Algorithm Ideas

1. Use dataset internal extrapolation to determine whether a model will generalize to external test data when trained on the full dataset. (and ideally to build the model itself..)
2. Create meta-ml algorithms, which instead of being given a functional form, learn their functional form assumptions from data and then use that knowledge to create an inductive bias
    a. The algorithm crawls over many datasets, seeing & remembering relationships
    b. Can start with 1d to 1d predictions, where every feature is predicted from every other feature (and the model learns to find a functional form that fits well / generalizes for every sub-task, where the model is a function of the data)
    c. like, learn a boxcox transform bias of something like it
3. New notion of model complexity - flexibility in the decision boundary, fit to bootstrap resamples of the data
    a. Need a way to compare decision boundaries to one another
    b. This would also allow the comparison of different models' variance to one another, it's on the same scale!
    c. This is much closer to what we mean by model flexibility. Notions of variance like the contribution to the error can easily be masked by different sources of error generating the same total contribution to the error.
4. Find a 'boundary wiggliness' metric that will let us compare decision boundaries learned over random labels vs boundaries learned over correct labels
    a. (Great example of variance being a function of the data)
5. It may make sense to use every linear modeling technique that is valuable on top of a network.
6. It would be great to have a 'linearize' function that takes some arbitrary feature and label and does transformations to the feature until it's as close to linearly related to the output as possible. Much like the boxcox transform, but for linearity. Adaptive basis functions are like this, but there are ways to do adaptive basis function regression that's not a neural net.
7. Metalearn extrapolation by predicting out-of-domain datapoints in the dataset continuously, as huge amounts of data on how successful extrapolation works
8. Take correlated features (when training a linear model) and synthesize them somehow, finding the ways in which they vary (something like pca over correlated features) and so combining the correlated sub-parts of them into individuated features.
9. Use heuristics for variance (model capacity, aspects of the dataset) as features to predict the variance as measured by overfitting, or the variance as measured by multiple re-fits of the same data (where the difference in the decision boundaries is the response / label)

10. In cases where the test error varies substantially based on the training dataset used, we can learn which training datapoints to sample or attend to in order to generalize well. A metalearning multi-task setup which discovers this could be useful.
11. What if we (confirmation bias) modify the data to fit the parameters? We do this all over the place to create consistency or do data correction - maybe it's not all BS! If you realized that your data sources can be biased or noisy, correcting them to be more aligned with a model you're justified in being confident in is very helpful for avoiding poor fits due to outliers, appropriate weighting for future fitting, etc.
12. 'Stacking' as treated here is about weighting models intelligently, not feeding the inputs into

Algorithms

1. Forward and Backward Stepwise Selection
    a. Forward-stepwise selection sequentially and greedily ads into the model the predictor that most improves the fit. Backwards-stepwise selection starts with the full model and deletes the predictor with the least impact on the fit.
    b. Clever use of the QR decomposition for the current fit can rapidly establish the next best candidate.
2. Forward-Stagewise Regression
    a. Adds the variable most correlated with the current residual error, and adds to that variable's coefficient until it is no longer the variable most correlated with the current residual. It continues until no variables had correlation with the residuals.
3. Least Angle Regression
    a. Like forward stepwise selection, but for adding as to the variable until it is not longer the most correlated with the response. Once it catches up with another variable, that variable joints the active set.
4. Principal Components Regression
    a. Linear regression over the top M principal components of the input set.
5. Partial Least Squares
    a. Uses the labels relationship with the features to iteratively create components that are linear combinations of the features, and then does linear regression over those components.
6. Perceptron
    a. Finds a separating hyperplane by minimizing the distance of misclassified points to the decision boundary. Uses SGD to minimize a binary objective function.
7. Cubic Spline
    a. Spline where the function is continuous, nad hs continuous first and second derivatives at the knots.
    b. Optimized by minimizing a penalized residual sum of squares
8. Natural Cubic Splines

    a. Adds a constraint to splines that the function be linear beyond the boundary knots.
9. Smoothing Splines
    a. Adds a smoothing term and parameter to the sum of squares loss function.
10. Wavelet Smoothing
    a. Shrinkage towards a sparse representation of the coefficients, using a threshold over the smaller coefficients. Fit by least squares.
11. One-Dimensional Kernel Smoothers
    a. Local Linear Regression, Local Polynomial Regression
12. Kernel Density Classification
13. Generalized Additive Models
14. Projection Pursuit Regression
15. Support Vector Machine
16. K-medoids
17. Non-negative Matrix Factorization

