

A thick black L-shaped frame is positioned around the text. It starts at the top-left, goes right, then down, then right again, and finally down to the bottom-right corner.

MACHINE LEARNING IN R

GradQuant Workshop, February 27th, 2018

Overview

- Fundamentals of Machine Learning
- Regression tasks
- Classification tasks
- Deep Learning with TensorFlow

Why is this workshop a little different?

- I usually like to run code as part of the workshop
- Many machine learning tasks take too long to run as part of the workshop
 - *Some models took three episodes of Twin Peaks to run!*
 - *That would make for a very awkward workshop*
- This is meant as a introduction to an introduction to an overview of ML

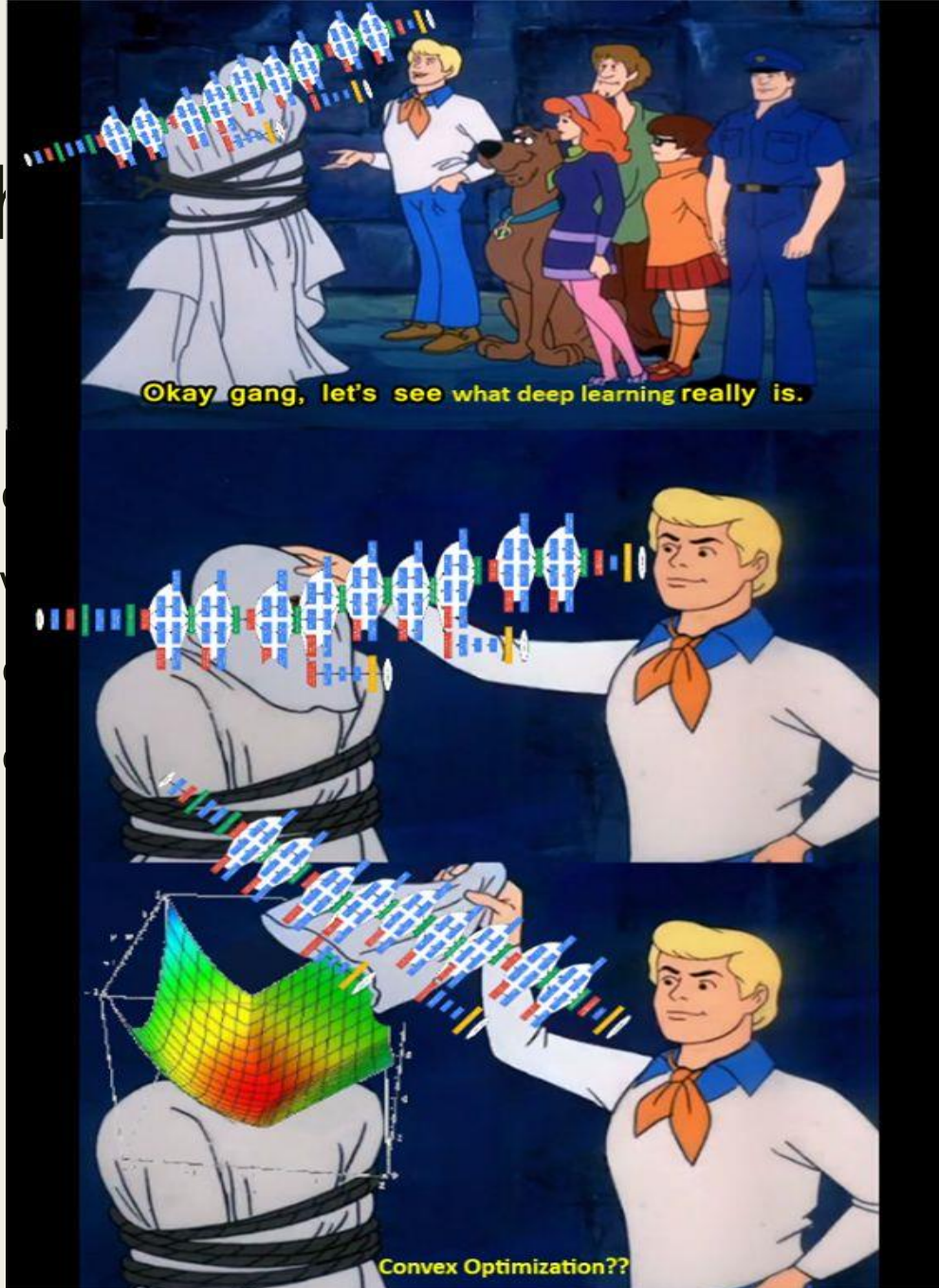
What is Machine Learning?

- Computer science field devoted to learning with data
- Evolved out of desire for AI and computational learning
- Related to computational statistics, and heavily utilizes numerical optimization
- Related to, but not equivalent to, data mining
 - *Data mining is more exploratory*

What

ing?

- C
- E
- R
- R



h data

learning

y utilizes numerical optimization

Prediction vs. Understanding

- The goal of many research projects is to understand processes influencing our outcome
- Goal of machine learning is to predict our outcome
 - *This often includes a tradeoff with interpretation*
- For example, consider the creepiness of Facebook photos
- Many of these methods are ‘black boxes’

Separating Test and Training Data

- Machine Learning involves the use of two datasets
 - *Training and test data*
- The use of test data ensures that we train and test our model on different data points
- Training data often give us an overly optimistic view of our model
- Test data provide a more accurate view of our model performance
- Important to occasionally update our model with new data to avoid rot

Preprocessing

- Some decisions about the data need to be made before constructing models
- Some models prefer data to be on a 0-1 scale, some prefer z-scores, others don't care
- Most models will blow up your computer if there are missing data
- Some other issues to consider, such as...

Low variance

- Imagine a variable with 98% of the sample being a 1, the other 2% being a 0
 - *This will likely unduly influence our model*
- R uses a 10% rule to determine low variance predictors
- Often the best (and easiest) solution is to delete variables
- But may be usable if the training data set is sufficiently large

High correlation

- Many models will also have trouble optimizing if variables are too related
- R can find those variables that are highly correlated
- Again, we can chose to delete these variables
 - *Best solution with perfect collinearity (i.e. columns for Employed and Unemployed)*
- We can also preprocess by using dimensionality reduction (e.g. PCA)

Over/Underfitting

- Our training model can either over or under fit the data
 - *Ideal is a model that performs well on both datasets*
- Overfitting occurs when our model performs well on training data, but poor on test data
 - *May want to try a regularization technique, or a simpler model*
- Underfitting occurs when our model performs poorly on training data
 - *Try a more complex model, or different predictors*

Tuning Parameters

- Most models have some modifications that can be made
- R will automatically search a limited subspace of tuning parameters
 - *We can also tell it what to use*
- Finding optimal tuning parameters also part of model training
- I will point out relevant tuning parameters, but forego exhaustive search

Regression Tasks

- Linear Regression
- Regularization techniques
- Multivariate Adaptive Regression Splines
- K-Nearest Neighbors
- Regression Trees
- Random Forests
- Identifying the most important predictors

Measuring Performance in Regression Tasks

- Two main statistics when predicting continuous outcomes
- RMSE is equal to the average misprediction of our outcomes values

- $RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$

- R^2 is the amount of variability in our outcome that our model is explaining
 - $R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$

Wine Quality Dataset

- Wine quality dataset from the UCI Machine Learning Repository
- 1,599 ratings of quality of different wines
 - *1-8 scale, with 1 being undrinkable and 8 being amazing*
- 11 predictors of wine quality
 - *e.g. Chlorides, ABV*
- Our task is to create a model that adequately predicts

Wine Quality Dataset

```{r}  
winequality  
```

fixed acidity <dbl>	volatile acidity <dbl>	citric acid <dbl>	residual sugar <dbl>	chlorides <dbl>	free sulfur dioxide <dbl>
7.4	0.700	0.00	1.90	0.076	11.0
7.8	0.880	0.00	2.60	0.098	25.0
7.8	0.760	0.04	2.30	0.092	15.0
11.2	0.280	0.56	1.90	0.075	17.0
7.4	0.700	0.00	1.90	0.076	11.0
7.4	0.660	0.00	1.80	0.075	13.0
7.9	0.600	0.06	1.60	0.069	15.0
7.3	0.650	0.00	1.20	0.065	15.0
7.8	0.580	0.02	2.00	0.073	9.0
7.5	0.500	0.36	6.10	0.071	17.0

