Machine Bias

propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

There’s software used across the country to predict future criminals. And it’s biased against blacks.

by Julia Angwin, Jeff Larson, Surya Mattu and Lauren Kirchner, ProPublica May 23, 2016

On a spring afternoon in 2014, Brisha Borden was running late to pick up her god-sister from school when she spotted an unlocked kid’s blue Huffy bicycle and a silver Razor scooter. Borden and a friend grabbed the bike and scooter and tried to ride them down the street in the Fort Lauderdale suburb of Coral Springs.

Just as the 18-year-old girls were realizing they were too big for the tiny conveyances — which belonged to a 6-year-old boy — a woman came running after them saying, “That’s my kid’s stuff.” Borden and her friend immediately dropped the bike and scooter and walked away.

But it was too late — a neighbor who witnessed the heist had already called the police. Borden and her friend were arrested and charged with burglary and petty theft for the items, which were valued at a total of $80.

Compare their crime with a similar one: The previous summer, 41-year-old Vernon Prater was picked up for shoplifting $86.35 worth of tools from a nearby Home Depot store.

Prater was the more seasoned criminal. He had already been convicted of armed robbery and attempted armed robbery, for which he served five years in prison, in addition to another armed robbery charge. Borden had a record, too, but it was for misdemeanors committed when she was a juvenile.

Yet something odd happened when Borden and Prater were booked into jail: A computer program spat out a score predicting the likelihood of each committing a future crime. Borden — who is black — was rated a high risk. Prater — who is white — was rated a low risk.

Two years later, we know the computer algorithm got it exactly backward. Borden has not been charged with any new crimes. Prater is serving an eight-year prison term for subsequently breaking into a warehouse and stealing thousands of dollars’ worth of electronics.

Scores like this — known as risk assessments — are increasingly common in courtrooms across the nation. They are used to inform decisions about who can be set free at every stage of the criminal justice system, from assigning bond amounts — as is the case in Fort

Lauderdale — to even more fundamental decisions about defendants' freedom. In Arizona, Colorado, Delaware, Kentucky, Louisiana, Oklahoma, Virginia, Washington and Wisconsin, the results of such assessments are given to judges during criminal sentencing.

Rating a defendant's risk of future crime is often done in conjunction with an evaluation of a defendant's rehabilitation needs. The Justice Department's National Institute of Corrections now encourages the use of such combined assessments at every stage of the criminal justice process. And a landmark sentencing reform bill currently pending in Congress would mandate the use of such assessments in federal prisons.

Two Petty Theft Arrests

Prior Offenses	2 armed robberies, 1 attempted armed robbery
Low Risk	3
Subsequent Offenses	1 grand theft

Vernon Prater

Prior Offenses	4 juvenile misdemeanors
High Risk	8
Subsequent Offenses	None

Brisha Borden

Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

In 2014, then U.S. Attorney General Eric Holder warned that the risk scores might be injecting bias into the courts. He called for the U.S. Sentencing Commission to study their use. “Although these measures were crafted with the best of intentions, I am concerned that they inadvertently undermine our efforts to ensure individualized and equal justice,” he said, adding, “they may exacerbate unwarranted and unjust disparities that are already far too common in our criminal justice system and in our society.”

The sentencing commission did not, however, launch a study of risk scores. So ProPublica did, as part of a larger examination of the powerful, largely hidden effect of algorithms in American life.

We obtained the risk scores assigned to more than 7,000 people arrested in Broward County, Florida, in 2013 and 2014 and checked to see how many were charged with new crimes over the next two years, the same benchmark used by the creators of the algorithm.

The score proved remarkably unreliable in forecasting violent crime: Only 20 percent of the people predicted to commit violent crimes actually went on to do so.

When a full range of crimes were taken into account — including misdemeanors such as driving with an expired license — the algorithm was somewhat more accurate than a coin flip. Of those deemed likely to re-offend, 61 percent were arrested for any subsequent crimes within two years.

We also turned up significant racial disparities, just as Holder feared. In forecasting who would re-offend, the algorithm made mistakes with black and white defendants at roughly the same rate but in very different ways.

- The formula was particularly likely to falsely flag black defendants as future criminals, wrongly labeling them this way at almost twice the rate as white defendants.
- White defendants were mislabeled as low risk more often than black defendants.

Could this disparity be explained by defendants' prior crimes or the type of crimes they were arrested for? No. We ran a statistical test that isolated the effect of race from criminal history and recidivism, as well as from defendants' age and gender. Black defendants were still 77 percent more likely to be pegged as at higher risk of committing a future violent crime and 45 percent more likely to be predicted to commit a future crime of any kind. (Read our analysis.)

The algorithm used to create the Florida risk scores is a product of a for-profit company, Northpointe. The company disputes our analysis.

In a letter, it criticized ProPublica's methodology and defended the accuracy of its test: "Northpointe does not agree that the results of your analysis, or the claims being made based upon that analysis, are correct or that they accurately reflect the outcomes from the application of the model."

Northpointe's software is among the most widely used assessment tools in the country. The company does not publicly disclose the calculations used to arrive at defendants' risk scores, so it is not possible for either defendants or the public to see what might be driving the disparity. (On Sunday, Northpointe gave ProPublica the basics of its future-crime formula — which includes factors such as education levels, and whether a defendant has a job. It did not share the specific calculations, which it said are proprietary.)

Northpointe's core product is a set of scores derived from 137 questions that are either answered by defendants or pulled from criminal records. Race is not one of the questions. The survey asks defendants such things as: "Was one of your parents ever sent to jail or prison?" "How many of your friends/acquaintances are taking drugs illegally?" and "How often did you get in fights while at school?" The questionnaire also asks people to agree or disagree with statements such as "A hungry person has a right to steal" and "If people make me angry or lose my temper, I can be dangerous."