Optimizers

1. Expectation Maximization
2. Bootstrap

Linear Methods for Regression

1. Unique Value:
    a. Small data settings
    b. Low signal-to-noise ratio settings
    c. Sparse data settings
2. Introduction:
    a. Form of the linear regression model
    b. Types of basis function transformation
    c. Loss function (Sum of squares)
    d. Minimization
        i. Matrix form loss function
        ii. Differentiate with respect to beta
        iii. Obtain unique solution
    e. Inference equation (as a function of unique solution)
    f. Principles around linear independence, full rank assumption, estimate of y as orthogonal projection, singular condition
3. Gauss-Markov Theorem - least squares estimates of the parameters have the smallest variance among all linear unbiased estimates. That said, shouldn't restrict ourselves to unbiased estimates - regularization can be biased, say. Trading some bias for a large variance reduction makes sense.
    a. This makes me think that I don't understand what bias is.

    b. bias-variance decomposition
4. Multiple Regression from Simple Univariate Regression
5. Multiple Outputs
    a. Generalize the univariate loss function by turning Y into a vector.
    b. Outputs do not affect one another
        i. In this case it maps to a multivariate gaussian
        ii. There are other cases where it's worth combining the regressions
6. Subset Selection
    a. Best subset-selection
    b. Forward and Backward Stepwise Selection
    c. Forward Stagewise Regression
7. Shrinkage / Regularization
    a. Subset selection as 'high variance', lol, operators applied to operators. Shrinkage as continuous regularization, where subset selection is discrete regularization.
    b. Ridge Regression
        i. Ridge solutions aren't equivariant under scaling.
        ii. Singular value decomposition for least squares
            1. / Principal components
            2. / Eigendecomposition (Eigenvectors are principal components)
            3. Principal components capture maximum variances, subject to being orthogonal to vectors previous (strong constraint!)
        iii. Can see lambda as the effective degrees of freedom. Continuous degrees of freedom, as opposed to the discrete version you get by counting parameters.
    c. Lasso
    d. Principal components Regression
    e. Partial Least Squares
    f. Least Angle Regression
        i. Can be paired with lasso
8. Methods that make updates to the input space
    a. Principal Components Regression
    b. Partial Least Squares
9. Multiple Outcome Shrinkage and Selection
    a. Reduced-rank regression
10. More Lasso and Related Regularization Path Algorithms
    a. Incremental Forward Stagewise Regression
        i. Connection to boosting
    b. Dantzig Selector
        i. Solution is to use linear programming.
    c. Grouped Lasso
    d. Further Properties of the Lasso
    e. Pathwise Coordinate Optimization
11. Computational Considerations

    a. Fitting is usually done via Cholesky decomposition of X^TX or QR decomposition of X. Cholesky runs in $p^3 + Np^2/2$ operations. QR runs in Np^2 operations.
    b. Depending on the relative size of N and p, Cholesky can sometimes be faster, but it can be less numerically stable.

Thought:

1. I feel like linear algebra people can nerd out about least squares, and so when they're teaching first principles they teach things that don't actually cleanly generalize (like closed form solutions, convexity, computability of the hessian, etc.)
2. It seems like these guys abandoned structure, or at least aren't granular in a way that makes it completely clear what is happening in each space from a higher level.
3. The concept of shrinkage as lower variance regularization than subset selection is exactly why this book is awesome. General principles weaved throughout the entire text. It's such a treat to read.
4. These guys are weirdly obsessed with auto-regularizing linear models. Feels far too incremental to me, as well as structurally lacking - it's clear that linearity isn't enough.

