Week 7 Discussion

CS M148 Section 1B
12 Nov 2021
Danning Yu

Slides adapted from Lionel Levine's slides.

Announcements

- **Do not come to class/discussion/campus if you are feeling sick, came into close contact with someone who tested positive for COVID-19, or tested positive yourself**

- Project 3 released
  - Due 12/3 (Friday Week 10)
- Homework 3 released
  - Due 11/30 at 11:59 PM (Tuesday Week 10)

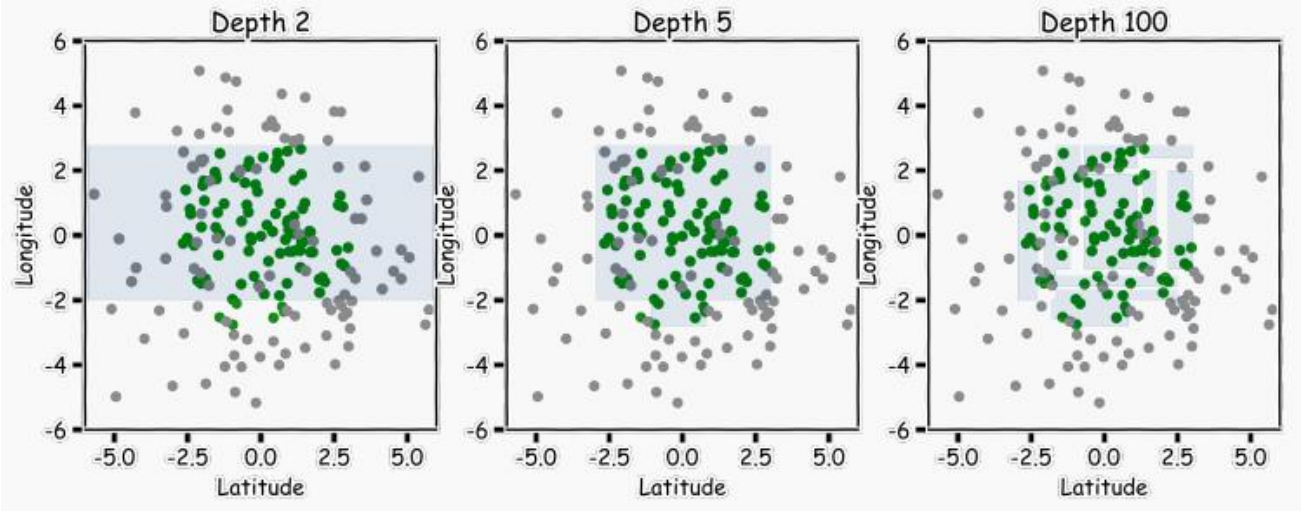
Lecture Recap

Lecture Recap

Bagging and Random Forest

Recall: Decision Tree (DT)

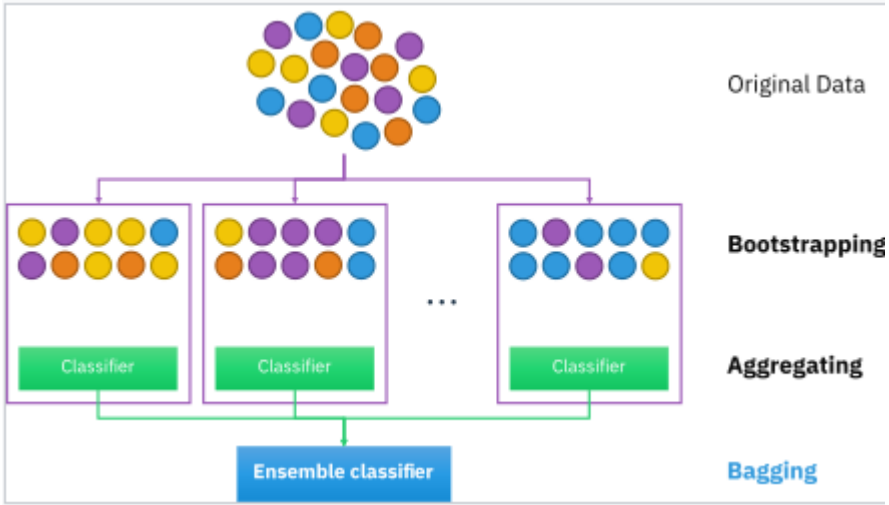
- Intuitive way to do classification or regression
- Locally linear, but complex overall decision boundary
- Downside: to capture complex decision boundary, need to use large tree
    - This causes overfitting