1-10 of 1,599 rows | 1-6 of 12 columns

Previous 2 3 4 5 6 ... 100 Next


```
wine <- read.csv('winequality.csv')
trainrows <- createDataPartition(wine$quality, p = .8, list = FALSE)
winetrain <- wine[trainrows,]
winetest <- wine[-trainrows,]
dim(winetrain)

## [1] 1281  12

dim(winetest)

## [1] 318  12
```

Linear Regression

- Linear regression is often used as the “baseline” or default model
 - *Other models will be compared this regression*
- Objective is to find the plane that most reduces the Sums of Squares Error (SSE)
 - $SSE = \sum_{i=1}^n (y_i - \check{y}_i)^2$
- The optimal plane of parameters is:
 - $(X^T X)^{-1} X^T y$
 - *Returns a vector of the optimal parameters*

Linear Regression Results

```
linreg <- train(quality ~ ., data = winetrain, method = 'lm')
linreg

## Linear Regression
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results:
##
##      RMSE      Rsquared    MAE
## 0.8528505 0.2383112 0.5082964
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

linregpred <- predict(linreg, newdata = winetest)
R2(linregpred, winetest$quality)

## [1] 0.2112612

RMSE(linregpred, winetest$quality)

## [1] 0.8782964
```

Lasso Regression

- Least Absolute Shrinkage and Selection Operator
- Imposes a penalty on the model parameters
 - *It both shrinks the regression parameters, and sets some to zero*
 - *This shrinking usually reduces the degree of overfitting*
- This achieves both regularization and variable selection
- $SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^P |\beta_j|$
 - *Where λ is the shrinking parameter*

Lasso Regression Results

```
lasso <- train(quality ~ ., data = winetrain, method = 'lasso')
lasso

## The lasso
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results across tuning parameters:
##
## fraction RMSE      Rsquared  MAE
## 0.1      0.8558870  0.1728793 0.6343735
## 0.5      0.7543567  0.2501568 0.5191820
## 0.9      0.7418589  0.2639738 0.4989601
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.

lassopred <- predict(lasso, newdata = winetest)
R2(lassopred, winetest$quality)

## [1] 0.235743

RMSE(lassopred, winetest$quality)

## [1] 0.7699115
```

Multivariate Adaptive Regression Spline

- Creates a piecewise linear model
 - *This can identify nonlinearities in the predictors*
- Each variable is broken into two at a “hinge”
 - *Hinge function: $h(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$*
- This allows to estimates different slopes at different levels of predictors

MARS Result

```
earth <- train(as.numeric(quality)~ . , data = winetrain, method = 'earth')
earth

## Multivariate Adaptive Regression Spline
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results across tuning parameters:
##
##   nprune  RMSE      Rsquared  MAE
##    2      0.7131664  0.2054139  0.5677883
##    9      0.6674768  0.3214609  0.5108443
##   17      0.6787601  0.3157428  0.5109675
##
## Tuning parameter 'degree' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 9 and degree = 1.

earthpred <- predict(earth, newdata = winetest)
RMSE(earthpred, winetest$quality)

## [1] 0.6528861

R2(earthpred, winetest$quality)

##           y
## [1,] 0.3716706
```

K-Nearest Neighbors

- Among the machine learning algorithm
- Predicts the value of a new sample using the k nearest neighbors
 - *User specifies k , or tries several values*
- Does not lend itself to a clear model specification, but is built upon the surrounding data points
- The predicted value becomes the mean of those k neighbors

KNN Results

```
knn <- train(quality ~ ., data = winetrain, tuneLength = 15, method = 'knn')
knn

## k-Nearest Neighbors
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results across tuning parameters:
##
##  k  RMSE      Rsquared  MAE
##  5  0.7968283  0.1305966  0.6034272
##  7  0.7780717  0.1315245  0.5995070
##  9  0.7689441  0.1305925  0.5972475
## 11  0.7659321  0.1265367  0.5967810
## 13  0.7632057  0.1247015  0.5952672
## 15  0.7624019  0.1204451  0.5949367
## 17  0.7596138  0.1214243  0.5941398
## 19  0.7590425  0.1201026  0.5929951
## 21  0.7586970  0.1186734  0.5936998
## 23  0.7580342  0.1181360  0.5931863
## 25  0.7569988  0.1186653  0.5933399
## 27  0.7564540  0.1182458  0.5933587
## 29  0.7561103  0.1175893  0.5938933
## 31  0.7567026  0.1155202  0.5945899
## 33  0.7563577  0.1153775  0.5949910
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 29.
```

KNN Results

```
knn <- train(quality ~ ., data = winetrain, tuneLength = 15, method = 'knn')
knn

## k-Nearest Neighbors
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results across tuning parameters:
##
##  k  RMSE      Rsquared  MAE
##  5  0.7968283  0.1305966  0.6034272
##  7  0.7780717  0.1315245  0.5995070
##  9  0.7689441  0.1305925  0.5972475
## 11  0.7659321  0.1265367  0.5967810
## 13  0.7632057  0.1247015  0.5952672
## 15  0.7624019  0.1204451  0.5949367
## 17  0.7596138  0.1214243  0.5941398
## 19  0.7590425  0.1201026  0.5929951
## 21  0.7586970  0.1186734  0.5936998
## 23  0.7580342  0.1181360  0.5931863
## 25  0.7569988  0.1186653  0.5933399
## 27  0.7564540  0.1182458  0.5933587
## 29  0.7561103  0.1175893  0.5938933
## 31  0.7567026  0.1155202  0.5945899
## 33  0.7563577  0.1153775  0.5949910
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 29.
```

```
knnpred <- predict(knn, newdata = winetest)
RMSE(knnpred, winetest$quality)

## [1] 0.782951

R2(knnpred, winetest$quality)

## [1] 0.1021323
```

Regression Tree

- Prediction is included with this model as a series of nested if-then statements
- If: $x_1 < 20$ then $y = 5.3$
 - *Else If: $x_2 > 50$ then $y = 7.6$*
 - *Else $y = 12.6$*
- We can differ how many potential branches we have
- Trees will find the optimum branches and cutoffs to predict y
- Single trees are VERY prone to overfitting