Concepts from Foundations that I need to learn:

1. Matrix Differentiation:
    a. Used in deriving least squared gradient for both solution and descent
2. Joint Distribution PR(X,Y)
    a. Used in defining real valued input and output vectors for learning, choosing and solving a loss function

Linear Methods for Classification

1. Introduction
    a. Cool thought - see the decision boundaries in linear models as the places where indicator functions for two classes have equal value.
2. Linear Regression of an Indicator Matrix
    a. Fit a linear regression model to every class in the dataset. Output the class that linear regression returns the highest value for. (Where your target Y is a one-hot coded output)
    b. There's a huge problem when you hit 3 classes, where a class whose values are in the same direction of another class but are larger will always have a higher value in the indicator function - linear regression can't cut out space in between classes, and so ends up missing these masked classes completely.

    c. It would be wonderful to formalizing masking (as a problem) and ask whether all other linear methods completely avoid it (this notion of 'being between' classes and so being missed one way or the other.
3. Linear Discriminant Analysis
    a. Assume the covariance matrix for each class is identical
    b. The data often can only support simple decision boundaries such as linear or quadratic - what a great insight, about what level of complexity a dataset can support. Ideally algorithms would be adaptive to this.
        i. This is a bias-variance tradeoff, so of course I love this idea of the kinds of boundaries that a datset can support. Here, it looks like tolerating the bias in LDA / QDA to get the reduction in variance.
4. Regularized Discriminant Analysis
    a. Here, the covariance matrix is a linear combination of a class-specific covariance matrix and a pooled covariance matrix (the one used in LDA), interpolating between LDA and QDA with a hyperparameter alpha.
    b. My instinct - combining with the pooled covariance matrix feels wrong. There should be an even weighting between the covariance matricies, equally respecting them. (what principle was violated? What drove this intuition?)
5. Whoa, canonical correlation analysis between data and the label indicator matrix
6. One major difference is that because LDA assumes gaussian data, it overweights outliers.
    a. This is similar to this thought on one kind of overfitting - how much do you weight dense data generalizing against sparse data that's closer by?
    b. These are the subtle differences between techniques…
    c. It's unprincipled to use LDA with qualitative predictors (though in practice it's only slightly less robust than logistic regression)
    d. Implication is that linear / logistic regression aren't gaussian. LDA is gaussian. Naive bayes is gaussian.

Thoughts

1. Linear models on simple basis transforms looks like it gets fairly strong generalization, especially outside of domain, empirically.
2. The way that the correspondence between LDA and linear regression of an indicator matrix falls apart if you add more than one class is a decent example of seen similarity or unity or correspondence that fails to generalize.
    a. I'd like to know why LDA avoids masking
    b. Would be cool to see this correspondence fall apart formally

Basis Expansions and Regularization

1. Introduction
    a. Reaction - basis transforms are weaksauce - we should abandon linear models entirely, and fit nonlinear functions, rather than doing the transformations to the data beforehand and then just running them through a linear model.
        i. Then, I realize that this is what deep learning is, and I just invalidated the modern paradigm. Oh well. Do I invalidate it? I do invalidate it.
    b. Common (static) basis function transforms:
        i. Polynomials
        ii. Interactions
        iii. Log
        iv. Square Root
        v. Norm (across all features, or subsets of features)
    c. WOW, what a strong reframe - you can see decision trees as deciding or not deciding to include features from a dictionary. It's a selection method.
        i. MARS does this too.
    d. Controlling complexity
        i. Model simplicity (restriction methods)
        ii. Selection methods (over features)
        iii. Regularization methods
2. Piecewise Polynomials and Splines
    a. Cubic spline is called cubic because it's continuous in the objective function value, the first derivative and second derivative
    b. Cubic splines are the first level at which the knots are not visible to the human eye - I have to wonder about higher orders in high dimensions. Continuity in the third, in the fourth derivative.
    c. Polynomials fit to data are erratic and often fail to extrapolate.
        i. Splines are even worse at the boundaries.
3. Smoothing Splines
    a. There's plenty of content here on eigen-vectors and values that's going over my head but it likely really strong.
4. Multidimensional Splines
    a. I expect that these have horrible scaling properties with dimension. Looking forward to this getting wrecked by the curse of dimensionality (though, maybe it can be done on top of a DL representation…)
5. Regularization and Reproducing Kernel Hilbert Spaces
    a. Punish functions inversely to their eigenvalue size. Functions with large eigenvalues get penalized less.