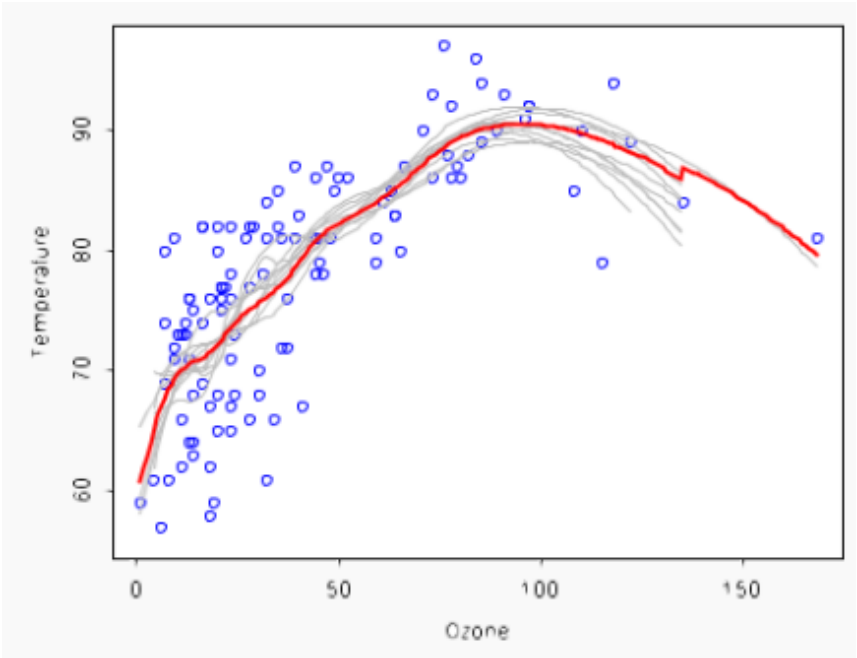


Bagging

- Create multiple decision trees on subsets of data and average the results
  - Averaging the results reduces variance
- Short for bootstrap aggregating
  - Bootstrap: generate multiple samples of data, and train full DT on each
  - Aggregate: to make prediction, run it through all the DTs and take their majority vote or average
- Usage of full trees allows for complex decision boundaries
- Downsides
  - The resulting model is less interpretable
  - The trees are very similar to each other
    - More on this later - random forest solves this issue
- Hyperparameters: number of trees, depth/complexity of each tree

Bagging


Original Data
Bootstrapping
Classifier
Classifier
Classifier
Aggregating
Ensemble classifier
Bagging


Temperature
60
70
80
90
0
50
100
150
Ozone

Out-of-Bag Error (1 of 2)