The appeal of risk scores is obvious: The United States locks up far more people than any other country, a disproportionate number of them black. For more than two centuries, the key decisions in the legal process, from pretrial release to sentencing to parole, have been in the hands of human beings guided by their instincts and personal biases.

If computers could accurately predict which defendants were likely to commit new crimes, the criminal justice system could be fairer and more selective about who is incarcerated and for how long. The trick, of course, is to make sure the computer gets it right. If it's wrong in one direction, a dangerous criminal could go free. If it's wrong in another direction, it could result in someone unfairly receiving a harsher sentence or waiting longer for parole than is appropriate.

The first time Paul Zilly heard of his score — and realized how much was riding on it — was during his sentencing hearing on Feb. 15, 2013, in court in Barron County, Wisconsin. Zilly had been convicted of stealing a push lawnmower and some tools. The prosecutor recommended a year in county jail and follow-up supervision that could help Zilly with "staying on the right path." His lawyer agreed to a plea deal.

But Judge James Babler had seen Zilly's scores. Northpointe's software had rated Zilly as a high risk for future violent crime and a medium risk for general recidivism. "When I look at the risk assessment," Babler said in court, "it is about as bad as it could be."

Then Babler overturned the plea deal that had been agreed on by the prosecution and defense and imposed two years in state prison and three years of supervision.

Criminologists have long tried to predict which criminals are more dangerous before deciding whether they should be released. Race, nationality and skin color were often used in making such predictions until about the 1970s, when it became politically unacceptable, according to a survey of risk assessment tools by Columbia University law professor Bernard Harcourt.

In the 1980s, as a crime wave engulfed the nation, lawmakers made it much harder for judges and parole boards to exercise discretion in making such decisions. States and the federal government began instituting mandatory sentences and, in some cases, abolished parole, making it less important to evaluate individual offenders.

But as states struggle to pay for swelling prison and jail populations, forecasting criminal risk has made a comeback.

Two Drug Possession Arrests

Prior Offense	1 attempted burglary
Low Risk	3
Subsequent Offenses	3 drug possessions

Dylan Fugett

Prior Offense	1 resisting arrest without violence
High Risk	10
Subsequent Offenses	None

Bernard Parker

Fugett was rated low risk after being arrested with cocaine and marijuana. He was arrested three times on drug charges after that.

Dozens of risk assessments are being used across the nation — some created by for-profit companies such as Northpointe and others by nonprofit organizations. (One tool being used in states including Kentucky and Arizona, called the Public Safety Assessment, was developed by the Laura and John Arnold Foundation, which also is a funder of ProPublica.)

There have been few independent studies of these criminal risk assessments. In 2013, researchers Sarah Desmarais and Jay Singh examined 19 different risk methodologies used in the United States and found that “in most cases, validity had only been examined in one or two studies” and that “frequently, those investigations were completed by the same people who developed the instrument.”

Their analysis of the research through 2012 found that the tools “were moderate at best in terms of predictive validity,” Desmarais said in an interview. And she could not find any substantial set of studies conducted in the United States that examined whether risk scores were racially biased. “The data do not exist,” she said.

Since then, there have been some attempts to explore racial disparities in risk scores. One 2016 study examined the validity of a risk assessment tool, not Northpointe’s, used to make probation decisions for about 35,000 federal convicts. The researchers, Jennifer Skeem at University of California, Berkeley, and Christopher T. Lowenkamp from the Administrative Office of the U.S. Courts, found that blacks did get a higher average score but concluded the differences were not attributable to bias.

The increasing use of risk scores is controversial and has garnered media coverage, including articles by the Associated Press, and the Marshall Project and FiveThirtyEight last year.

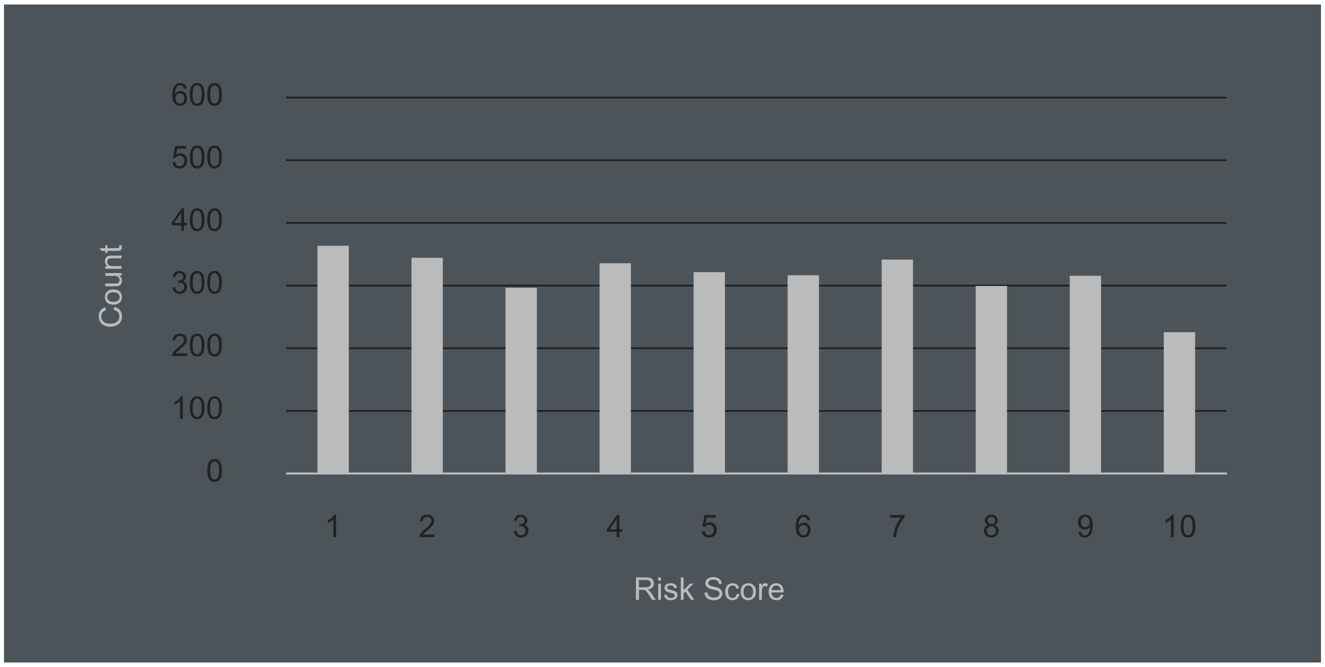
Most modern risk tools were originally designed to provide judges with insight into the types of treatment that an individual might need — from drug treatment to mental health counseling.