Thoughts

1. Adding tons of parameters and then regularizing gives up on strong generalization
    a. The kind of 'not overfitting' you get with regularization is different than the kind of not overfitting you get by keeping your input space simple.

    b. This is why learning to abstract is necessary for strong generalization. You have to fit your model in a low-feature-count space. Over-parameterized models will only be able to interpolate, not extrapolate.
2. Piecewise models makes thinking about which datapoints you should / can do transfer from, as well as how much you should do that transfer an explicit object of thought.
    a. You can think of all modeling as extrapolation from other datapoints in the dataset.
    b. Meta-learn to extrapolate by predicting out-of-domain datapoints in the dataset continuously, as huge amounts of data on how successful extrapolation works
3. Why don't people fit feature conditional models? I gues it's like putting a model on top of a decision tree, where first you split on one feature (say, male / female) and then you train models.
    a. You'd like to automatically search the space of useful splits for this kind of modeling.
4. There are some cases where extrapolation should be the only standard for judging the quality of a fit.
    a. We need training procedures that fit <u>for extrapolation</u>. Something like an enhanced version of cross validation that fits to subsets of the data chosen to not have complete coverage, so that extrapolation can be learned from data in uncovered regions.
5. There actually aren't that many concepts that I don't understand.
    a. Eigen-expantion / eigen-function
    b. Fourier Transform
    c. Kernel
        i. RBF-Kernel
    d. Orthogonal Projection
    e. Joint likelihood
    f. Whoa, gaussian random fields…

Kerenel Smoothing Methods

1. Local Regression in R^P
    a. Whoa - you cna just fit linear regression models to subspaces, and get better boundary performance than KNN
    b. And (as I predicted!) "Local regression becomes less useful in diemensions much higher than two or three" - your variance blows up because the number of datapoints in each neighborhood shrinks dramatically with increased dimensionality.
2. Kernel Density Estimation and Classification
    a. Non-parametric kernel models have curse of dimensionality issues (they look at local neighborhoods, and so become too high variance)

    b. Yeah, density estimation is still unattractive in high dimensional feature spaces (because we don't know how to effectively interpolate / extrapolate between datapoints)
    c. Upside to naive bayes - since density estimation is hard in high diemsions, just make kernel density estimates in one dimension for each feature and then aggregate as if they were independent!! (This is an alternative to using univariate gaussians)
        i. What a way to lower the variance of your estimate…
    d. Naive bayes -> generalized additive models as LDA -> logistic regression
3. Radial Basis Functions and Kernels

Thoughts

1. Locality is in time series data!! Close in time is a super important interaction signal. So we have locality in temporal and visual data, which can help explain the success of CNNs / RNNs which look at local interactions (The RNN by only taking in the last n time steps).

Model Assessment and Selection

1. The Bias-Variance Decomposition
    a. kNN with 1 neighbor has zero bias on the training set but non-zero bias on the test set - makes me wonder, should we be measuring bias/variance on the training set or the validation set? Variance is the difference between training and validation error… bias is distance from the bayes error (irreducible error)
    b. A good example of irreducible error is your training data being inaccurate (say, you're measuring continuous values and your data is noisy due to measurement error)

Vapnik-Chervonenkis Dimension

VC dimention is a notion of compexity & model flexibility that is more general than the use of number of parameters used to determine model complexity in the Akike Information Criterion and the Bayesian Information Criterion.

Instead, VC dimension can differentiate between model complexity of different functional forms, bounding the optimism (that is, the distance between train and test error).