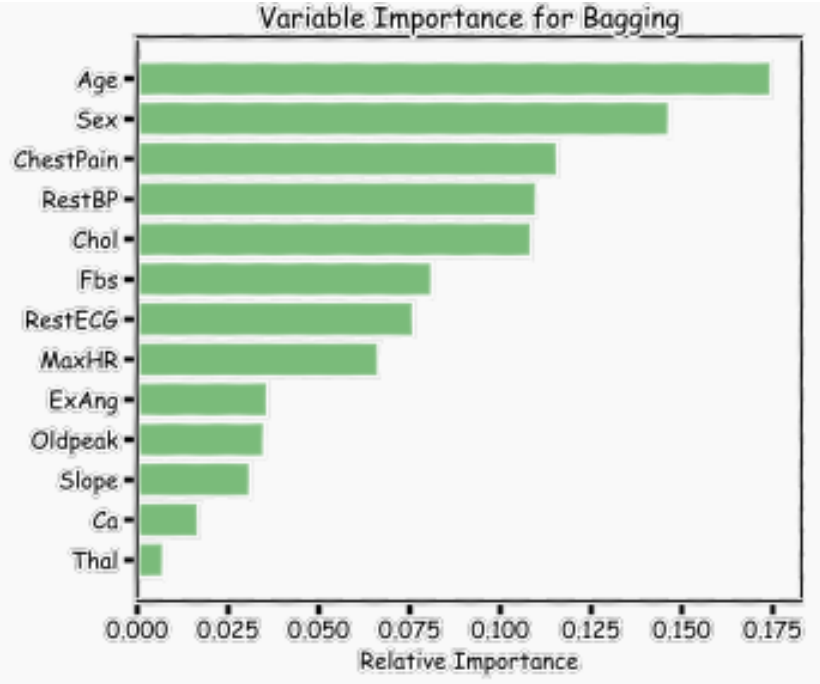
- Each individual decision tree only uses a subset of the training data
- Use the remaining data to measure the goodness of the model
- Point-wise out-of-bag error
  - For a point x_i, have all trees that did not see this point make a prediction on it
  - For classification, take majority vote over all trees; for regression, take the average
  - Then compare predicted value to actual value
- Overall out-of-bag error: average or combination of point-wise out-of-bag error across all points in training set

Out-of-Bag Error (2 of 2)

- Alternative way of measuring out-of-bag (OOB) error:
  - Record prediction accuracy on OOB samples for each tree (same as before)
  - Permute data in column *j* for OOB samples and remeasure prediction accuracy
    - Essentially, shuffle the data so that it's now meaningless
  - Greater change in prediction accuracy means predictor *j* was more important
- Key difference: measuring change in accuracy rather than absolute accuracy

Predictor Importance in Bagging

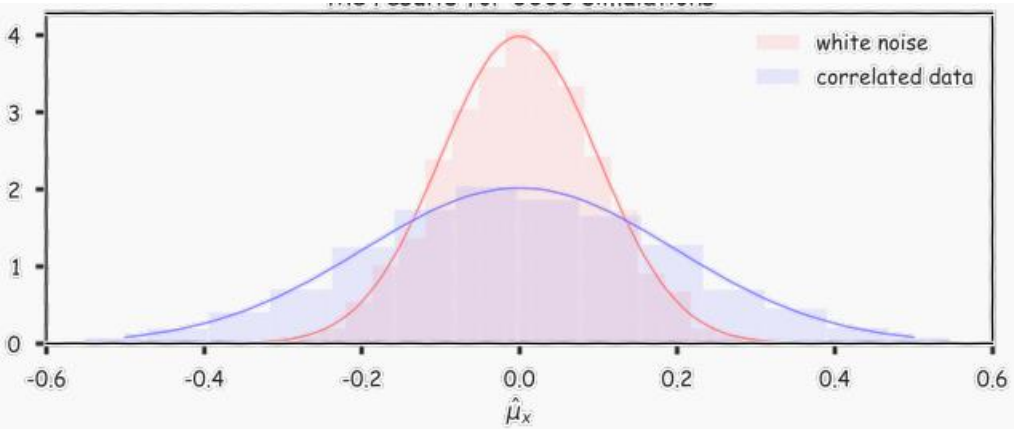
- A way to interpret the result of bagging since individual trees are no longer available
- Every time a predictor (variable) is use to make a split, calculate the gain
    - Higher gain means more important variable



$$Gain(R) = \Delta(R) = m(R) - \frac{N_1}{N} m(R_1) - \frac{N_2}{N} m(R_2)$$

Correlated Trees

- If there is a very strong predictor *j*, then because trees are constructed greedily, that predictor will tend to be split on early
- This creates similar trees
  - The original goal of bagging was to create different trees
- Averaging using correlated data has wider confidence intervals