“What it tells the judge is that if I put you on probation, I’m going to need to give you a lot of services or you’re probably going to fail,” said Edward Latessa, a University of Cincinnati professor who is the author of a risk assessment tool that is used in Ohio and several other states.

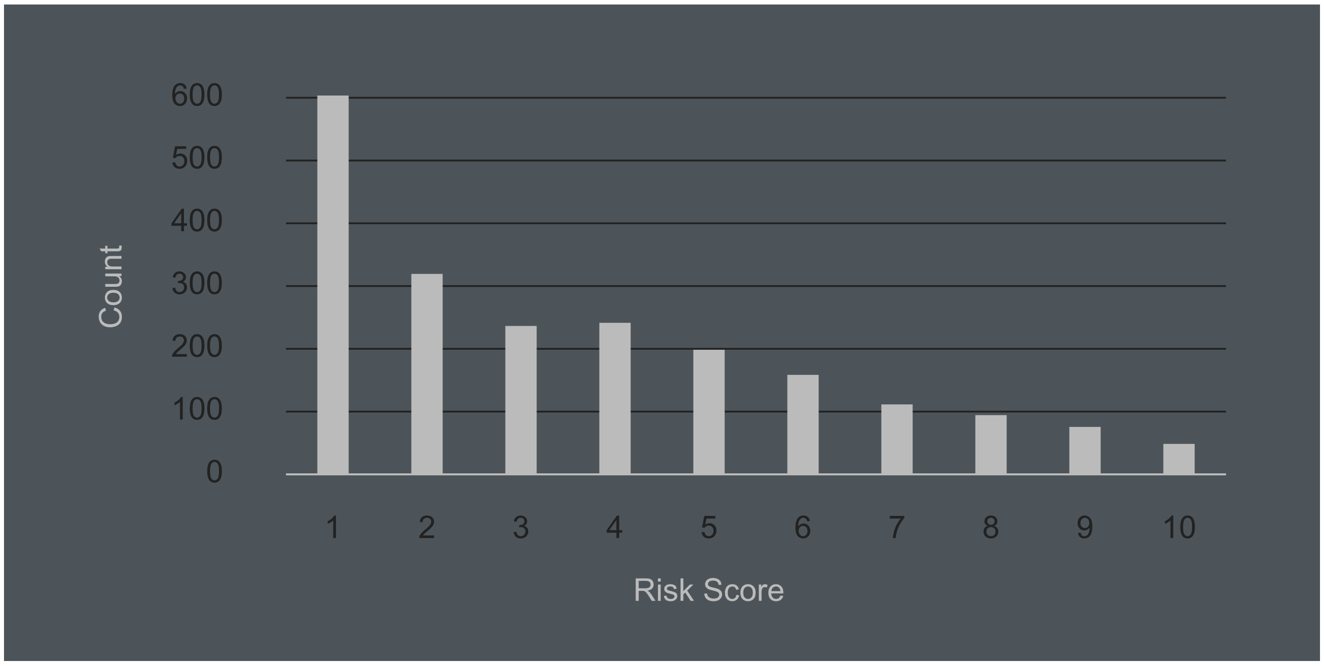
But being judged ineligible for alternative treatment — particularly during a sentencing hearing — can translate into incarceration. Defendants rarely have an opportunity to challenge their assessments. The results are usually shared with the defendant’s attorney, but the calculations that transformed the underlying data into a score are rarely revealed.

“Risk assessments should be impermissible unless both parties get to see all the data that go into them,” said Christopher Slobogin, director of the criminal justice program at Vanderbilt Law School. “It should be an open, full-court adversarial proceeding.”

Black Defendants’ Risk Scores



White Defendants’ Risk Scores



These charts show that scores for white defendants were skewed toward lower-risk categories. Scores for black defendants were not. (Source: ProPublica analysis of data from Broward County, Fla.)

Proponents of risk scores argue they can be used to reduce the rate of incarceration. In 2002, Virginia became one of the first states to begin using a risk assessment tool in the sentencing of nonviolent felony offenders statewide. In 2014, Virginia judges using the tool sent nearly half of those defendants to alternatives to prison, according to a state sentencing commission report. Since 2005, the state's prison population growth has slowed to 5 percent from a rate of 31 percent the previous decade.

In some jurisdictions, such as Napa County, California, the probation department uses risk assessments to suggest to the judge an appropriate probation or treatment plan for individuals being sentenced. Napa County Superior Court Judge Mark Boessenecker said he finds the recommendations helpful. "We have a dearth of good treatment programs, so filling a slot in a program with someone who doesn't need it is foolish," he said.

However, Boessenecker, who trains other judges around the state in evidence-based sentencing, cautions his colleagues that the score doesn't necessarily reveal whether a person is dangerous or if they should go to prison.

"A guy who has molested a small child every day for a year could still come out as a low risk because he probably has a job," Boessenecker said. "Meanwhile, a drunk guy will look high risk because he's homeless. These risk factors don't tell you whether the guy ought to go to prison or not; the risk factors tell you more about what the probation conditions ought to be."

