
     + Pick the model that wins
  4. Stacking
     + runs multiple models at once on the data and combines the results of all those models together to arrive at a final result

**DATA WAREHOUSING OVERVIEW: ETL AND ELT**

What is data warehousing

* A large, centralized database that contains information from many sources
* Often used for business analysis in large corporations or organization
* Queried via SQL or tools (i.e. Tableau)
* Often entire departments are dedicated to maintaining a data warehouse
  + Data normalization is tricky
    - How does all of this data relate to each other?
    - What views do people need?
  + Maintaining the data feeds is a lot of work
  + Scaling is tricky

**ETL: EXTRACT, TRANSFORM, LOAD**

ETL and ELT refer to how data gets into a data warehouse

If you move from a monolithic database built on Oracle or MySQL to more modern distributed databases built on top of Hadoop you can take the transform stage and do that after you load in the raw data (as opposed to before) and that can end up being more simple and more scalable, more efficiently.

* **ETL:** Traditionally the flow was Extract, ,Transform, Load (Old school way)
  + Raw data from operational systems is periodically extracted
  + Then the data is transformed in to a schema needed by the DW
  + Finally, the data is loaded into the data warehouse, ,already in the structure needed
* **ELT: Extract, Load, Transform**
  + Today a huge Oracle instance isn’t the only choice for a large data warehouse
  + Things like Hive let you host massive databases on a Hadoop cluster
  + Or you might store it in a large, distributed NoSQL data store
    - And query it using things like Spark or MapReduce
  + The scalability of Hadoop lets you flip the loading process on its head
    - Extract raw data as before
    - Load it in as-is
    - Then use the power of Hadoop to transform it in place

**BIAS AND VARIANCE**

**Bias:** how far removed the mean of your predicted values is from the “real” answer

**Variance**: is how scattered your predicted values are from the “real” answer

But you really care about is error

Error = Bias2 + Variance

But it’s error you want to minimize, not bias or variance specifically

**DATA CLEANING**

The reality is, much of your time as a data scientist will be spent preparing and ‘cleaning’ your data

* Outliers
* Missing data
* Malicious data
* Erroneous Data
* Irrelevant Data
* Inconsistent Data
* Formatting

**NORMALIZING NUMERICAL DATA**

* ‘Whiten’ data
* If your model is based on several numerical attributes – are they comparable?
  + Some models are ok with data that is not normalized (regression)
  + Some models may not perform well when different attributes are on very different scales
  + It can result in some attributes counting more than others
  + Bias in the attributes can also be a problem
* Examples
  + Scikit-learn PCA implementation has a “whiten” option that does this for you. Use it
  + Scikit-learn has a preprocessing module with handy normalize and scale functions
  + Your data may have ‘yes’ and ‘no’ that needs to be converted to 1 and 0
* Read the docs
  + Mot data mining and machine learning techniques work fine with raw, un-normalized data
  + But double check the one you’re using before you start
  + Don’t forget to re-scale your results when you’re done
    - To interpret the results you get

**DEALING WITH OUTLIERS**

* Our old friend standard deviation provides a principled way to classify outliers
* Find data points more than some multiple of a standard deviation in your training data
* What multiple? You just have to use common sense
* Identify outliers by looking at the number of standard deviations from the median (or mean) in a more principled manner rather than an arbitrary cutoff

def reject\_outliers(data):

u = np.median(data)

s = np.std(data)

filtered = [e for e in data if (u - 2 \* s < e < u + 2 \* s)]

return filtered

filtered = reject\_outliers(incomes)

plt.hist(filtered, 50)

plt.show()

A/B TESTS

* Controlled experiments on a website to measure the impact of a given change
* A controlled experiment, usually in the context of a website
* You test the performance of some change to your website (the variant) and measure conversion relative to your unchanged site (the control)
* Test all sorts of things
  + Design changes
  + UI flow
  + Algorithmic changes
  + Pricing changes
  + You name it
* How do you measure conversion
  + Ideally choose what you are trying to influence (optimize)
    - Order amounts
    - Profit
    - Ad clicks
    - Order quantity
  + But attributing actions downstream from your change can be hard
    - Especially if you’re running more than one experiment
* Variance is your enemy
  + - There’s so much random variance in order amounts to begin with that your result could be just based on chance
    - Sometimes you need to also look at conversion metrics with less variance
      * Order quantities vs order dollar amounts, for example
* Just looking at the differences in means is not enough, need to take into account variance as well.