Random Forest

- Just like bagging, except at each split, choose a subset of possible predictors to do the split on
  - Still train trees on a sample of the training data
- Hyperparameters
  - Number of predictors to choose from at each split
  - Those previously mentioned for bagging
  - Typically tuned through out-of-bag error
- Downsides
  - Does not work well if there are many predictors
    - The important predictors might not get included in the subset that we pick

Lecture Recap
Boosting

Boosting

- Use linear combination of simple models T_h to create a more complex model

$$T = \sum_h \lambda_h T_h$$

- Gradient boosting: for regression problems
  a. Create a simple model (weak learner) $T^{(0)}$, let $T = T^{(0)}$, and let $r^{(0)}$ be its errors (residuals)
  b. Set $i = 1$
  c. Fit another simple model $T^{(i)}$ to the residuals $r^{(i - 1)}$
  d. $T = T + \lambda T^{(i)}$, $r^{(i)} = r^{(i)} - \lambda T^{(i)}(x_n)$, $i = i + 1$
  e. Repeat (c) and (d) until a stopping condition is met
- Idea: at each step, train a model on the remaining error and add that to the overall model
- This is very similar to gradient descent!

Gradient Boosting Details

- Possible stopping conditions
  - Magnitude of residuals reaches a certain threshold
  - Limit maximum possible number of iterations
- Learning rate λ
  - Too small: model will take too long to converge
  - Too large: model might never converge
  - Variable rate λ: proportional to the gradient

Boosting for Classification

- Gradient boosting applied to classification
- Loss/error to optimize: misclassification rate
  - Not differentiable
- Replace with exponential loss:

$$\text{ExpLoss} = \frac{1}{N}\sum_{n=1}^{N}\exp\left(-y_n\hat{y}_n\right),\ y_n \in \{-1, 1\}$$

- Gradient of exponential loss w.r.t $\hat{y}_i$ (single point): $-y_i \exp(-y_i\hat{y}_i) = -y_i w_i$
- Update step: focusing on points that were misclassified:

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda w_n y_n, \qquad n = 1, \ldots, N$$

AdaBoost

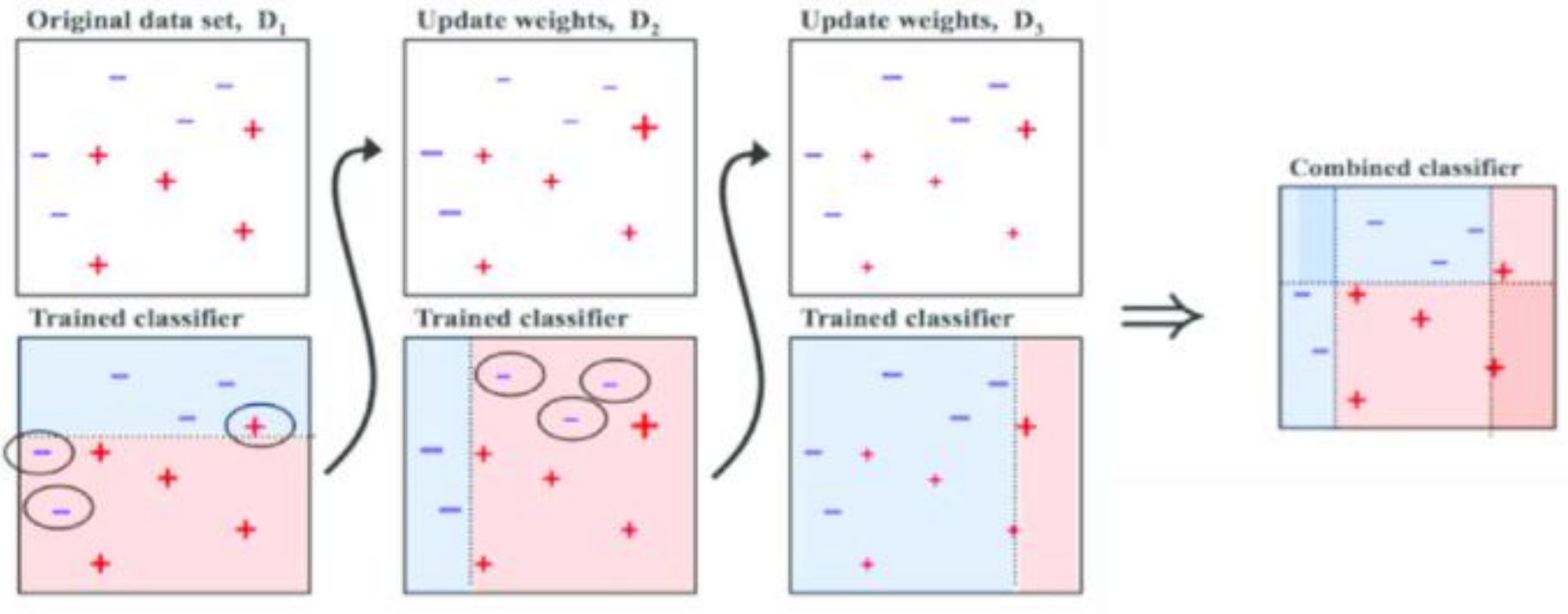
1. Choose initial distribution of w_i, such as $w_i = 1 / N$ for all i, or random values
2. Initialize $i = 0$
3. Fit a simple classifier $T^{(i)}$ on the data pairs $(x_i, w_i y_i)$, $i = 1...N$
4. Update the weights with the following equation