"I'm surprised [my risk score] is so low. I spent five years in state prison in Massachusetts." (Josh Ritchie for ProPublica)

Sometimes, the scores make little sense even to defendants.

James Rivelli, a 54-year old Hollywood, Florida, man, was arrested two years ago for shoplifting seven boxes of Crest Whitestrips from a CVS drugstore. Despite a criminal record that included aggravated assault, multiple thefts and felony drug trafficking, the Northpointe algorithm classified him as being at a low risk of reoffending.

"I am surprised it is so low," Rivelli said when told by a reporter he had been rated a 3 out of a possible 10. "I spent five years in state prison in Massachusetts. But I guess they don't count that here in Broward County." In fact, criminal records from across the nation are supposed to be included in risk assessments.

Less than a year later, he was charged with two felony counts for shoplifting about $1,000 worth of tools from Home Depot. He said his crimes were fueled by drug addiction and that he is now sober.

Northpointe was founded in 1989 by Tim Brennan, then a professor of statistics at the University of Colorado, and Dave Wells, who was running a corrections program in Traverse City, Michigan.

Wells had built a prisoner classification system for his jail. “It was a beautiful piece of work,” Brennan said in an interview conducted before ProPublica had completed its analysis. Brennan and Wells shared a love for what Brennan called “quantitative taxonomy” — the measurement of personality traits such as intelligence, extroversion and introversion. The two decided to build a risk assessment score for the corrections industry.

Brennan wanted to improve on a leading risk assessment score, the LSI, or Level of Service Inventory, which had been developed in Canada. “I found a fair amount of weakness in the LSI,” Brennan said. He wanted a tool that addressed the major theories about the causes of crime.

Brennan and Wells named their product the Correctional Offender Management Profiling for Alternative Sanctions, or COMPAS. It assesses not just risk but also nearly two dozen so-called “criminogenic needs” that relate to the major theories of criminality, including “criminal personality,” “social isolation,” “substance abuse” and “residence/stability.” Defendants are ranked low, medium or high risk in each category.

Two DUI Arrests

Prior Offenses	3 DUIs, 1 battery
Low Risk	1
Subsequent Offenses	1 domestic violence battery

Gregory Lugo

Prior Offenses	2 misdemeanors
Medium Risk	6
Subsequent Offenses	None

Mallory Williams

Lugo crashed his Lincoln Navigator into a Toyota Camry while drunk. He was rated as a low risk of reoffending despite the fact that it was at least his fourth DUI.

As often happens with risk assessment tools, many jurisdictions have adopted Northpointe’s software before rigorously testing whether it works. New York State, for instance, started using the tool to assess people on probation in a pilot project in 2001 and rolled it out to the rest of the state’s probation departments — except New York City — by 2010. The state didn’t publish a comprehensive statistical evaluation of the tool until 2012. The study of more than 16,000 probationers found the tool was 71 percent accurate, but it did not evaluate racial differences.

A spokeswoman for the New York state division of criminal justice services said the study did not examine race because it only sought to test whether the tool had been properly calibrated to fit New York’s probation population. She also said judges in nearly all New York counties are given defendants’ Northpointe assessments during sentencing.

In 2009, Brennan and two colleagues published a validation study that found that Northpointe’s risk of recidivism score had an accuracy rate of 68 percent in a sample of 2,328 people. Their study also found that the score was slightly less predictive for black men than white men — 67 percent versus 69 percent. It did not examine racial disparities beyond that, including whether some groups were more likely to be wrongly labeled higher risk.

Brennan said it is difficult to construct a score that doesn’t include items that can be correlated with race — such as poverty, joblessness and social marginalization. “If those are omitted from your risk assessment, accuracy goes down,” he said.

In 2011, Brennan and Wells sold Northpointe to Toronto-based conglomerate Constellation Software for an undisclosed sum.

Wisconsin has been among the most eager and expansive users of Northpointe’s risk assessment tool in sentencing decisions. In 2012, the Wisconsin Department of Corrections launched the use of the software throughout the state. It is used at each step in the prison system, from sentencing to parole.

In a 2012 presentation, corrections official Jared Hoy described the system as a “giant correctional pinball machine” in which correctional officers could use the scores at every “decision point.”

Wisconsin has not yet completed a statistical validation study of the tool and has not said when one might be released. State corrections officials declined repeated requests to comment for this article.

Some Wisconsin counties use other risk assessment tools at arrest to determine if a defendant is too risky for pretrial release. Once a defendant is convicted of a felony anywhere in the state, the Department of Corrections attaches Northpointe’s assessment to the confidential presentence report given to judges, according to Hoy’s presentation.

In theory, judges are not supposed to give longer sentences to defendants with higher risk scores. Rather, they are supposed to use the tests primarily to determine which defendants are eligible for probation or treatment programs.

Prediction Fails Differently for Black Defendants

	White	African American
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

Overall, Northpointe's assessment tool correctly predicts recidivism 61 percent of the time. But blacks are almost twice as likely as whites to be labeled a higher risk but not actually re-offend. It makes the opposite mistake among whites: They are much more likely than blacks to be labeled lower risk but go on to commit other crimes. (Source: ProPublica analysis of data from Broward County, Fla.)

But judges have cited scores in their sentencing decisions. In August 2013, Judge Scott Horne in La Crosse County, Wisconsin, declared that defendant Eric Loomis had been "identified, through the COMPAS assessment, as an individual who is at high risk to the community." The judge then imposed a sentence of eight years and six months in prison.

Loomis, who was charged with driving a stolen vehicle and fleeing from police, is challenging the use of the score at sentencing as a violation of his due process rights. The state has defended Horne's use of the score with the argument that judges can consider the score in addition to other factors. It has also stopped including scores in presentencing reports until the state Supreme Court decides the case.

