**Unit 10 – Machine Learning**

Data Camp – ML using scikit-learn

* Supervised learning has labels on the data while unsupervised does not
* Classification with categorical data, regression for quantitative data
* ML needs to learn from already classified data
* Need to convert categorical features to numerical?
* *Classified data:* KNeightborsClassifier, can have different number of k, higher k is smoother, smaller k is in danger of overfitting
* Get data into proper formatted array prior to using scikit-learn
* OLS – ordinary least squares – minimize the square of the errors
* K-fold cross validation – divide into 5 groups, train on groups of four and test on one, 5 times
* Regularized regression – models are penalized for large coefficients
  + Picking alpha similar to picking k in k-NN
  + Lasso regression similar to ridge regression with the alpha
    - Shrinks coefficients of unimportant variable to 0
    - Lassos in only important variables
  + Alpha of 0 is OLS
* Classification problems
  + 99% of emails are not spam so if we build a model that labels all emails as not spam, our model would be 99% accurate, but useless
  + High precision: not many real emails predicted as spam
  + High recall: predict most spam emails correctly
  + Confusion matrix for categorial data
  + Logistic regression of binary category regression
    - Comes up with a probability for each
    - Have a probability threshold to categorize yes/no
  + A perfect model would maximize data under the AUC
* Scikit-learn does not accept categorical variables, need to convert them to dummy variables
  + pd.get\_dummies in pandas
* K nearest neighbors takes the actual distance, so a larger range will make the variable more impactful
  + We may want to scale our data to make each variable have equal have similar impact
* Minimize a log loss function (binary prediction) rather than maximizing accuracy (spam email predictor with 99% chance of success)

DataCamp Machine Learning with the Experts School Budget

* You cannot just pipeline in modifications to text and numeric data as the functions will try to operate on the entire data set
* FunctionTransformer
* The FeatureUnion brings the two separate pipelines together
* Tokens can also check and see if terms appear together (interaction terms
  + Beta1 \* x1 + Beta2 \* x2 + Beta3 (x1\*x2); x1 and x2 dummies must both be one only if they appear together

Bias and Regression Lecture

* Publication bias for p values (0.05 is convention)
* Survey non-response bias
* Selection bias
* Length bias (prisoners with longer sentence more likely to be in prison)
* Bias(theta-hat) = E(theta-hat) – theta; how far from the actual average
* MSE(theta-hat) = Var(theta-hat) + bias^2(theta-hat)
* I DON’T REALLY UNDERSTAND BIAS AT THIS POINT
* More complicated model has more variance although it could minimize bias

Regression Continued

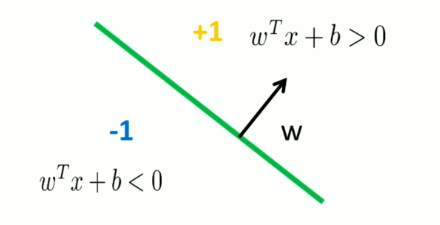
* Logistic Regression is for binary outcome; logit(p) = ln(p/(1-p)) log of the odds
  + Standard way to fit the model is MLE (maximum likelihood estimation)
  + Great property of logistic regression e^Beta gives you the odds ratio holding everything else equal except for the associate Beta
    - Easy to interpret the parameters
* Taking the log of variables of continuous variables is common for linear regression
* 5 dummies for five cases would be a blunder because of collinearity
  + All five would always add up to one and would always be a perfect linear relationship
* Collinearity makes the parameters meaningless and unstable
* Odds ratio – comparing probabilities across groups
  + Could there be other (confounding) factors? Fen/Phen users are more likely to be obese which is associated with heart disease; the users are different
* Ridge Regression: subtract out the sum of squared Betas; try to penalize model complexity; will reduce Betas towards 0
* LASSO Regression: subtract out the absolute value of Betas; penalize model complexity. Helps induce sparsity, reducing the number of variables one has to deal with; this will “brutally” reduce Betas to 0