$$w_n \leftarrow \frac{w_n \exp(-\lambda^{(i)} y_n T^{(i)}(x_n))}{Z}$$

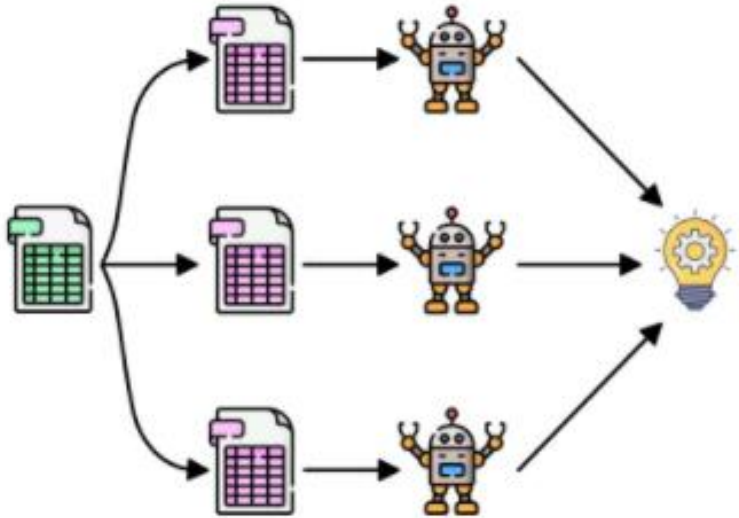
5. Update $T = T + \lambda T^{(i)}$

- Z is a normalizing constant, *λ is learning rate*

AdaBoost Example

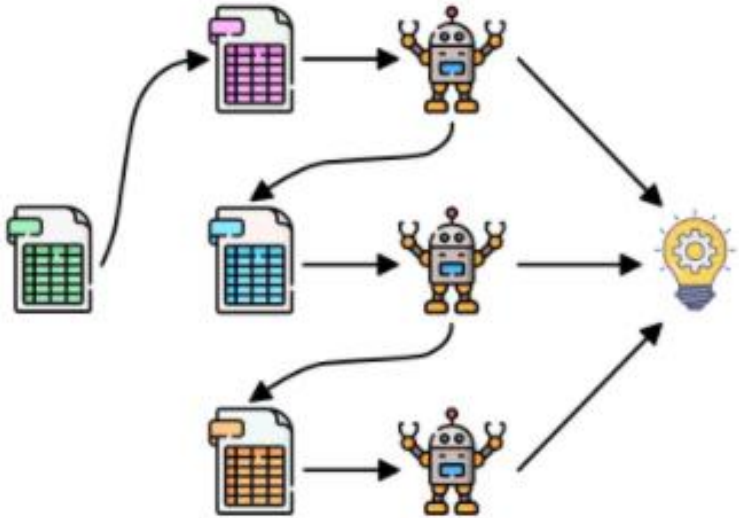


Bagging



Parallel

Boosting



Sequential

Project 3

Regression for Feature Selection

- Uses of feature selection
  - Model optimization
    - Feature trimming
  - Data collection
    - Identify and prioritize most important features during data collection
  - Market Analysis
    - Ongoing strategic planning