"The risk score alone should not determine the sentence of an offender," Wisconsin Assistant Attorney General Christine Remington said last month during state Supreme Court arguments in the Loomis case. "We don't want courts to say, this person in front of me is a 10 on COMPAS as far as risk, and therefore I'm going to give him the maximum sentence."

That is almost exactly what happened to Zilly, the 48-year-old construction worker sent to prison for stealing a push lawnmower and some tools he intended to sell for parts. Zilly has long struggled with a meth habit. In 2012, he had been working toward recovery with the help of a Christian pastor when he relapsed and committed the thefts.

After Zilly was scored as a high risk for violent recidivism and sent to prison, a public defender appealed the sentence and called the score's creator, Brennan, as a witness.

Brennan testified that he didn't design his software to be used in sentencing. "I wanted to stay away from the courts," Brennan said, explaining that his focus was on reducing crime rather than punishment. "But as time went on I started realizing that so many decisions are made, you know, in the courts. So I gradually softened on whether this could be used in the courts or not."

"Not that I'm innocent, but I just believe people do change." (Stephen Maturen for ProPublica)

Still, Brennan testified, “I don’t like the idea myself of COMPAS being the sole evidence that a decision would be based upon.”

After Brennan’s testimony, Judge Babler reduced Zilly’s sentence, from two years in prison to 18 months. “Had I not had the COMPAS, I believe it would likely be that I would have given one year, six months,” the judge said at an appeals hearing on Nov. 14, 2013.

Zilly said the score didn’t take into account all the changes he was making in his life — his conversion to Christianity, his struggle to quit using drugs and his efforts to be more available for his son. “Not that I’m innocent, but I just believe people do change.”

Florida’s Broward County, where Brisha Borden stole the Huffy bike and was scored as high risk, does not use risk assessments in sentencing. “We don’t think the [risk assessment] factors have any bearing on a sentence,” said David Scharf, executive director of community programs for the Broward County Sheriff’s Office in Fort Lauderdale.

Broward County has, however, adopted the score in pretrial hearings, in the hope of addressing jail overcrowding. A court-appointed monitor has overseen Broward County’s jails since 1994 as a result of the settlement of a lawsuit brought by inmates in the 1970s. Even now, years later, the Broward County jail system is often more than 85 percent full, Scharf said.

In 2008, the sheriff’s office decided that instead of building another jail, it would begin using Northpointe’s risk scores to help identify which defendants were low risk enough to be released on bail pending trial. Since then, nearly everyone arrested in Broward has been scored soon after being booked. (People charged with murder and other capital crimes are not scored because they are not eligible for pretrial release.)

The scores are provided to the judges who decide which defendants can be released from jail. “My feeling is that if they don’t need them to be in jail, let’s get them out of there,” Scharf said.

Two Shoplifting Arrests

Prior Offenses	1 domestic violence aggravated assault, 1 grand theft, 1 petty theft, 1 drug trafficking
Low Risk	3
Subsequent Offenses	1 grand theft

James Rivelli

Prior Offense	1 petty theft
Medium Risk	6
Subsequent Offenses	None

Robert Cannon

After Rivelli stole from a CVS and was caught with heroin in his car, he was rated a low risk. He later shoplifted $1,000 worth of tools from a Home Depot.

Scharf said the county chose Northpointe's software over other tools because it was easy to use and produced "simple yet effective charts and graphs for judicial review." He said the system costs about $22,000 a year.

In 2010, researchers at Florida State University examined the use of Northpointe's system in Broward County over a 12-month period and concluded that its predictive accuracy was "equivalent" in assessing defendants of different races. Like others, they did not examine whether different races were classified differently as low or high risk.

Scharf said the county would review ProPublica's findings. "We'll really look at them up close," he said.

Broward County Judge John Hurley, who oversees most of the pretrial release hearings, said the scores were helpful when he was a new judge, but now that he has experience he prefers to rely on his own judgment. "I haven't relied on COMPAS in a couple years," he said.

Hurley said he relies on factors including a person's prior criminal record, the type of crime committed, ties to the community, and their history of failing to appear at court proceedings.

ProPublica's analysis reveals that higher Northpointe scores are slightly correlated with longer pretrial incarceration in Broward County. But there are many reasons that could be true other than judges being swayed by the scores — people with higher risk scores may also be poorer and have difficulty paying bond, for example.

Most crimes are presented to the judge with a recommended bond amount, but he or she can adjust the amount. Hurley said he often releases first-time or low-level offenders without any bond at all.

However, in the case of Borden and her friend Sade Jones, the teenage girls who stole a kid’s bike and scooter, Hurley raised the bond amount for each girl from the recommended $0 to $1,000 each.

Hurley said he has no recollection of the case and cannot recall if the scores influenced his decision.

Sade Jones, who had never been arrested before, was rated a medium risk. (Josh Ritchie for ProPublica)

The girls spent two nights in jail before being released on bond.

“We literally sat there and cried” the whole time they were in jail, Jones recalled. The girls were kept in the same cell. Otherwise, Jones said, “I would have gone crazy.” Borden declined repeated requests to comment for this article.

Jones, who had never been arrested before, was rated a medium risk. She completed probation and got the felony burglary charge reduced to misdemeanor trespassing, but she has still struggled to find work.

“I went to McDonald’s and a dollar store, and they all said no because of my background,” she said. “It’s all kind of difficult and unnecessary.”

The 10-Page Machine Learning Book

(The title here alludes to Andriy Burkov's excellent work, *The Hundred-Page Machine Learning Book*. Note too my own forthcoming book, *The Art of Machine Learning: Algorithms+Data+R*.)

Here we give an overview of the most widely used predictive methods in statistical/machine learning (ML). For each one, we present

- background
- overview of how it works
- function in the R package, **qeML** (readers without R background or who simply wish to acquire an overview of ML may skip the R code without loss of comprehehnsion of the text)

Contents

Notation

For convenience, we'll let Y denote the variable to be predicted, i.e. the response variable, and let X denote the set of predictor variables/features. (ML people tend to use the term *features*, while In stat, the term *predictors* is common. In applied fields, e.g. economics or psychology, some use the term *independent variables*.)