Shattering is the number of datapoints for which a class of functions cannot perfectly separate the data. VC dimension is the maximum number of points that is shattered by a functional form.

For linear models, vc dimension is the number of free parameters.
Some functional forms will have infinite VC dimention (can shatter an infinite number of datapoints).

It is possible to bound the error above (bounding the optimism of the training error). These bounds are parameterized, though you can look at the (probabilistic) worst case. It's possible to use VC dimension in your search for a quality model, with the lowest upper bound on the optimism being your selection critereon (this is structural risk minimization).

They compare AIC against BIC against Structural Risk Minimization (SRM) for model selection. Num parameters underestimated the degrees of freedom fo a linear model, for a reason I don't understand - "because it doesn't charge for the search for the best model of that size." (Oh, I see - more expensive search can happen without any change to the number of parameters)

The AIC performs well across knn and linear models. BIC is slightly worse, SRM is kind of all over the place (higher variance and often worse performance, though it does do well with linear regression.

Cross Validation

Cross Validation checks the error on the extra-example data, that is on a holdout dataset that gives an estimate of the generalization error (rather than the training error). K-fold cross validation splits the data into k pieces and then trains on one fraction, validating on the other fraction.

You can use cross validation to pick your hyperparameters empirically, by looking for the hyperparameter that performs best. Leave one out cross validation - use one datapoint for validation, train on the rest.

It's important to only use the folds / training (and not validation) set when making choices about, say, which features to include or what hyperparameters should be used. In general with the modeling procedure, cross validation must be carried out over the entire sequence of modeling steps. (This is actually importantly different from systematized predictive modeling, where cross validation is just seen as a way to get a good estimate of the validation error. But much of the transformations are labelless, and so legitimate.)

Bootstrap Methods

Bootstrap methods work by resampling the dataset, with replacement, to the size of the original training data (and so allowing for repeat datapoints, and missing corresponding datapoints) and then training a given model on each resampled version of the dataset.

When you estimate error for the bootstrap, use the datapoints that were not sampled as training points. (weirdly, on average you sample 63-64% of the dataset when sampling with replacement to the same value)

You can't used the Akike Information Criterion without a notion of the number of parameters in the model. For adaptive non-linear models like decision trees this is very difficult, and so we are left with the bootstrap or cross validation for estimating the generalization error. Moreover, the boostrap and cross validation do a much much better job of actually correctly estimating the test error.

The quality of the cross validation or bootstrap estimate is a function of the variance in your model. Methods like trees can lead to underestimations of the true error of up to 10%, because the seach for the best tree is strongly affected by the validation set.?

Thoughts

1. The fact that variance doesn't increase with the depth or width of a neural network is an example of a failure to generalize to deep learning from other (say, linear) models. Which should start to say something about the generality of our knowledge.

Model Inference and Averaging

Regression and classification have been taught, thusfar, through maximum likelihood (whether through sum of squares or cross entropy). There is a bayesian alternative (and I would love to know what the space of things like likelihood are - this is general across ML).
The sectio nwill cover committee methods, including stacking!, bumping (which I've never heard of), bagging and committee methods.

The bootstrap can be used to assess uncertainty.

When we fit a basis expation linear model, there's a covariance matrix of the estimated weights of the model.

Can see the bootstrap as being based on the data, and so call it the 'non-parametric' bootstrap - this is the same intuition that drove my sense that the probability distribution should be an arbitrary function of the data, rather than the result of a distributional ssumtion.
There's a version of the bootstrap where you add gaussian noise to the existing datapoints, referred to as the parametric boostrap.

This feels degenerate to me. Aren't you just building up error bands from your arbitrary distributional assumptions? Maybe it'll work if you're implicitly doing transfer from other contexts where the noise actually is gaussian. But this will fail everywhere that that's not true.
gasp in general the parametric boostrap agrees with maximum likelihood??? Who knew that maximum likelihood made such strong assumptions? It it implicitly gaussian??