Regression Tree Results

```
library(rpart)
tree <- rpart(quality ~ ., data = winetrain)
tree

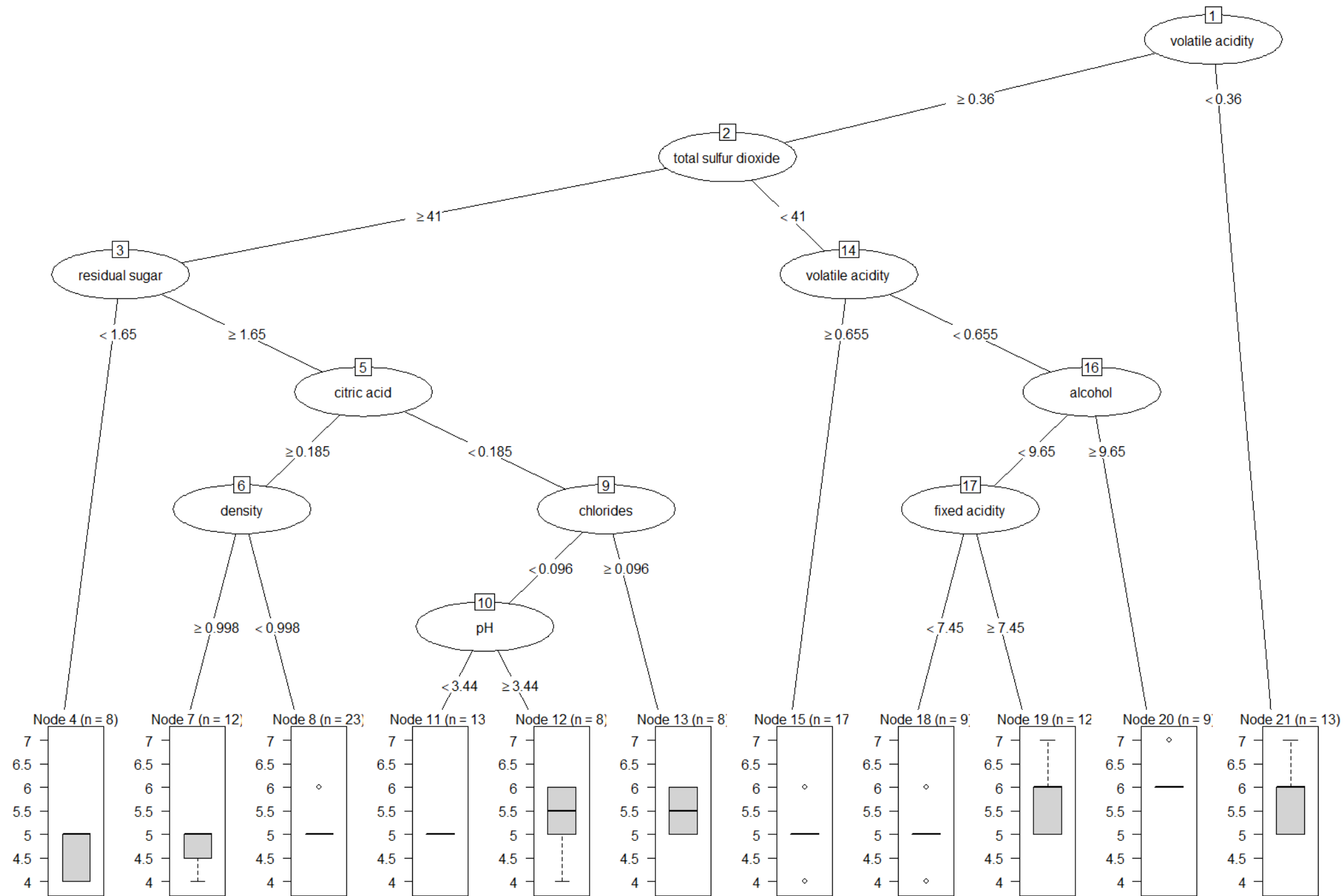
## n= 1281
##
## node), split, n, deviance, yval
##      * denotes terminal node
##
## 1) root 1281 826.47930 5.636222
##    2) alcohol< 10.525 781 336.05890 5.370038
##      4) sulphates< 0.575 318 107.49690 5.163522 *
##      5) sulphates>=0.575 463 205.68470 5.511879
##        10) volatile.acidity>=0.4175 332 125.91570 5.403614 *
##        11) volatile.acidity< 0.4175 131 66.01527 5.786260
##          22) sulphates< 0.675 55 15.74545 5.490909 *
##          23) sulphates>=0.675 76 42.00000 6.000000 *
##    3) alcohol>=10.525 500 348.64800 6.052000
##      6) sulphates< 0.645 225 155.79560 5.715556
##        12) volatile.acidity>=1.015 9 6.00000 4.000000 *
##        13) volatile.acidity< 1.015 216 122.20370 5.787037
##          26) alcohol< 11.45 114 54.35965 5.535088 *
##          27) alcohol>=11.45 102 52.51961 6.068627 *
##    7) sulphates>=0.645 275 146.54550 6.327273
##      14) alcohol< 11.55 164 87.77439 6.140244
##        28) volatile.acidity>=0.395 87 31.60920 5.873563 *
##        29) volatile.acidity< 0.395 77 42.98701 6.441558
##          58) pH>=3.255 46 21.21739 6.130435 *
##          59) pH< 3.255 31 10.70968 6.903226 *
##    15) alcohol>=11.55 111 44.55856 6.603604 *
```

```
treepred <- predict(tree, newdata = winetest)
RMSE(treepred, winetest$quality)

## [1] 0.6619079

R2(treepred, winetest$quality)

## [1] 0.3547117
```



Bagging

- Bootstrap AGGregating
- Designed to reduce variance in estimates and avoid overfitting the data
- A model averaging approach
- Take m data sets with resampling from the original data
- Perform a regression tree on each bootstrapped data set
- Average the results for regression trees

Random Forests

- Bootstrapped samples may feature different observations, but have the same variables included
- This introduces “tree correlation” among the samples
- Random forests use bootstrapping, but also random selection of predictors
 - *This eliminates correlation across different samples*
- While decision trees are easily interpreted, random forests are not

Random Forest Results

```
|forest <- train(as.numeric(quality)~ . , data = winetrain, method = 'rf')
forest

## Random Forest
##
## 1281 samples
## 11 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1281, 1281, 1281, 1281, 1281, 1281, ...
## Resampling results across tuning parameters:
##
## mtry RMSE Rsquared MAE
## 2 0.6055493 0.4411636 0.4508052
## 6 0.6065994 0.4336613 0.4470510
## 11 0.6139803 0.4201776 0.4499041
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.

forestpred <- predict(forest, newdata = winetest)
RMSE(forestpred, winetest$quality)

## [1] 0.5757686

R2(forestpred, winetest$quality)

## [1] 0.5273906
```


Summary of Regression Results

Model	R ²	RMSE
Linear model	.211	.878
LASSO	.235	.770
MARS	.371	.653
KNN	.102	.783
Regression tree	.354	.662
Random Forest	.527	.575

Most Important Predictors

```
varImp(forest)
```

##	Overall
## fixed.acidity	35.92760
## volatile.acidity	50.32300
## citric.acid	33.68082
## residual.sugar	23.39666
## chlorides	36.97330
## free.sulfur.dioxide	32.22192
## total.sulfur.dioxide	49.06415
## density	41.97915
## pH	30.15177
## sulphates	63.03920
## alcohol	76.66128

Classification Tasks

- Logistic Regression
- Naïve Bayes
- KNN
- Linear Discriminant Analysis
- Support Vector Machines
- Decision Trees
- Random Forests

German Credit Data

- From UCI Machine Learning Repository
- 1000 observations of 62 variables
- Prediction is whether a person has good or bad credit

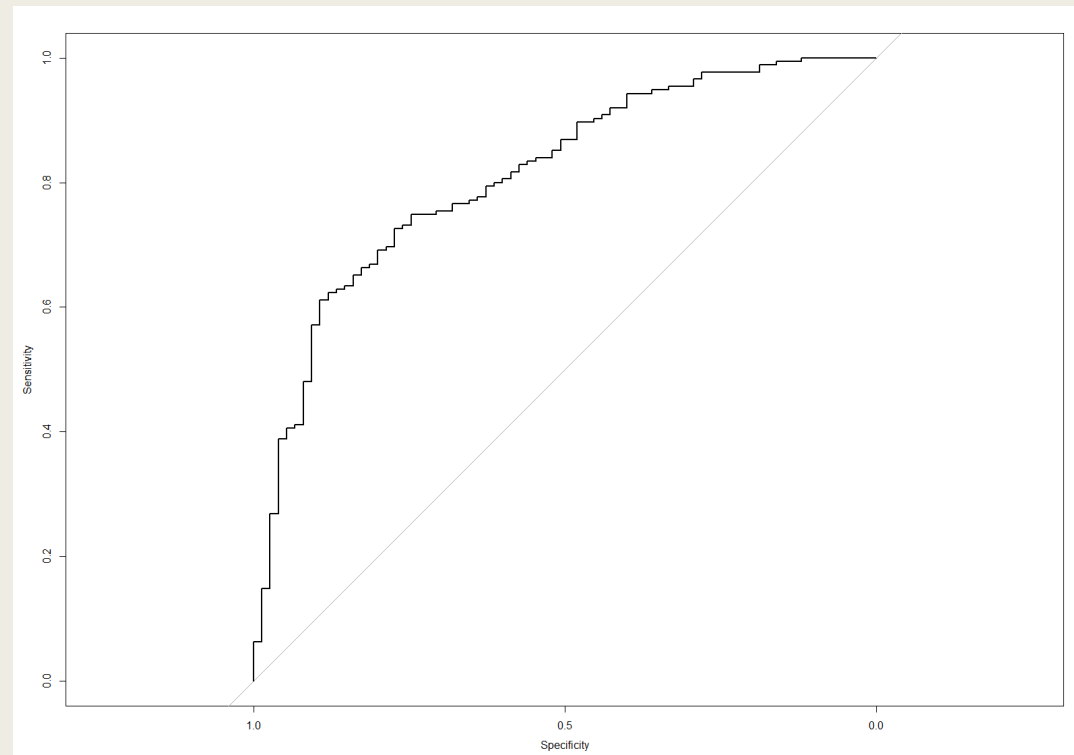
German Credit Data

tailmentRatePercentage	ResidenceDuration	Age	NumberExistingCredits	NumberPeopleMaintenance	Telephone	ForeignWorker	Class
4	4	67	2	1	0	1	Good
2	2	22	1	1	1	1	Bad
2	3	49	1	2	1	1	Good
2	4	45	1	2	1	1	Good
3	4	53	2	2	1	1	Bad
2	4	35	1	2	0	1	Good
3	4	53	1	1	1	1	Good
2	2	35	1	1	0	1	Good
2	4	61	1	1	1	1	Good
4	2	28	2	1	1	1	Bad
3	1	25	1	1	1	1	Bad
3	4	24	1	1	1	1	Bad
1	1	22	1	1	0	1	Good
4	4	60	2	1	1	1	Bad
2	4	28	1	1	1	1	Good