We develop our prediction rule from available *training data*, consisting of n data points, denoted by (X_1, Y_1),.., (X_n, Y_n). We wish to predict new cases (X_{new},Y_{new}) in the future, in which X_{new} is known but Y_{new} needs to be predicted.

So, we may wish to predict human weight Y from height X, or from height and age in a 2-component vector X. Say we have the latter situation, and data on n = 100 people. Then for instance X_{23} would be the vector of height and age for the 23rd person in our training data, and Y_{23} would be that person's weight.

The vector X may include *indicator* variables, which have values only 1 or 0. We may for instance predict weight from height, age and gender, the latter being 1 for female, 0 for male.

If Y represents a binary variable, we represent it as an indicator variable. In the famous Pima Diabetes dataset in the UCI Machine Learning Repository, 1 means diabetic, 0 means not.

If Y itself is categorical, we represent it by several indicator variables, one for each category. In another disease-related UCI dataset, Y is status of a person's vertebrae condition; there are three classes: normal (NO), disk hernia (DH), or spondilolysthesis (SP). Y = (1,0,0) for a normal person, for instance. Thus Y can be a vector too.

Running example

The package's built-in dataset **mlb** consists of data on major league baseball players (courtesy of the UCLA Dept. of Statistics).

Here is a glimpse of the data:

```
> data(mlb)
> head(mlb)
             Name Team       Position Height Weight   Age PosCategory
1   Adam_Donachie  BAL        Catcher     74    180 22.99     Catcher
2       Paul_Bako  BAL        Catcher     74    215 34.69     Catcher
3 Ramon_Hernandez  BAL        Catcher     72    210 30.78     Catcher
4    Kevin_Millar  BAL  First_Baseman     72    210 35.43   Infielder
5     Chris_Gomez  BAL  First_Baseman     73    188 35.71   Infielder
6   Brian_Roberts  BAL Second_Baseman     69    176 29.39   Infielder
```

The R package's qe*-series functions

Here "qe" stands for **"quick and easy,"** and we really mean that!

The functions have a simple, uniform interface, and most importantly, **require no setup.** To fit an SVM model, say, one simply calls **qeSVM**, no preparation calls to define the model etc.

The call form is

```
model fit <- qe<model name>(<data name>,<Y name>)
```

As noted, no prior calls are needed to define the model, etc.

Example

Let's predict weight from height and age, using two methods, k-Nearest Neighbor and random forests. (Details of the methods will be explained shortly; for now, let's just see how to invoke them.)

```
mlb1 <- mlb[,4:6]  # columns for height, weight and age
knnout <- qeKNN(mlb1,'Weight')  # fit k-Nearest Neighbor model
rfout <- qeRF(mlb1,'Weight')  # fit random forests model
```

Most methods have *hyperparameters*, values that can be used to tweak the analysis in various ways. In **qeML**, default values of hyperparameters are set but can be overridden.

Prediction of new cases is equally easy, in the form

```
predict(<model fit>, <new X value>)
```

E.g. to predict the weight of a new player of height 70 and age 28, using our random forests model created above, run

```
> predict(rfout,c(70,28))
       2
184.1626
```

Such a player would be predicted to weigh about 184 pounds.

The data is partitioned into a training set and a holdout set, with the model being fit on the former and then tested on the latter.

```
> rfout$testAcc  # mean absolute prediction error
[1] 15.16911
```

So, on average, our predictions are off by about 15 pounds. What about the k-NN model?

```
> knnout$testAcc
[1] 13.20277
```

Seems to be better, though we should try other values of the hyperparameters.

(The size of the holdout set size can be set differently from the default, or suppressed entirely.)

Regression and classification problems, regression functions

Prediction applications in which Y is a continuous variable, say weight, or at least ordinal, are called *regression settings*. Applications in which Y is categorical, i.e. Y is a factor variable in R, say predicting the player's position (e.g. pitcher) are *classification settings*.

Somewhat confusingly, both settings make use of the *regression function*, m(t) = E(Y | X = t), the mean value of Y in the subpopulation defined by X = t. If say we are predicting weight in the **mlb** data, then for instance m(71,23) would be the mean weight among all players of height 71 inches and 23 years old. To predict the weight of a new player, say height 77 and age 19, we use m(77,19).

In classification problems, Y is converted to a set of indicator variables. For the position 'pitcher' in the **mlb** data, we would have Y = 1 or 0, depending on whether the player is a pitcher or not. (Position is in column 3 of the original dataset.) Then E(Y | X = t) reduces to P(Y = 1 | X = t), the probability that the player is a pitcher given the player's height, weight and age, say. We would typically find probabilities for each position, then guess the one with the highest probability.

In other words, the regression function m(t) is central to both regression and classification settings. The statistical/machine learning methods presented here amount to ways to estimate m(t). The methods are presented below in an order that shows connection between them.

Even though each ML method has its own special *tuning parameters* or *hyperparameters*, used to fine-tune performance, they all center around the regression function m(t).

The **qe** series function sense whether the user is specifying a regression setting or a classification setting, by noting whether the Y variable (second argument) is numeric or an R factor.

ML predictive methods

We now present the "30,000 foot" view of the major statistical/machine learning methods.

k-Nearest Neighbors

This method was originally developed by statisticians, starting in the 1950s and 60s.

It's very intuitive. To predict, say, the weight of a new player of height 72 and age 25, we find the k closest players in our training data to (72,25), and average their weights. This is our estimate of m(72,25), and we use it as our prediction. The default value of k in **qeKNN** is 25.