**T- TESTS AND P VALUE**

Determining significance

* How do we know the result is likely to be ‘real’ as opposed to just random variation?
  + Was this just a result of random variance inherit in the data itself or are we seeing an actual statically significant change in behavior between our control group and test group
* T-Statistic ( aka T Test)
  + A measure of the difference between two sets expressed in units of standard error
  + The size of the difference relative to the variance in the data
  + A high t value means there’s probably a real difference between two sets
    - Have to decide threshold
    - Sign tells you if its good or bad change
  + Assumes a normal distribution of behavior
    - This is a good assumption if you’re measuring revenue as conversion
    - See also more refined version of t-statistic
      * Fisher’s exact test (for click-through rates)
      * E-test (for transactions per user)
      * Chi-squared test (for product quantities purchased)
* P Value
  + Probabililty of A and B satisfying the ‘null hypothesis’ (there is no real difference between control and treament’s behavior)
  + Low P-Value implies significance
    - Low probability that your change had no effect
  + It is the probability of an observation lying at an extreme t-value assuming the null hypothesis
  + Using P Values
    - Choose some threshold for ‘significance’ before your experiment
      * Liklihood there is no real effect, just a result of random variance
    - When your experience over
      * Measure your P-Value
      * If it’s less than your significance threshold then you can reject the null hypothesis that the change has no effect
        + Low p value means there is a low probability that this is just a result of random variance
        + + roll it out
        + – discard it before you lose more money

To declare significance need to see high t statistic (terms of standard deviations) and p value low (lower than threshold below 5% or ideally below 1%)

**HOW LONG DO I RUN AN EXPERIMENT?**

How do I know when I’m done with an A/B test

* You have achieved significance ( + or -)
* You no longer observe meaningful trends in your p-value
  + That it, you don’t see any indication that your experiment will ‘converge’ on a result over time
  + P value should come down over time, more data it gets the more significant your results should be getting
  + Flat line or going all over the place tells you that p-value isn’t going anywhere and it doesn’t matter how long your run the experiment
  + Need to agree up front in case you don’t see any trends in p-values, what’s the longest you’re willing to run this experiment
* You reach some pre-established upper bound on time

**A/B TEST GOTCHAS**

* Correlations does not imply causation
  + Even your low p-value on a well-designed experiment does not imply casuation
    - It could still be random chance
    - Other factors could be at play
    - It’s your duty to ensure business owners understand this
* Novelty Effects
  + Changes to a website will catch the attention of previous users who are used to the way it used to be
    - They might click on something simply because it is new
    - But this attention won’t last forever
  + Good idea to re-run experiments much later and validate their impace
    - Often the old website will outperform the new one after a while because it is a change
    - Measure it again when its no longer novel
* Seasonal Effects
  + An experiment run over a short period of time may only be valid for that period of time
    - Ex: consumer behavior near Christmas is very different than other times of year
    - An experiment run near Christmas may not represent behavior during the rest of the year
    - Look at conversion metric behavior for seasonal fluctuations, want to avoid running experiment during those peaks and values
* Selection Bias
  + Sometimes your random selection of customers for A or B isn’t really random
    - For example: assignment is based somehow on customer ID
    - But customers with low ID’s are better customer than ones with high ID
  + Run an A/A test periodically to check
  + Audit your segment assignment algorithms
* Data Pollution
  + Are robots (both self-identified and malicious) affecting your experiment
    - Good reason to measure conversion based on something that requires spending real money
  + More generally, are outliers skewing the result?
* Attribution Errors
  + Often there are errors in how conversion is attributed to an experiment
  + Using a widely used A/B test platform can help mitigate that risk
    - If yours is home-grown, it deserves auditing
  + Watch for ‘gray areas’
    - Are you counting purchases toward an experiment within some given time frame of exposure to it?
    - It that time frame too large?
    - Could other changes downstream form the change you’re measuring affect your results?
    - Are you running multiple experiments at once?
  + Data science from scratch, Joel Grus