Classification Lecture

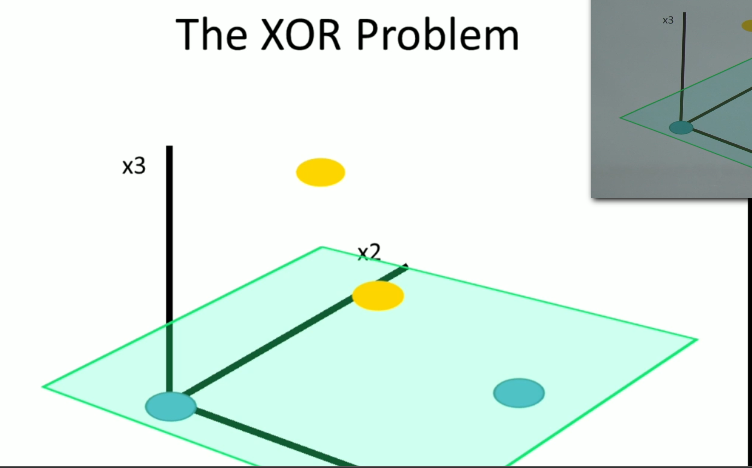
* Machine learning helped with image classification massively
* Humans can look through images and then the machine can learn from this
* You need to pick the right Feature (Selection) to help you distinguish in your model
* kNN: how many neighbors? Majority voting which is why we make k odd
* 1-NN: can create islands which isn’t ideal; the more complex the smoother the decision curve
* We would like training to take as long as necessary, but we want testing to be rapid
* We pick the k that has the lowest error?
* You need to hold out test data otherwise your results are not generalizable
* How can we bring the number of dimensions of our data down while persevering distance?
* Linear Regression vs PCA (principal component analysis)
  + Regression minimizes the vertical distances, PCA minimizes the orthogonal distance from the fit line

SVM and Trees

* Support Vector Machines
* kNN has the problem of needing the entire data set to predict. Training is fast because no calculation is really necessary, prediction is very slow
* How can we address this problem? Classifiers that have knowledge of the decision boundry



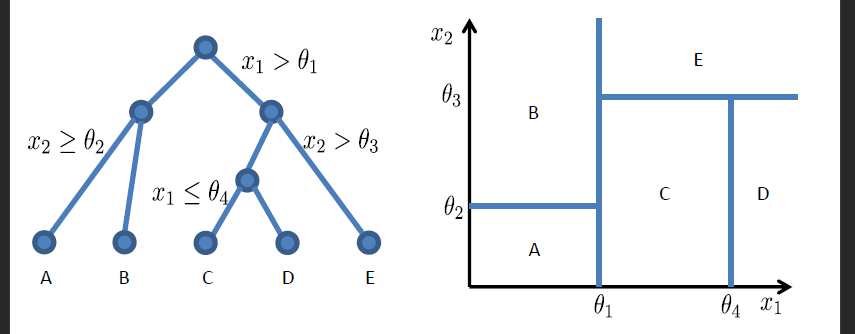
* this model will predict very quickly however we have lost a lot because we have a hyperplane (line) rather than the curved shape of kNN
* Perceptron is a mathematical representation of a neuron in the brain



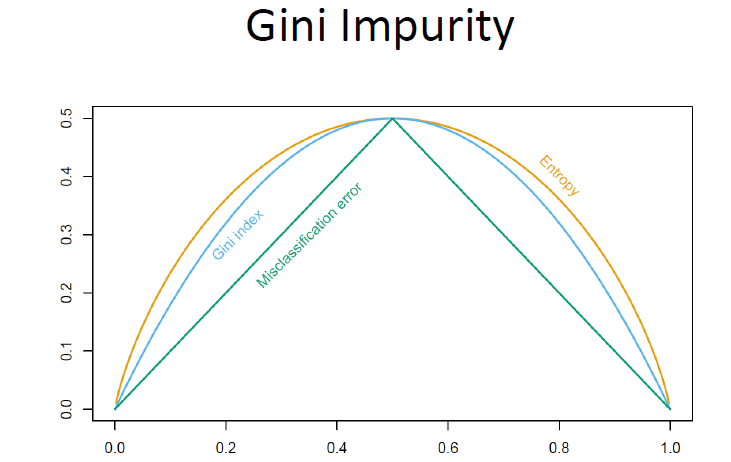
* You could have a simple 2d problem that could not be separated by a hyperplane. If you lift to an additional dimension you could separate the two. You want to keep only the dimensions that are useful, not too few or too many.
* Support Vector Machines
  + some say it is the best off the shelf classifier
  + however, you may need to tune hyperparameters for best performance
  + Maximum margin classification to max the distance from the hyperplane to the data
* An outlier (or mislabeling) could radically impact the hyperplane
* We can tell the machine to give slack with slack variables; large C means less slack; higher gamma means more df (more dimensions); tune these together
* Only need support vectors which define your decision boundary
* You should normalize data for SVM (mean: 0, std: 1)
* Plotting ROC curve
* Recall: if I pick a random positive example what is the probability of making the right prediction
* Precision: if I take a positive prediction example, what is the probability that it is a positive example?
* Run n classifiers for n categories, one against all
* Compute a confusion matrix