The **qeKNN** function wraps **kNN** in **regtools**. The main hyperparameter is the number of neighbors k. As with any hyperparameter, the user aims to set a "Goldilocks" level, not too big, not too small. Setting k too small will result in our taking the average of just a few Y values, too small a sample. On the other hand, too large a value for k will some distant data points may be used that are not representative.

To choose k, we could try various values, run **qeKNN** on each one, then use the value that produced the best (i.e. smallest) **testAcc**.

If Y is an indicator variable, its average works out to be the proportion of 1s. This will then be the estimated probability that Y = 1 for a new case. If that is greater than 0.5 in the neighborhood of the new case, we guess Y = 1 for the new case, otherwise 0.

If Y is categorical, i.e. an R factor as in the case of predicting player position in the **mlb** dataset, we then find a probability for each category in this manner, and guess Y to be whichever category has the highest probability.

The hyperparameter k as default value 25. Let's see if, say, 10 is better:

```
replicMeans(50,"qeKNN(mlb1,'Weight')$testAcc")
[1] 13.61954
attr(,"stderr")
[1] 0.1346821
replicMeans(50,"qeKNN(mlb1,'Weight',k=10)$testAcc")
[1] 14.25737
attr(,"stderr")
[1] 0.1298224
```

Since the holdout set is randomly generated, we did 50 runs in each case. The average test accuracy over 50 runs is printed out, along with a standard error for the figure. (1.96 times the standard error will be the radius of an approximate 95% confidence interval.) Changing k to 10 reduced accuracy.

The **qeML** function **qeFit** makes it easier to try various values of the hyperparameters.

K-NN edge bias

One potential problem is bias at the edge of a neighborhood. Say we are predict weight from height, and a new case involving a very tall person. Most data points in the neighborhood of this particular height value will be for people who are shorter than the new case. Those neighbors are thus likely to be lighter than the new case, making our predicted value for that case biased downward.

Rare for k-NN implementations, **qeKNN** has a remedy for this bias. Setting the argument **smoothingFtn = loclin** removes a linear trend within the neighborhood, and may improve predictive accuracy for new cases that are located near the edege of the training set.

Random forests

This method stems from *decision trees*, which were developed mainly by statisticians in the 1080s, and which were extended to random forests in 1990s. Note too the related (but unfortunately seldom recognized) *random subspaces* work of Tin Kam Ho, who did her PhD in computer science.

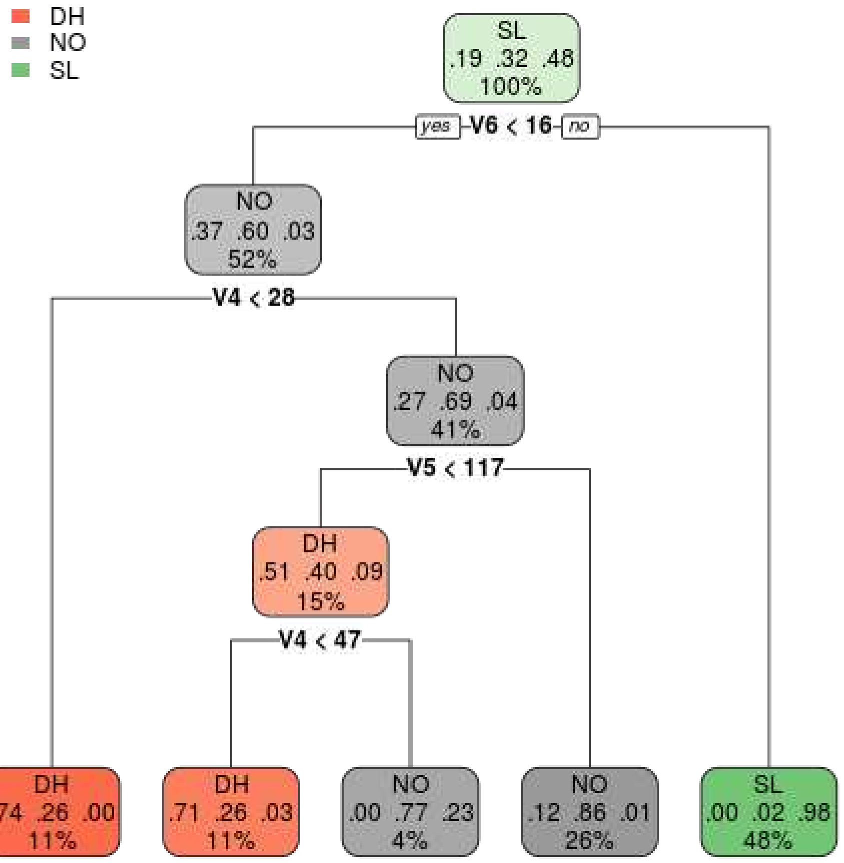
This is a natural extension of k-NN, in that it too creates a neighborhood and averages Y values within the neighborhood. However, it does so in a different way, creating tree structures.

Say in some dataset we are predicting blood pressure from height, weight and age. We first ask whether the height is above or below a certain threshold. After that, we ask whether weight is above or below a certain (different) threshold. This partitions height-weight space into 4 sectors. We then might subdivide each sector according to whether age is above or below a threshold, now creating 8 sectors of height-weight-age space. Each sector is now a "neighborhood." To predict a new case, we see which neighborhood it belongs to, then take our prediction to be the average Y value among training set points in that neighborhood.

The word *might* in the above paragraph alludes to the fact that the process may stop early, if the current subdivision is judged by the algorithm to be fine enough to produce good accuracy. So we might end up using only height and weight, not age.

And one generally wants to avoid having neighborhoods (*nodes* in the tree) that don't have many data points; this is controlled by a hyperparameter, say setting a minimum number of data points per node; if a split would violate that rule, then don't split. Of course, if we set out threshold too high, we won't do enough splits, so again we need to try to find a "Goldilocks" level.