Measuring Performance in Classification

- Accuracy is a measure of how many predictions were right, versus the total dataset
 - *Many problems with this metric*
- $Sensitivity = \frac{\# \text{ Number of correct Yes's}}{\text{Number of Yes's}}$
- $Specificity = \frac{\# \text{ Number of correct No's}}{\text{Number of No's}}$
- Tradeoff between Sensitivity and Specificity

Area under ROC Curve



Logistic Regression

- Same basic concept as linear regression
- Form a model that is linear in the parameters by minimizing residuals
- Model the log odds of an event occurring
 - $\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \dots \beta_k x_k$
 - *Where p is the probability of the event occurring*
- Similar to linear regression, this often used as our baseline model

Logistic Regression Results

```
logistic <- glm(Class ~ ., data = CreditTrain, family = 'binomial')
logistic

##
## Call:  glm(formula = Class ~ ., family = "binomial", data = CreditTrain)
##
## Degrees of Freedom: 749 Total (i.e. Null);  706 Residual
## Null Deviance:      916.3
## Residual Deviance: 641.5    AIC: 729.5

credpred <- predict(logistic, newdata = CreditTest, type = 'response')
|roc(response = CreditTest$Class, predictor = credpred)

##
## Call:
## roc.default(response = CreditTest$Class, predictor = credpred)
##
## Data: credpred in 75 controls (CreditTest$Class Bad) < 175 cases
(CreditTest$Class Good).
## Area under the curve: 0.7255
```

Naïve Bayes

- Uses Baye's theorem to establish probability of outcome based in prior evidence
- $p(Event | \mathbf{x}) = \frac{p(Event)*p(\mathbf{x} | Event)}{p(\mathbf{x})}$
- Train the parameters of this equation on the test set, apply to the training data
- Considering the simplicity of this model, does surprisingly well

Naïve Bayes Results

```
naive <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),  
              method = 'naive_bayes')
```

```
naive
```

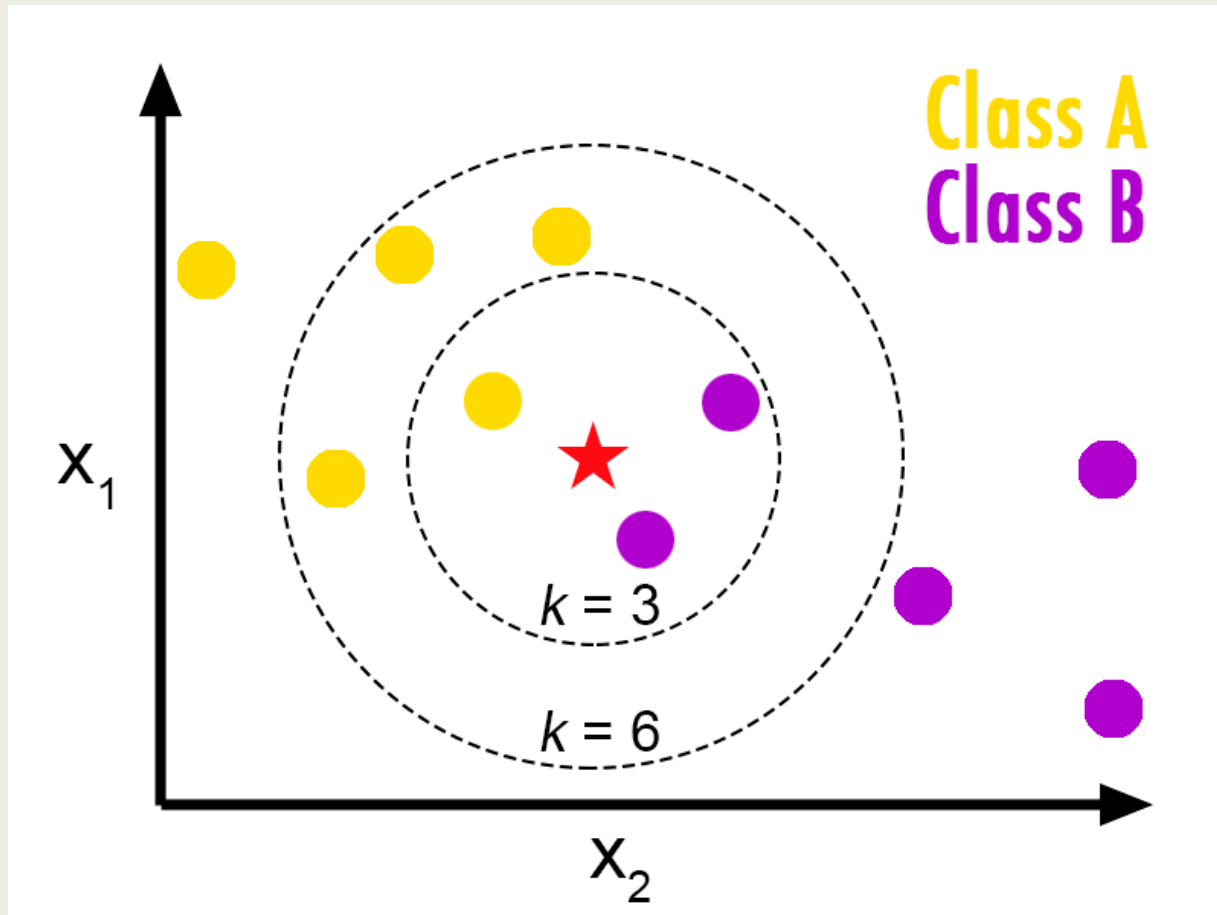
```
## Naive Bayes  
##  
## 750 samples  
## 49 predictor  
## 2 classes: 'Bad', 'Good'  
##  
## Pre-processing: centered (49), scaled (49)  
## Resampling: Bootstrapped (25 reps)  
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...  
## Resampling results across tuning parameters:  
##  
## usekernel Accuracy Kappa  
## FALSE 0.7112586 0.363746290  
## TRUE 0.6968461 0.001733411  
##  
## Tuning parameter 'laplace' was held constant at a value of 0  
##  
## Tuning parameter 'adjust' was held constant at a value of 1  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were laplace = 0, usekernel =  
## FALSE and adjust = 1.
```

```
naivepred <- predict(naive, newdata = CreditTest, type = 'prob')  
roc(response = CreditTest$Class, predictor = naivepred$Bad)  
  
##  
## Call:  
## roc.default(response = CreditTest$Class, predictor = naivepred$Bad)  
##  
## Data: naivepred$Bad in 75 controls (CreditTest$Class Bad) > 175 cases  
## (CreditTest$Class Good).  
## Area under the curve: 0.6957
```

KNN

- The same idea of KNN for regression can be used for classification
- For a new sample, use the label associated with nearest k neighbors
- Unlikely that all neighbors agree, so voting is typically used among k neighbors
- This is a simple model, but easily interpretable

KNN



KNN Results

```
knn <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),  
            method = 'knn', tuneLength = 10)
```

```
## k-Nearest Neighbors
```

```
##
```

```
## 750 samples
```

```
## 49 predictor
```

```
## 2 classes: 'Bad', 'Good'
```

```
##
```

```
## Pre-processing: centered (49), scaled (49)
```

```
## Resampling: Bootstrapped (25 reps)
```

```
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
```

```
## Resampling results across tuning parameters:
```

```
##
```