Ugh. Immediately it starts off with this random notation, z_i proportional to g sub theta of z. I'm trying to *fucking learn maximum likelihood* and they're pulling me around.

You can use likelihood to assess the precision of the weights (theta). In the precision / recall sense? Or some other sense?
Question - why do you provide no motivation for the information matrix (as the second derivative respecting theta of the log likelihood), or for expected information (the fisher information matrix), or for the value of standard errors on them?

Advantage of the bootstrap over maximum likelihood is that it lets us compute standard errors or confidence intervals.

The reason that nobody knows what maximum likelihood is is that all anyone does is throw math at you. It's finding the parameters that maximize the probability of the data. It's implicitly generative, where maximizing the probability of the data implies that your model should be the one that is most likely to generate the data.

Bayesian Methods

Distributions on functions:
A Gaussian process prior in which we specify the prior covariance between any two function values.
Not understanding covariance is by far my biggest bottleneck right now.
I really would benifit from playing with the covariances of a few probability distributions. And really getting a feel for how variance and covariance are computed.

The bootstrap is an (approximate) nonparametric, noninformative posterior distribution for the parameter in bayseian inference. But the bootstrap doesn't require that you specify a prior and sample from the posterior distribution.

Bagging

Important to bag with the probability estimates of models, rather than the classifications themselves.
Bagging can make a bad classifier (that's worse than guessing) even worse, by lowering its variance in randomly choosing the right option.

Interesting, never thought of the wisdom of the crouds (when they're independent of one another) as an ensemble model, but it totally is.

Model Averaging and Stacking

Yes, linear regression on your candidate models! Love it.
Frame - stacking accounts for the complexity of the model when weighing the models against one another. It does this by using validation error (cross validation) rather than training error.
This is very different from feeding the results of one algorithm into another algorithm.
More advanced weightings over models can depend on the datapoint.
Bumping - use the bootstrap, but pick the model that performs best. This actually reminds me (weakly) of my algorithm idea, searching for the model that can generalize.

Additive Models, Trees, and Related Methods

Oh, f - is generalized additive models an automatic feature transformer?
Oh, whoa - they fit each basis function independently??
Instead of linear weights, you have a more general functional form. Link functions. Logs, identity, logits. No wonder people just use linear models - generalized additive models with link functions are a special case of feature engineering a linear model.
Oh, this is stacking as I know it!! You have two models - the cubic spline of a single feature (which 'finesces the curse of dimensionality), say, and a loss that aggregates over all of them.
Oh, you can average over these low level functions - kind of like an ensemble but with each feature being the dataset.
Algorithms as sub-functions in generalized additive models:

1. Cubic Splines
2. Local Polynomial Regression
3. Kernel Methods
4. Piecewise constant fits
5. Surface Smoothers

Ah, I see - it ends up estimating the residual. That makes some sense.

Write-up of Contents

- Overview of Supervised Learning
    a. Least Squares and Nearest Neighbors
        i. Least squares works by performing a linear transformation on a set of input features (basis functions that can be an assortment of

transformations on the original input) and then optimizing the output over a sum of squared error loss function. Optimization proceeds via the normal equations and QR Decomposition / Cholesky Decomposition / SVD, or via gradient based methods like Newton's Method, Gradient Descent etc.

ii. Nearest neighbors measures the euclidean distance to each point in a training dataset and returns the label of the most common class of the closest k datapoints, or the average of their outputs in a regression task.