Code from here

```
In [39]: import statsmodels.api as sm
```

```
In [47]: # build the OLS model (ordinary least squares) from the training data
         avo_stats = sm.OLS(new_y, new_avocado_prepared)

         # do the fit and save regression info (parameters, etc) in results_stats
         results_stats = avo_stats.fit()
```

```
In [48]: print(results_stats.summary())
```

                            OLS Regression Results

Interpreting Output of `statsmodels.api.OLS`

- R-squared: goodness-of-fit measure for linear regression models. It indicates the percentage of the variance in the label that the features explain collectively. It measures the strength of the relationship between your model and the label on a 0–100% scale.
- F-Stat and T-Stat: An F-statistic is a value you get when you run an ANOVA test to find out if the means between two populations are significantly different. A T-test will tell you if a single variable is statistically significant and an F-test will tell you if a group of variables are jointly significant.
    - If you want to know whether your regression F-value is significant, you'll need to find the critical value in the f-table.
- P-Value: A p value is used in hypothesis testing to help you support or reject the null hypothesis. The p value is the evidence against a null hypothesis. The smaller the p-value, the stronger the evidence that you should reject the null hypothesis.

Curse of Dimensionality

- Adding irrelevant features improves model performance on training data
- Example: kernel functions
  - For every feature added, data dimensionality grows and it becomes sparser, so easier to find a separable hyperplane
  - Likelihood that a training sample lies on the wrong side of the best hyperplane becomes infinitely small when the number of dimensions becomes infinitely large
- Projecting the highly dimensional classification result back to a lower dimensional space shows that model is actually overfitted
- Also, overly rich feature sets in our data can make calculating a learning model intractable
- Integer data with range of 5: 1D has 5 possible values, 2D has 25 possible values, 3D has 125 possible values
  - Data rapidly becomes sparse, need exponential increase in data to keep up

Principal Component Analysis

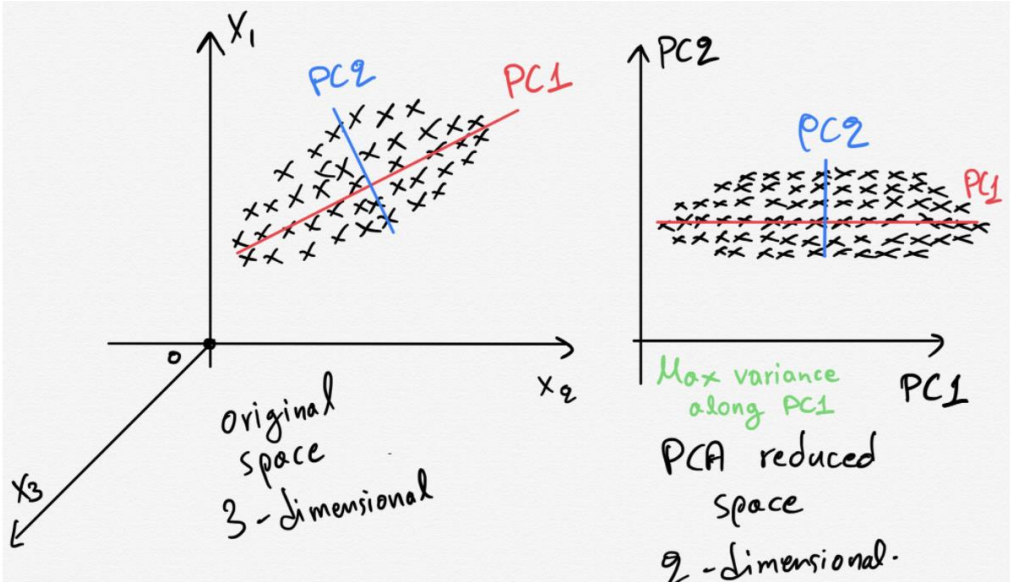
- A technique used to take a large list of interdependent variables and choose the subset that best suit a model
- Focusing in on only a few variables is called dimensionality reduction and helps reduce complexity of our dataset
    - Principal components serve to "summarize" data
- Derives components, made up of multiple predictors, that are linearly independent with respect to each other
- Types of dimensionality reduction
    - Feature elimination: prune features from a dataset, losing information held in those dropped features
    - Feature extraction: create new variables by combining existing features, reducing simplicity and interpretability but allowing for all important information to be kept
        - This is what PCA does