##	k	Accuracy	Kappa
##	5	0.6878687	0.2149655
##	7	0.7028838	0.2328485
##	9	0.7071430	0.2302960
##	11	0.7096588	0.2276259
##	13	0.7175363	0.2402718
##	15	0.7194484	0.2365087
##	17	0.7252919	0.2467146
##	19	0.7248848	0.2408036
##	21	0.7276833	0.2412301
##	23	0.7266859	0.2355054

```
##
```

```
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final value used for the model was k = 21.
```

```
knnpred <- predict(knn, newdata = CreditTest, type = 'prob')  
roc(response = CreditTest$Class, predictor = knnpred$Bad)
```

```
##
```

```
## Call:
```

```
## roc.default(response = CreditTest$Class, predictor = knnpred$Bad)
```

```
##
```

```
## Data: knnpred$Bad in 75 controls (CreditTest$Class Bad) > 175 cases  
## (CreditTest$Class Good).
```

```
## Area under the curve: 0.7382
```

Linear Discriminant Analysis

- Attempts to find a linear combination of the predictors that will best distinguish classes
- Often used as a dimensionality reduction technique as well
- Among the oldest classification technique

LDA Results

```
lda <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),
             method = 'lda2', tuneLength = 10)

ldapred <- predict(lda, newdata = CreditTest, type = 'prob')
lda

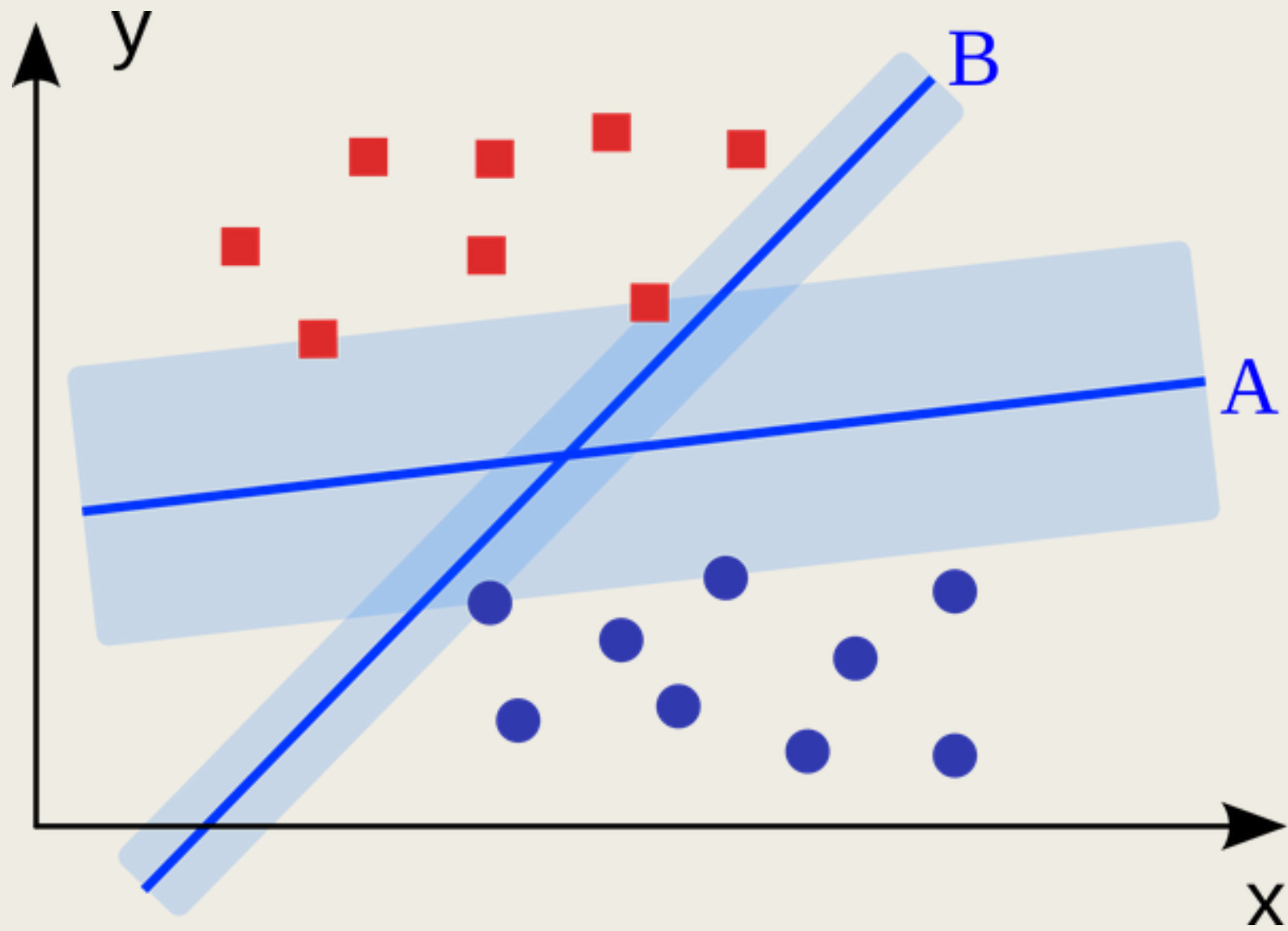
## Linear Discriminant Analysis
##
## 750 samples
## 49 predictor
## 2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (49), scaled (49)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
## Resampling results:
##
##   Accuracy   Kappa
## 0.7419048 0.353103
##
## Tuning parameter 'dimen' was held constant at a value of 1
```

```
roc(response = CreditTest$Class, predictor = ldapred$Bad)

##
## Call:
## roc.default(response = CreditTest$Class, predictor = ldapred$Bad)
##
## Data: ldapred$Bad in 75 controls (CreditTest$Class Bad) > 175 cases
##        (CreditTest$Class Good).
## Area under the curve: 0.7268
```


Support Vector Machines

- Assume that we have two classes, assessed on a single X variable
- SVMs find the line the best distinguishes the two classes, with margins as large as possible
- SVMs can classify observations, but cannot deliver probabilities
- One tuning parameter is how strict are the boundaries when classes are not linearly separable



SVM Results

```
svm <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),  
            method = 'svmRadial')  
credpred <- predict(logistic, newdata = CreditTest, type = 'response')  
  
|svmpred <- predict(svm, newdata = CreditTest, type = 'raw')  
confusionMatrix(svmpred, CreditTest$Class)  
  
## Confusion Matrix and Statistics  
##  
##           Reference  
## Prediction Bad Good  
##      Bad    28   12  
##      Good   47  163  
##  
##              Accuracy : 0.764  
##              95% CI : (0.7064, 0.8152)  
##      No Information Rate : 0.7  
##      P-Value [Acc > NIR] : 0.01474  
##  
##              Kappa : 0.3516  
##  Mcnemar's Test P-Value : 9.581e-06  
##  
##              Sensitivity : 0.3733  
##              Specificity : 0.9314  
##      Pos Pred Value : 0.7000  
##      Neg Pred Value : 0.7762  
##      Prevalence : 0.3000  
##      Detection Rate : 0.1120  
##      Detection Prevalence : 0.1600  
##      Balanced Accuracy : 0.6524  
##  
##      'Positive' Class : Bad  
##
```

Classification Tree

- Same basic logic as the regression tree, but for classification
- Good trees are classified by “purity”
 - *i.e. Proportions are close to zero or one for a node*
- Variables will be split on the variable that maximizes the purity
 - *Will then choose other variables to split on*
 - *Will continue until maximum depth is reached, sample size in each node reaches a minimum*
- Also very prone to overfitting
 - *Trimming usually helps this*

Decision Tree Results

```
rpart <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),  
              method = 'rpart2', tuneLength = 10)
```

rpart

```
## CART  
##  
## 750 samples  
## 49 predictor  
## 2 classes: 'Bad', 'Good'  
##  
## Pre-processing: centered (49), scaled (49)  
## Resampling: Bootstrapped (25 reps)  
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...  
## Resampling results across tuning parameters:  
##  
##   maxdepth  Accuracy  Kappa  
##    2       0.7139595 0.2212803  
##    3       0.7056917 0.2345376  
##    7       0.7080855 0.2534540  
##    8       0.7070693 0.2543217  
##   12       0.7014758 0.2414044  
##   15       0.7005167 0.2442806  
##   17       0.6975333 0.2379228  
##   21       0.6966232 0.2405164  
##   24       0.6962002 0.2403841  
##   30       0.6962002 0.2403841  
##  
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was maxdepth = 2.
```

```
treepred <- predict(rpart, newdata = CreditTest, type = 'prob')  
roc(response = CreditTest$Class, predictor = treepred$Bad)  
  
##  
## Call:  
## roc.default(response = CreditTest$Class, predictor = treepred$Bad)  
##  
## Data: treepred$Bad in 75 controls (CreditTest$Class Bad) < 175 cases  
##        (CreditTest$Class Good).  
## Area under the curve: 0.3424
```