Decision Trees and Random Forests

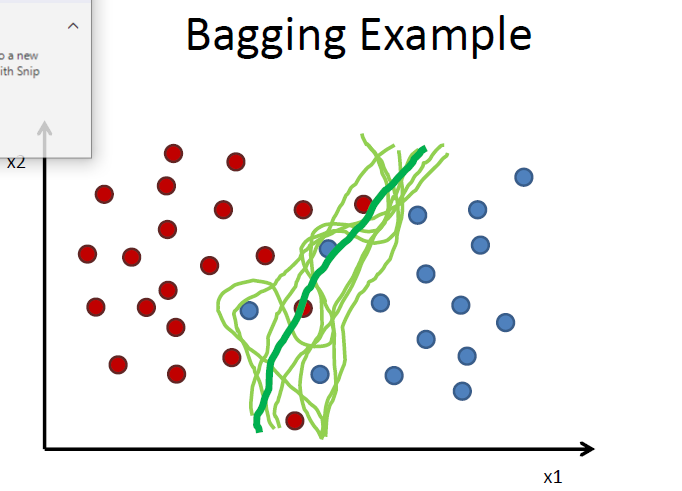
* Decision trees are fast to train, fast to predict, easy to understand, easy to interpret
  + What is the drawback?
  + Looks like simple classification decisions



* Imagine drawing straight lines through two variables, one feature at a time
  + What is the downside?
    - You can only draw straight lines from querying one feature at a time
    - SVM can draw diagonal lines
  + What is the benefit?
    - Performance is very fast only looking at one feature at a time
    - Intuitive
    - No need to train multiple classifiers like SVM
* How do we build a tree from our data?
* Which features do we choose and what threshold do we use?
* Gini Impurity is the expected error if you randomly choose a sample and predict the class of the entire node based on it
  + I don’t completely understand. RESEARCH THIS. Do this mean just picking at random with no attempt to classify?



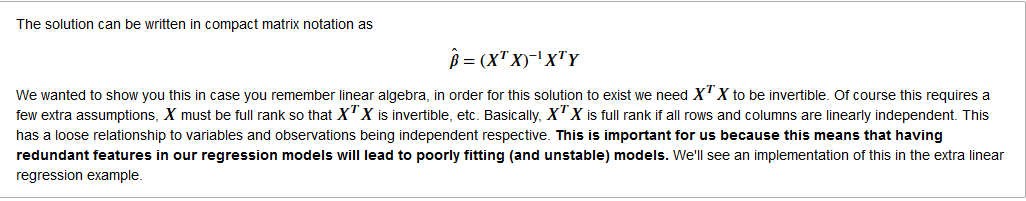
* We would like to optimize misclassification error; however, the Gini index is easier to work with and you can more easily take the derivative
* Pseudocode
  + Check if already finished
  + For each feature x(subi)
    - Calculate the gain from splitting on x(subi)
    - Let x(best) be the feature with the highest gain
  + Create a decision note that splits on x(best)
  + Repeat on sub-nodes
* When do you stop optimizing?
  + Node contains only one class (pure)
  + Node contains less than x data points
  + Max depth is reached
  + Node purity is sufficient
  + Overfitting (validated through cross-validation)
* Decision trees aren’t that popular because they tend to overfit (sensitive to small changes in the data)
  + One method to solve this is tree pruning to remove decisions
  + Easily handles missing values as we only look at one feature at a time
* What if we train multiples trees? An ensemble?
* We need to make sure they train on different data so that we have different results
* Bootstrapping is a popular technique
* Cannot use bootstrapping for cross-validation (same data problem)
* Bagging – bootstrap aggregating
  + You can have high variance (complexity, overfit) but use the different models which you can then use to vote or average to produce a smooth line
  + Reduce the variance without having too much bias



* Bagging does perform better use for techniques with high variability (not linear regression)
* Random Forest
  + Node splits calculated from random feature subsets
  + Didn’t get too deep into it

Linear Regression Mini-Project 1

We assume iid for linear regression. Residuals are expected to be independent and identically distributed. Time series data often has the iid problem (heteroskedasticity problems).