Why Not Feature Elimination

- Deleting features assumes all features are statistically independent,
  - In practice, many features are correlated and dependent on each other or on an underlying unknown variable.
- A single feature could therefore represent a combination of multiple types of information by a single value
  - Removing such a feature would remove more information than needed.
- Before eliminating features, we need to transform the complete feature space such that the underlying uncorrelated components are obtained
  - This is the motivation for PCA

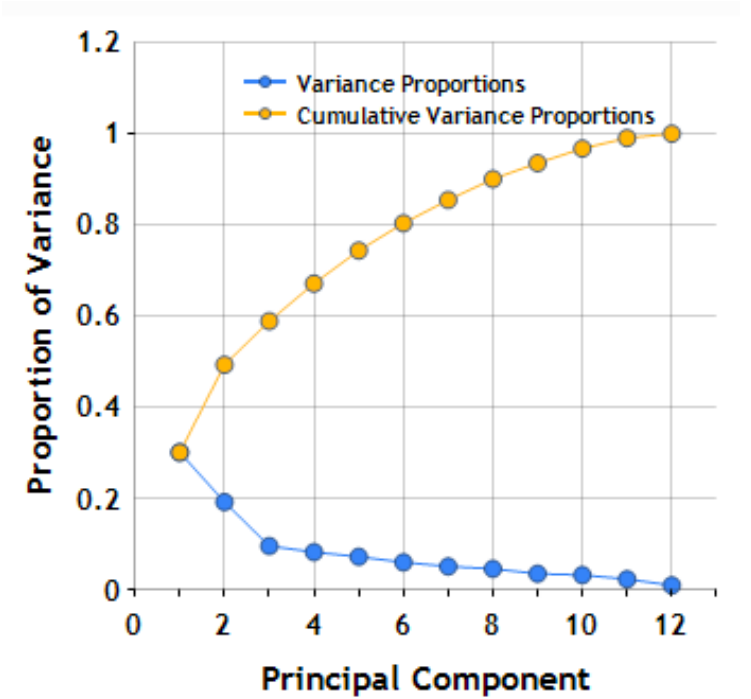
Principal Components

- Mutually orthogonal to each other
- Each successive principal component (PC) captures an increasing cumulative amount of variation in the data (when combined with previous PCs)



Calculating PCA

- Find the eigenvectors of a covariance matrix of the data, and sort them by their corresponding eigenvalues in descending order
- Highest eigenvalues means corresponding eigenvector captures the highest degree of variance in the data
- Each eigenvector is a basis vector for the new feature space
- Typically only use a subset of all eigenvectors to explain a certain threshold of variance (typically 0.8 or 0.9)
    - If you used all eigenvectors, you'd just get the original data back!



General Project 3 Guidance

- Data from the real world: experience what data science with real data is like
- General advice: leverage basic regression models you've already covered
- Use the 'time-series' element of the data to engineer chronological features, but then plug that data into standard regression models
  - We don't intend for you to do time series regression/prediction
- Look at both timeseries and brand features
- Note: There is a mismatch between sales data (at a brand level) and product data
  - Part of the challenge is you'll have to aggregate product features
- Start early on the project!