Random Forest

- Require a simple modification of Random Forest Regression algorithm
- Each decision tree votes for the classification in a new sample
 - *This is how we calculate the probabilities*
- Each constituent tree uses a random subset of variables
- Typically avoids some of the overfitting for individual decision trees

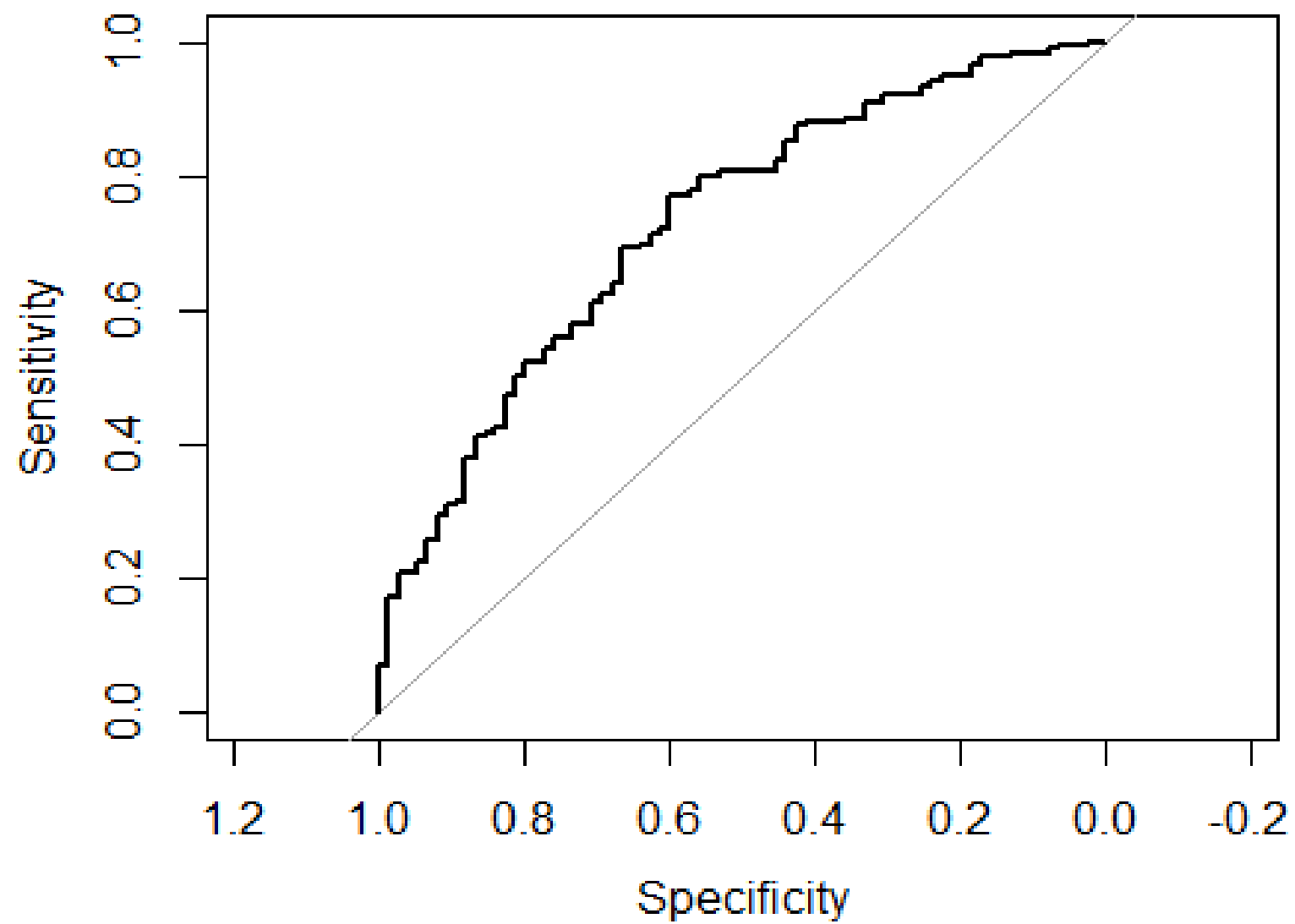
Random Forest Results

```
rf <- train(Class ~ ., data = CreditTrain, preProc = c('center', 'scale'),  
            method = 'rf', tuneLength = 10)  
rfpred <- predict(rf, newdata = CreditTest, type = 'prob')  
rf|  
  
## Random Forest  
##  
## 750 samples  
## 49 predictor  
## 2 classes: 'Bad', 'Good'  
##  
## Pre-processing: centered (49), scaled (49)  
## Resampling: Bootstrapped (25 reps)  
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...  
## Resampling results across tuning parameters:  
##  
##   mtry Accuracy   Kappa  
##    2  0.7253583 0.1678937  
##    7  0.7488560 0.3242074  
##   12  0.7459167 0.3257745  
##   17  0.7421611 0.3247984  
##   22  0.7408988 0.3268828  
##   28  0.7397046 0.3261598  
##   33  0.7370156 0.3208173  
##   38  0.7381571 0.3282542  
##   43  0.7346664 0.3226177  
##   49  0.7342829 0.3226030  
##  
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 7.
```

```
roc(response = CreditTest$Class, predictor = rfpred$Bad)  
  
##  
## Call:  
## roc.default(response = CreditTest$Class, predictor = rfpred$Bad)  
##  
## Data: rfpred$Bad in 75 controls (CreditTest$Class Bad) > 175 cases  
##        (CreditTest$Class Good).  
## Area under the curve: 0.7701
```

Summary of Classification Results

Model	AUC
Logistic Regression	.725
Naïve Bayes	.695
K-Nearest Neighbors	.738
Linear Discriminant Analysis	.727
Support Vector Machine	N/A
Decision tree	.342
Random Forest	.770



Most Important Predictors

```
varImp(rf)
```

```
rf variable importance
```

```
only 20 most important variables shown (out of 47)
```

	overall
Amount	100.000
Age	71.508
Duration	67.658
CheckingAccountStatus.none	48.718
CheckingAccountStatus.lt.0	29.163
ResidenceDuration	25.552
InstallmentRatePercentage	25.517
SavingsAccountBonds.lt.100	16.810
CreditHistory.Critical	13.468
CheckingAccountStatus.0.to.200	13.055
Purpose.NewCar	11.824
NumberExistingCredits	11.657
OtherInstallmentPlans.None	9.346
Job.skilledEmployee	8.742
Personal.Male.Single	8.485
Property.RealEstate	8.344
Telephone	8.111
Property.Insurance	7.658
EmploymentDuration.1.to.4	7.223
SavingsAccountBonds.Unknown	7.174

Deep Learning

- Characterized by a “black box”
 - *Subset of machine learning*
 - *Able to train itself*
- Appropriate for more complicated tasks, like image or sound recognition
- Algorithms are typically less organized, more flexible, and way more complicated
- The most popular methods are variations on artificial neural networks
 - *Inspired by the biology of human neural networks*



Change Alt Text



The automatically generated alternative text for this photo is:

2 people, people standing

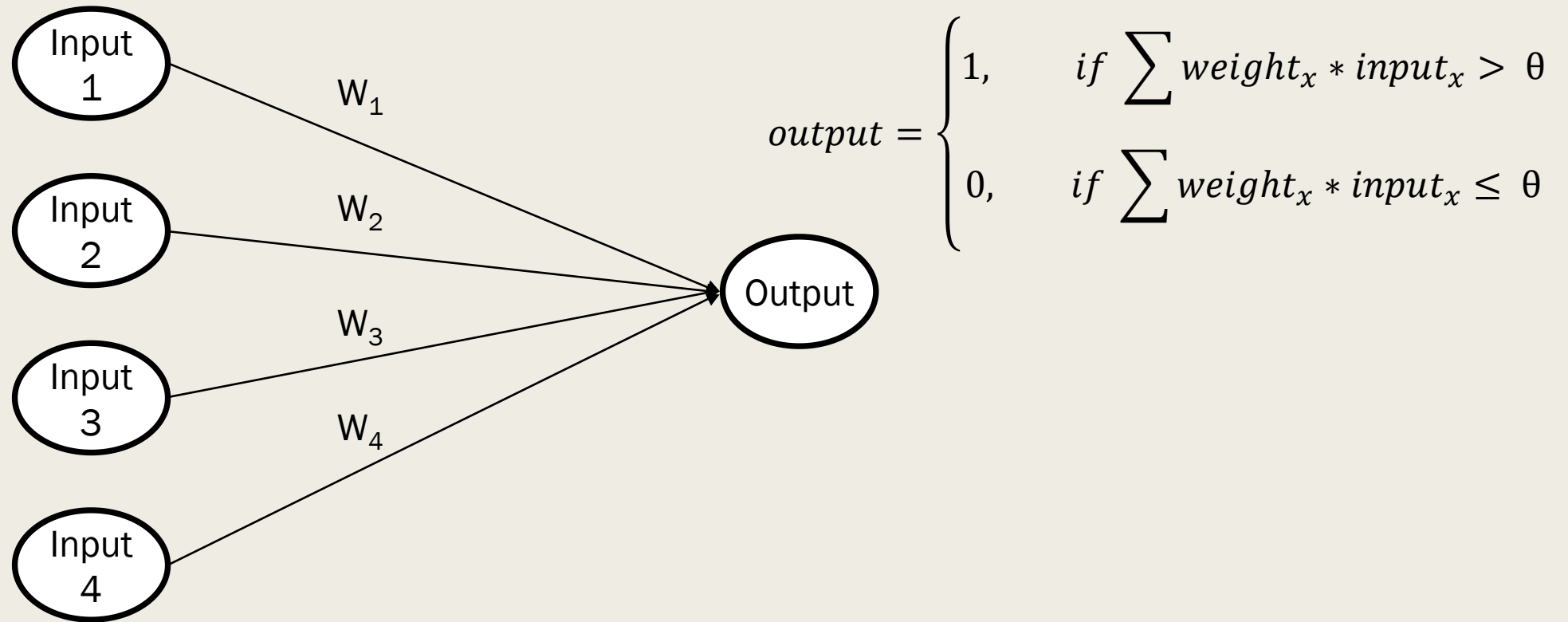
Add alternative text that describes the contents of the photo for people with visual impairments.

Override generated alt text

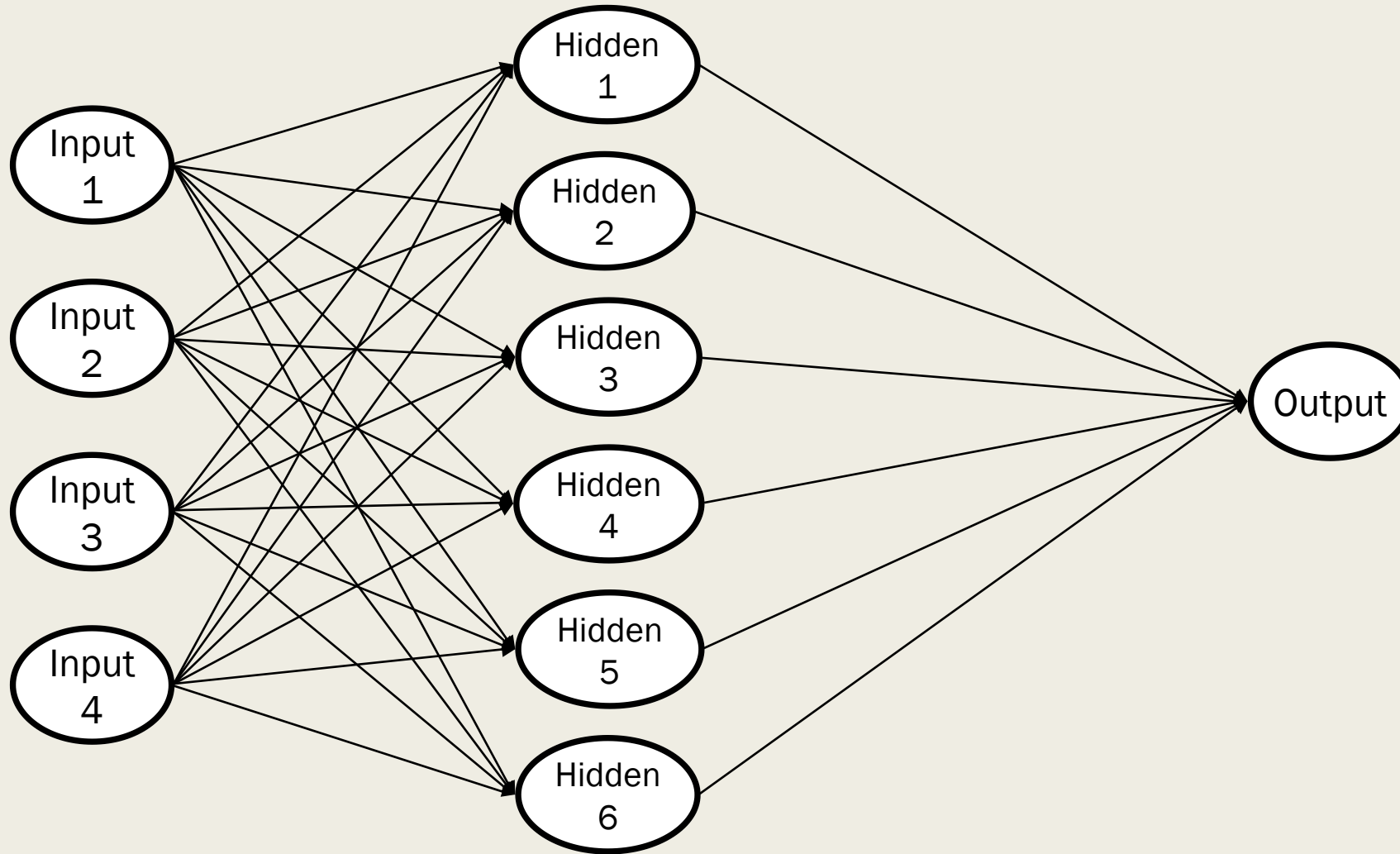
Cancel

Save

Single Layer Perceptron



Multilayer Perceptron



What is the point of the hidden layer(s)?

- Raw input is usually not able to be weighted in a way that can be used for the output
- Transform the input into something the output layer can use
- Output layer will then scale the hidden inputs into the example classes used
- This allows for multiple functions to be applied to input
 - *For instance, allows the network to pick up on different features of the input*

Convolutional Neural Networks

- Type of multilayer perceptron that utilizes convolutional layers
 - *Particularly useful for image recognition*
- Convolutional layers narrow the input down to a subset of input (like pixels in a picture), trained through optimization
- This subset is “pooled” together, and then multiple other pooled layers are combined
- Multilayer perceptron (or further convolution) is then applied to these pooled layers

MNIST Dataset

- The 'Hello World!' of machine learning datasets
- Images are handwritten digits that take on values of 0-9
- Each images is 28x28 pixels
 - *Each image is 784 dimensional, with a vector representing the value*
- Numeric values are represented by pixel darkness
- Consists of 60,000 training images and 10,000 test images

MNI

- The ‘
- Imag
- Each
-
- Num
- Cons

V92	V93	V94	V95	V96	V97	V98	V99	V100
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.6313726	0.9137256	0.62352943	0.9058824
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.36078432	0.8470589	0.9921569	0.60000002	0.2470588
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.000000000	0.000000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
0.003921569	0.09803922	0.5098040	0.5098040	0.50980395	0.8705883	1.0000000	1.00000000	1.0000000

What is Tensorflow?