* Independence of columns is crucial

Logistic Regression

Classifiers that set a hard boundary are known as discriminative

Gradient Boosting

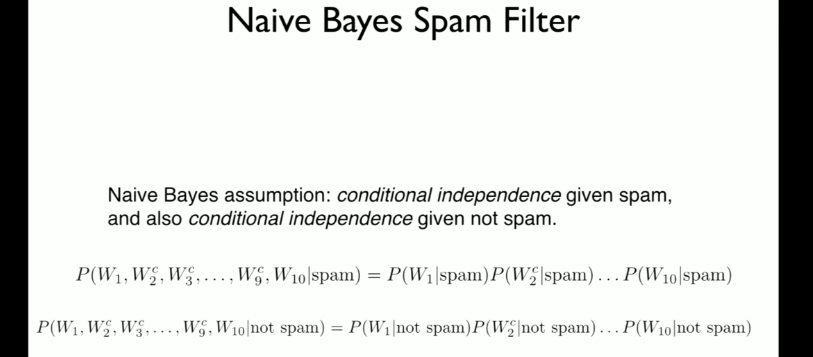
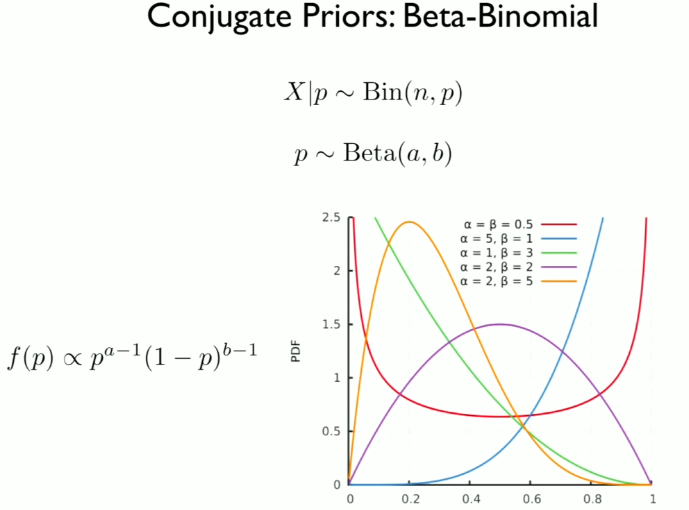
• We first model data with simple models and analyze data for errors.   
• These errors signify data points that are difficult to fit by a simple model.   
• Then for later models, we particularly focus on those hard to fit data to get them right.   
• In the end, we combine all the predictors by giving some weights to each predictor.

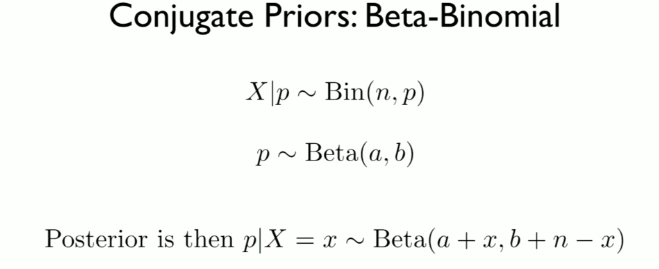
<https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d>

Using Random Forests in Python Lecture

* We want high information gain separators (if possible) separate the data into two equal groups
* Overfitting problem with trees; ensemble model … bagging
* Build a stronger aggregated model from weaker models (fewer predictive features)
* Random forests are robust to different types of data (non-linear, unscaled, missing values, poorly chosen features)
* Paramaters are intuitive and simple with forests
* Depth of the tree depends on the complexity generally shoot for SQRT(M) where M is the
* <https://github.com/NathanEpstein/pydata-london>