b. Statistical Decision Theory
    i. The choice of loss function (cross entropy loss, sum of squared error, hinge loss, kl-divergence, L1 or median, zero-one loss) is an important one, as different loss functions have different properties.
c. Local Methods in High Dimensions
    i. Bias-variance decomposition and the curse of dimensionality.
    ii. The curse of dimensionality afflicts k-Nearest neighbors, where all distances between datapoints become very small in euclidean space.
d. Statistical Models, Supervised Learning and Function Approximation
    i. We can see function approximation as a more general frame for classification and regression, where there is a function for mapping a set of datapoints to labels that is approximated via approximation.
e. Structured Regression Problems
    i. There's a structured output problem, where some xyzs zyx.
f. Classes of Restricted Estimators
    i. Roughness Penalty and Bayesian Methods
        1. Roughness penalties prioritize smoothness, that is that hypotheses with a smaller first derivative in the approximated function are prioritized over functions with a larger first derivative. Could be second derivative (say, in a cubic smoothing spline)
    ii. Kernel Methods and Local Regression
        1. Kernel methods will use a kernel function (say, Gaussian kernel) which measures the similarity of datapoints (for some notion of similarity). Points are weighted by their similairty, in a way that dies exponentially with their distance. Then you use a sum of squared error that's a function of this local weighting.
    iii. Basis Functions and Dictionary Methods
        1. Basis functions can be chosen adaptively via a neural network. Adaptive basis function methods are also called dictionary methods.

- Linear Methods for Regression
    a. Subset Selection
        i. Best-subset selection selects the most important feature, or deselects the least important feature.
        ii. Forward stepsize starts with just an intercept term, and adds the feature that most improves the fit.
        iii. Backward stepsize starts with all features, and removes the feature that least impacts the fit.
        iv. Forward stagewise regression starts with coefficients at 0, and adds to a coefficient that most improves the loss until some other coefficient would improve the loss even more. This goes until increased coefficients in any variable isn't correlated with the residual error.
- Linear Methods for Classification
    a. Linear Discriminant Analysis
        i. Model each class density as a multivariate Gaussian. Assume that those class densities have a common covariance matrix. We estimate the parameters of the multivariate gaussian (the means mu, the covariance sigma) by using the values in our training data for each feature for each class. We can write down the decision rule as a function of the mean, covariance, the input data, and the proportion of the datapoints for each class in the dataset.
    b. Separating Hyperplanes
        i. The Perceptron algorithm minimizes the distance of misclassified points to the decision boundary, using a 1 / -1 score for each kind of misclassification and minimizing the error only on misclassified points. Optimized via SGD, where the gradients come out of the loss.
- Basis Expansions and Regularization
    a. Piecewise polynomials and splines
        i. Piecewise polynomials
        ii. Wavelet Smoothing
- Kernel Smoothing Methods
    a. One-Dimensional Kernel Smoothers
    b. Kernel Density Estimation and Classification
- Model Assessment and Selection
    a. The Bayesian approach and BIC
    b. Minimum Description Length
    c. Vapnik-Chervonenkis Dimension
        i. VC dimension is a notion of complexity & model flexibility that is more general than the use of number of parameters used to determine model complexity in the Akike Information Criterion and the Bayesian Information Criterion. Instead, VC dimension can

differentiate between model complexity of different functional forms, bounding the optimism (that is, the distance between train and test error). Shattering refers to the number of datapoints for which a class of functions cannot perfectly separate the data. VC dimension is the maximum number of points that is shattered by a functional form. For linear models, vc dimension is the number of free parameters. Some functional forms will have infinite VC dimension (can shatter an infinite number of datapoints). It is possible to bound the error above (bounding the optimism of the training error). These bounds are parameterized, though you can look at the (probabilistic) worst case. It's possible to use VC dimension in your search for a quality model, with the lowest upper bound on the optimism being your selection criterion (this is structural risk minimization).

- Model Inference and Averaging
    a. The EM Algorithm
- Additive Models, Trees, and Related Methods
    a. Generalized Additive Models
        i. A Generalized Additive Model takes a function as its basis function transform and learnes a linear model over those functions. This looks like a re-name of generalized linear models, where link functions connect ….
            1. There's stuff about link functions that I don't think I get, and there's an optimization process that isn't just IRLS or newton's method that I also don't get.
            2. Automated methods for considering non-linear effects as pre-processing for a linear model. If the model was an expansion of basis functions, then it could still simply be fit by least squares. But here, each function is fit with a smooth function, like a cubic smoothing spline or a kernel smoother. There's an algorithm for simultaneously estimating all functions over features.
            3. The logit can be seen as a link function, which is a transformation that takes the sum over all of the input features into account and then becomes one feature in the model.
            4. The algorithm is fit by the backfitting algorithm, where there's a cylcel betwen each function over features and the global function until each function changes less than a given threshold.
- Boosting and Additive Trees
- Neural Networks
    a. Projection Pursuit Regression