- Developed by the Google Brain team for internal use
 - *Including the “Godfather of AI” Geoffrey Hinton*
- Replaced their DistBelief that was used for training neural networks
- Was issued as an open sourced library in 2015
- Python API is the most widely used, but interfaces with R (through Python)

Training a Simple Neural Network

- We have to initialize a TensorFlow 'session'
- Load in the data
- Create placeholders for the data
- Create batches that help the normalization of the data
- Train the network using Gradient Descent
- Assess accuracy

```
library(tensorflow)
install_tensorflow()
```

```
## Using r-tensorflow conda environment for TensorFlow installation
##
## Installation complete.
```

```
sess = tf$Session()
hello <- tf$constant('Hello, TensorFlow!')
sess$run(hello)
```

```
## b'Hello, TensorFlow!'
```

```
tfflow <- import("tensorflow")
```

```
datasets <- tf$contrib$learn$datasets
mnist <- datasets$mnist$read_data_sets("MNIST-data", one_hot = TRUE)
mnist
```

```
## Datasets(train=<tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet>, validation=<tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet>, test=<tensorflow.contrib.learn.python.learn.datasets.mnist.DataSet>)
```

```
mnist$train$images[1,]
```

```
## [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [7] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [13] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [19] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [25] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [31] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
```

```
## [655] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [661] 0.94901967 0.99607849 0.93725497 0.22352943 0.00000000 0.00000000
## [667] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [673] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [679] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [685] 0.00000000 0.00000000 0.00000000 0.34901962 0.98431379 0.94509810
## [691] 0.33725491 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [697] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [703] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [709] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [715] 0.01960784 0.80784321 0.96470594 0.61568630 0.00000000 0.00000000
## [721] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [727] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [733] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [739] 0.00000000 0.00000000 0.00000000 0.00000000 0.01568628 0.45882356
## [745] 0.27058825 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [751] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [757] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [763] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [769] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [775] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [781] 0.00000000 0.00000000 0.00000000 0.00000000
```

```
mnist$train$labels[1,]
```

```
## [1] 0 0 0 0 0 0 0 1 0 0
```

```
x <- tf$placeholder(tf$float32, shape(NULL, 784L))
W <- tf$Variable(tf$zeros(shape(784L, 10L)))
b <- tf$Variable(tf$zeros(shape(10L)))
y <- tf$nn$softmax(tf$matmul(x, W) + b)
y_ <- tf$placeholder(tf$float32, shape(NULL, 10L))
```

```
cross_entropy <- tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y), reduction_indices=1L))
optimizer <- tf$train$GradientDescentOptimizer(0.5)
train_step <- optimizer$minimize(cross_entropy)
init <- tf$global_variables_initializer()
sess <- tf$Session()
sess$run(init)
for (i in 1:1000) {
  batches <- mnist$train$next_batch(100L)
  batch_xs <- batches[[1]]
  batch_ys <- batches[[2]]
  sess$run(train_step,
            feed_dict = dict(x = batch_xs, y_ = batch_ys))
}
```

```
correct_prediction <- tf$equal(tf$argmax(y, 1L), tf$argmax(y_, 1L))
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
sess$run(accuracy, feed_dict=dict(x = mnist$test$images, y_ = mnist$test$labels))
```

```
## [1] 0.9176
```

Training a Convolutional Neural Network

- Same steps as before except...
- Set up convolutional and pooling layers
 - *Our example will include two layers of 32 and 64*
 - *Pooling layers are 2x2 dimensions*


```
x <- tf$placeholder(tf$float32, shape(NULL, 784L))
y_ <- tf$placeholder(tf$float32, shape(NULL, 10L))
W <- tf$Variable(tf$zeros(shape(784L, 10L)))
b <- tf$Variable(tf$zeros(shape(10L)))
sess$run(tf$global_variables_initializer())
y <- tf$nn$softmax(tf$matmul(x,W) + b)
y
```

```
## Tensor("Softmax:0", shape=(?, 10), dtype=float32)
```

```
cross_entropy <- tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y), reduction_indices=1L))
optimizer <- tf$train$GradientDescentOptimizer(0.5)
train_step <- optimizer$minimize(cross_entropy)
for (i in 1:1000) {
  batches <- mnist$train$next_batch(100L)
  batch_xs <- batches[[1]]
  batch_ys <- batches[[2]]
  sess$run(train_step,
            feed_dict = dict(x = batch_xs, y_ = batch_ys))
}
```

```
correct_prediction <- tf$equal(tf$argmax(y, 1L), tf$argmax(y_, 1L))
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
accuracy$eval(feed_dict=dict(x = mnist$test$images, y_ = mnist$test$labels))
```

```
weight_variable <- function(shape) {  
  initial <- tf$truncated_normal(shape, stddev=0.1)  
  tf$Variable(initial)  
}  
  
bias_variable <- function(shape) {  
  initial <- tf$constant(0.1, shape=shape)  
  tf$Variable(initial)  
}  
  
conv2d <- function(x, W) {  
  tf$nn$conv2d(x, W, strides=c(1L, 1L, 1L, 1L), padding='SAME')  
}  
  
max_pool_2x2 <- function(x) {  
  tf$nn$max_pool(  
    x,  
    ksize=c(1L, 2L, 2L, 1L),  
    strides=c(1L, 2L, 2L, 1L),  
    padding='SAME')  
}
```

```
W_conv1 <- weight_variable(shape(5L, 5L, 1L, 32L))
b_conv1 <- bias_variable(shape(32L))
x_image <- tf$reshape(x, shape(-1L, 28L, 28L, 1L))
h_conv1 <- tf$nn$relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 <- max_pool_2x2(h_conv1)
W_conv2 <- weight_variable(shape = shape(5L, 5L, 32L, 64L))
b_conv2 <- bias_variable(shape = shape(64L))

h_conv2 <- tf$nn$relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 <- max_pool_2x2(h_conv2)

W_fcl <- weight_variable(shape(7L * 7L * 64L, 1024L))
b_fcl <- bias_variable(shape(1024L))

h_pool2_flat <- tf$reshape(h_pool2, shape(-1L, 7L * 7L * 64L))
h_fcl <- tf$nn$relu(tf$matmul(h_pool2_flat, W_fcl) + b_fcl)

keep_prob <- tf$placeholder(tf$float32)
h_fcl_drop <- tf$nn$dropout(h_fcl, keep_prob)

W_fc2 <- weight_variable(shape(1024L, 10L))
b_fc2 <- bias_variable(shape(10L))

y_conv <- tf$nn$softmax(tf$matmul(h_fcl_drop, W_fc2) + b_fc2)
```

```
cross_entropy <- tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y_conv), reduction_indices=1L))
train_step <- tf$train$AdamOptimizer(1e-4)$minimize(cross_entropy)
correct_prediction <- tf$equal(tf$argmax(y_conv, 1L), tf$argmax(y_, 1L))
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
sess$run(tf$global_variables_initializer())

for (i in 1:20000) {
  batch <- mnist$train$next_batch(50L)
  if (i %% 100 == 0) {
    train_accuracy <- accuracy$eval(feed_dict = dict(
      x = batch[[1]], y_ = batch[[2]], keep_prob = 1.0))
    cat(sprintf("step %d, training accuracy %g\n", i, train_accuracy))
  }
  train_step$run(feed_dict = dict(
    x = batch[[1]], y_ = batch[[2]], keep_prob = 0.5))
}
```

```
## step 100, training accuracy 0.76
## step 200, training accuracy 0.94
## step 300, training accuracy 0.96
## step 400, training accuracy 0.92
## step 500, training accuracy 0.94
## step 600, training accuracy 0.96
## step 700, training accuracy 0.94
## step 800, training accuracy 0.94
## step 900, training accuracy 0.94
```



TWO AND A HALF HOURS LATER...

(During which I went to the grocery store, did laundry,
and made dinner)

```
test_accuracy <- accuracy$eval(feed_dict = dict(  
  x = mnist$test$images, y_ = mnist$test$labels, keep_prob = 1.0))  
cat(sprintf("test accuracy %g", test_accuracy))
```

```
## test accuracy 0.9918
```

Other Neural Networks

- Recurrent neural networks
 - *Used for time series data*
- Deep belief networks
 - *Unsupervised neural nets*
- Long short-term memory
 - *Maintains “memory” of values*
- Reinforcement Learning
 - *Allows the machine to learn over time*

Caveats about Neural Networks

- The computational time is crazy
- Have failed to meet high expectations
- Pretty much no interpretability