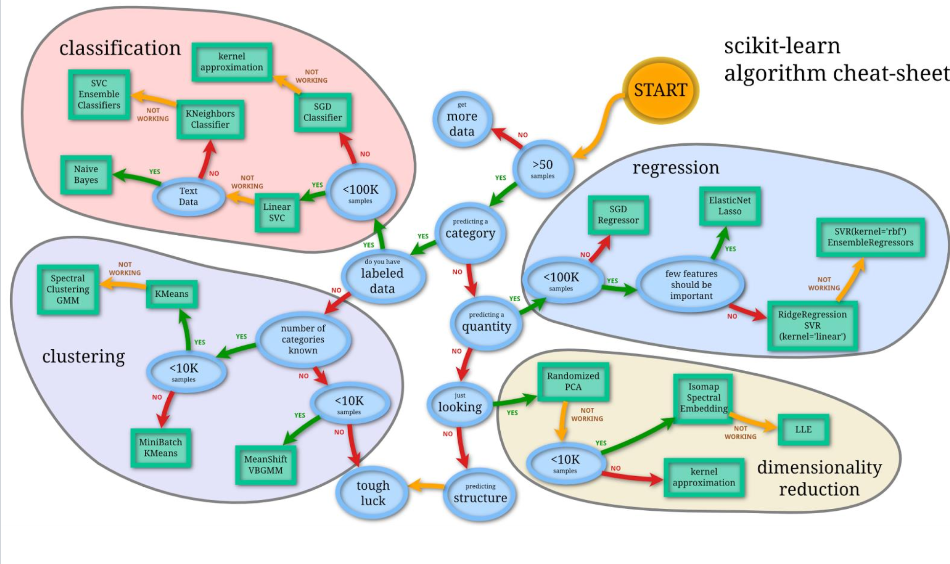
Bayesian Spam Filter Lecture

* Naïve Bayes assumes conditional independence
* 
* Think like a Bayesian, check like a frequentist
* 
* Very flexible distribution



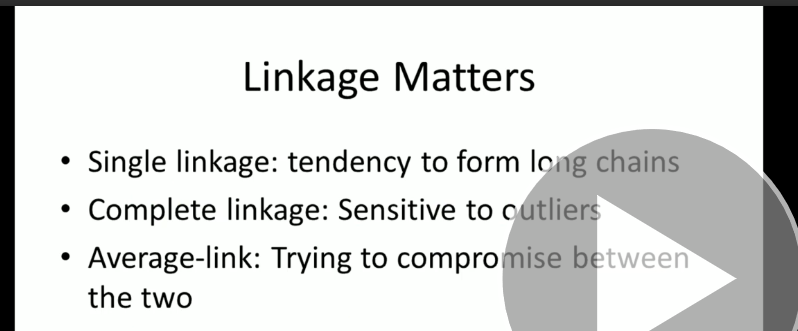
Best Practices Lecture

* Random forest is the bagging idea plus a subset of features
* Randomization is making one feature useless and seeing how much your prediction changes; this tends to spread out importance more uniformly
* Knn regression: uniform weights or distance weights? Changes look of regression line



* if your hyperparameters appear on the edge of your grid search you should widen your grid.
* Need to hold back the test data and don’t pollute
* Imbalanced data can create problems: what can you do?
  + Apply bootstrapping, add weight to smaller group, stratify across cross-validation?

Unsupervised Learning Lecture

* Works well with data mining for finding categories
* Can be used for outlier detection
* **K-means clustering**
  + randomly assign anchor points (or choose random points as anchors), what points are closest to which anchor points (new cluster)?
  + update the anchor points and reiterate (use mean or median)
  + exact same k can produce different results
  + the main drawback is you have to specify the number of clusters, choose k based on how well it improves your modeling? (don’t overfit)
  + Use cross validation (training data to identify cluster points), and test date to identify distances (model performance)
* Removing K: **Mean-shift clustering**
  + Must be used for every single data point, computationally expensive
  + Window is drawn in to clustering points
  + Many points will converge to center of a cluster
  + Specify the window size, not the number of clusters
  + Robust to initialization
* **Hierarchical Clustering**
* 
* Start with points that are closest together; then iterate
* Need to choose a threshold of when to stop
* 
* 