- Support Vector Machines and Flexible Discriminants
    a. Support Vector Classifier
        i. Introduce slack variables and modify the constraint to include slack variables for datapoints that are misclassified by the margin (while still maximizing the margin between the classes).
        ii. Optimized via quadratic programming using lagrange multipliers.
        iii. There are a set of datapoints with nonzero weight that represent the solution which are called the support vectors, the only datapoints used to solve for the margin.
        iv. You give the SVM a kernel transformation, and it outputs a decision boundary - not a prediction for each value. The kerel considers an inner product between each datapoint in the dataset. That allows for efficient computation across those inner products, where your decision boundary is a function of the same number of features regardless of how large your original feature space was.
    b. Flexible Discriminant Analysis
        i. Performing LDA using linear regression on derived response variables. You predict on transformed labels,
        ii. LDA can be perfmed by a sequence of linear regressions, followed by classification to the closest class centroid in the space of linear regression fits.
        iii. The linear regression fits can be replaced by nonlinear or nonparametric fits, and so generate a more flexible classifier than LDA.
- Prototype Methods and Nearest-Neighbors
    a. Prototype Methods (K-means, Vector Quantization, Gaussian Mixtures)
    b. Adaptive Nearest-Neighbor Methods
        i. Nearest-neighbor neighborhoods are deformed, stretching out indirections where the class probabilities don't change. Often, class probabilities only change on a low dimensional subspace of a high dimensional space, and so adapting the distance metric is reasonable.
            1. This feels like PCA, looking at the dimensions that have minmal variance over the class labels and shrinking them. This suggestes a general preprocessing technique for high dimensional data, but seems very much like training on just the principal components of a representation. There's Principal Componenet Regression but not Principle Component kNN or k-means, perhaps there should be.
- Unsupervised Learning
    a. Association Rules
    b. Self-Organizing Maps

        i. Constrained version of k-Means clustering, where the cluster means (prototypes) are constrained to be on a one or two-dimensional manifold in the feature space. Neighbors are defined to be within some threshold and the optimization is via an update that moves the prototypes in the direction of the difference between the current observation (which is processed online) and the current prototype values.
    c. Non-negative Matrix Factorization
        i. Alternative to PCA when the data and the principal components are assumed to be non-negative. An example application is image data, where all datapoints are positive. Also proceeds by reconstruction error, modeling the data with a two matrices (W and H) that are found by maximizing a log-likelihood which assumes Poisson distributed data. The decomposed matrices W and H are alternately optimized in turn to return as close to a correct reconstruction of the data as possible.
            1. Ways in which the non-negativity impacts the decomposition:
                a. Use of the Poisson distribution
                b. Appliation to data like Images that tends to be non-negative
    d. Independent Components Analysis
        i. Like PCA, but rather than assuming dimensions are uncorrelation with one another (and so the covariances are determined by that) assumes statistical independence across every moment (where mean is the first, covariance is hte second, and so on). Can use the mutual informatio nbetween the components as a natural measure of dependence, and looks to minimize that measure of dependence. There's a negentropy measure that can be optimized for. FastICA is an approximatio nof this negentropy measure.
    e. Multidimensional Scaling
    f. PageRank
- Random Forests
    a. Variance and the De-Correlation Effect
        i. Shows the variance of the random forest model as the product of the correlation induced by the sampling distribution between pairs of trees used in the averaging and the variance of any singly drawn tree.
- Ensemble Learning
    a. Learning Ensembles
- Undirected Graphical Models
    a. Estimation of Parameters when the Graph Structure is Known
- High Dimensional Problems

a. Diagonal Linear Discriminant Analysis and Nearest Shrunken Centroids
b. Supervised Principal Components