Possibly Useful Features

- Time series features
  - Rolling averages
  - Percent changes and month-to-month fluctuations
  - Year
  - Season/period
  - Aggregate sales and trends for previous timeframe
  - Length of time in market
  - Market share of the brand over time
- Brand features that might be useful
  - Product mix
  - Specific product categories
  - Number and variety of offerings
  - Particular strains/growths
  - Brand strength
  - Retail presence

Homework 3

Perceptron Model (1 of 3)

- A simple classifier model inspired by neurons in biology
  - Multiple signals arrive at a neuron, are integrated together, and if accumulated signal exceeds a certain threshold, an output signal is generated
- Model will always converge if data is linearly separable
  - Otherwise, it will never converge
- If data is not linearly separable, need multilayer perceptron
  - This is known today as a neural network model

Perceptron Model (2 of 3)

- Each neuron:
  - Takes inputs $\mathbf{x} = [1 \;\; x_1 \;\; x_2 \; ...]^T$
  - Weighs them separately with $\mathbf{w} = [w_0 \;\; w_1 \;\; w_2 \; ...]^T$
  - Sums them together: $\mathbf{w}^T\mathbf{x}$
  - Passes this sum through a nonlinear ‘activation’ function $f(x)$ to produce output
- Activation functions are used to map the output between required values such as [0, 1] or [-1, 1]

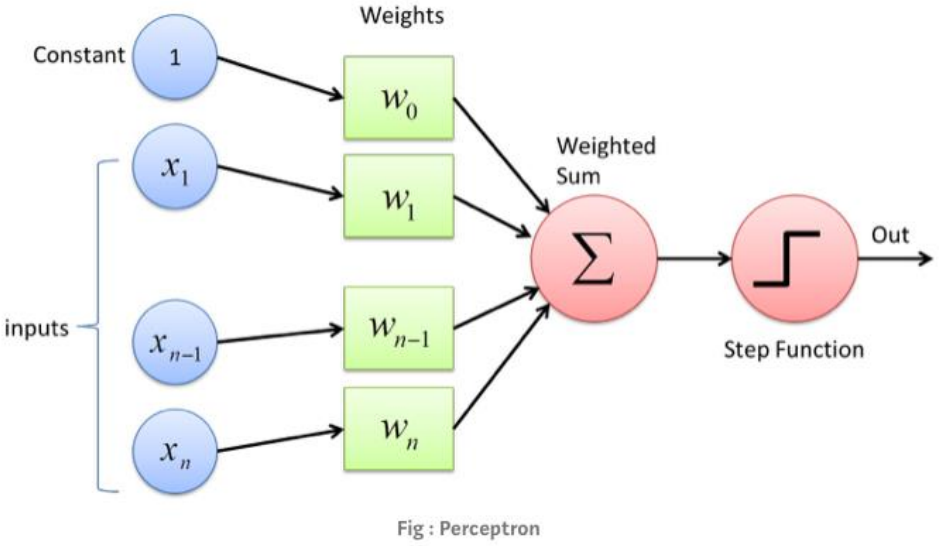


Fig : Perceptron

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron Model (3 of 3)

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Update rule: $\Delta\mathbf{w} = c(t - z)\mathbf{x}$ or $\Delta\mathbf{w} = c(y - \hat{y})\mathbf{x} = c(y - f(\mathbf{x}))\mathbf{x}$
  - $\mathbf{x}$ is input; $\mathbf{w}$ is weight vector
  - c is learning rate (similar to λ in gradient descent)
  - t or y is target value; z or $\hat{y}$ is predicted value
- $\mathbf{w}$ only changes if $y \neq \hat{y}$
- Amount of change in $\mathbf{w}$ is proportional to $\mathbf{x}$
- Perceptron training algorithm
  - Initialize weights $\mathbf{w}$ to $\mathbf{0}$ or random values
  - Pick point from training set and apply perceptron update rule
    - This is an epoch
  - Repeat previous step until $\mathbf{w}$ no longer changes (you might reuse data)
    - This is called convergence
- Perceptron Convergence Theorem: guaranteed to find a solution in finite time if a solution exists

Have a nice weekend!