Here is an example using the vertebrae dataset. There are 6 predictor variables, named V1 through V6, consisting of various bone measurements. A single tree is shown.



At the root, if the variable V6 in our new case to be predicted is < 16, we go left, otherwise right. Say we go left. Then if V4 < 28 in the new case, we go left again, getting to a leaf, in which we guess DH. The 0.74 etc. mean that for the training data that happen to fall into that leaf, 74% of them are in class DH, 26% are NO and 0% are SL. So we gues DH, based on there being an estimated 74% chance that this new case is of type DH..

In RF, many trees are generated at ranomd, using for instance random orders of entry of the features. To predict a new case, we find its preditions in the various trees, then average them. For indicator or categorical Y, each try gives a prodiction, and we predict whatever class is most common among the trees.

Clearly, the order in which the predictor variables are evaluated (e.g. height, weight and age in the **mlb** data) can matter a lot. So, more than one tree is constructed, with random orders. The number of trees is another hyperparameter. Each tree gives us a prediction for the unknown Y. In a regression setting, those predictions are averaged to get our final prediction. In a classification setting, we see which class was predicted most often among all those trees.

The thresholds used at each node are determined through a complicated process, depending on which implementation of RF one uses.

The **qeRF** function wraps the function of the same name in the **randomForests** package.

RF edge bias

The package also offers other implementations of RF, notably **qeRFgrf**, as follows.

The leaves in any tree method, such as random forests and boosting, are essentially neighborhoods, different in structure from those of k-NN, but with similar properties. In particular, they have an "edge bias" problem similar to that described for k-NN above. In the case of random forests, the **qeRFgrf** function (which wraps the **grf** package) deals with the same bias problem via removal of a linear trend.

Boosting

This method has been developed both by CS and statistics people. The latter have been involved mainly in *gradient* boosting, the technique used here.

The basic idea is to iteratively build up a sequence of trees, each of which is an improved update of the last. At the end, all the trees are combined, with more weight given to the more recent trees.

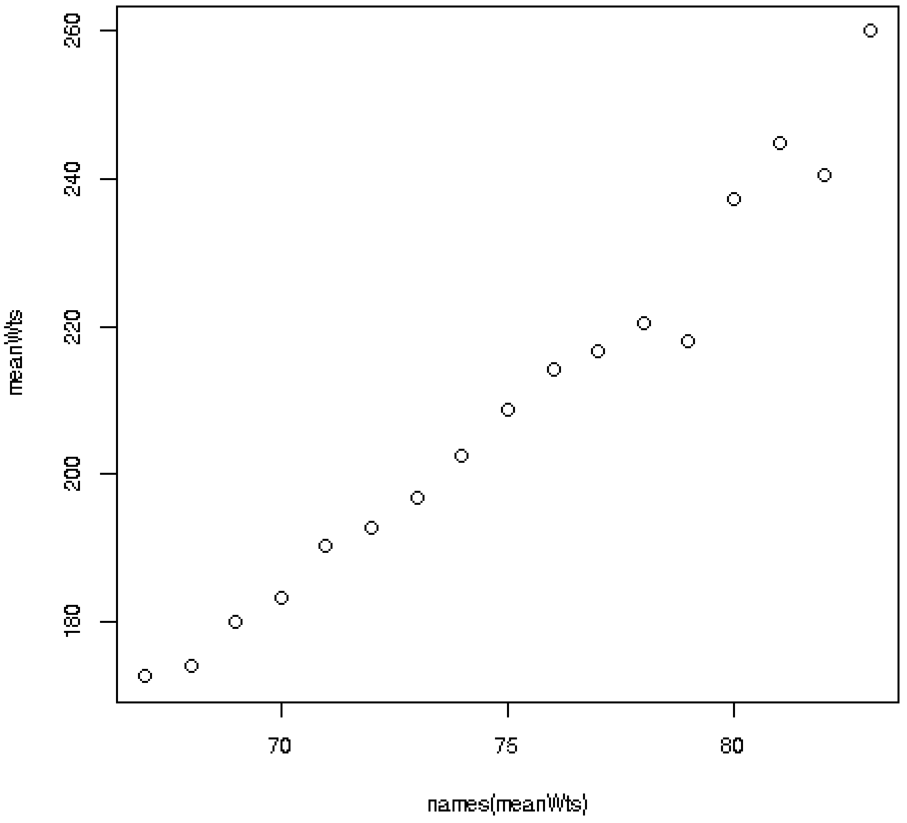
The **qeGBoost** wraps **gbm** in the package of the same name. It is gradient tree-based, with hyperparameters similar to the random forests case, plus a *learning rate*. The latter controls the size of iteration steps; more on this below.

The package also offers other implementations of boosting.

Linear model

This of course is the classical linear regression model, invented 200 years ago (!) and developed by statisticians.

For motivation, below is a graph of mean weight vs. height for the **mlb** data. (There are many players for each height level, and we find and plot the mean weight at each height.)



The means seem to lie near a straight line. (Remember, though, that these are sample means.) That suggests modeling m(t) is a linear function.

For example, a model for mean weight, given height and age, would be

$m(\text{height},\text{age}) = \beta_0 + \beta_1 \text{ height} + \beta_2 \text{ age}$

for unknown population constants β_i, which are estimated from our training data, using the classic *least-squares* approach. Our estimates of the β_i, denoted b_i, are calculated by minimizing

$\Sigma_i \, [\, \text{weight}_i - (b_0 + b_1 \text{height}_i + b_2 \text{age}_i) \,]^2$

This is a simple calculus problem. We find the partial derivatives of the sum of squares with respect to the b_i, and set them to 0. This gives us 3 equations in 3 unknowns, and since these equations are linear, it is easy to solve them for the b_i.

There are no hyperparameters here.

This model is mainly for regression settings, though some analysts use it in classification. If used in conjunction with polynomials (see below), this may work as well or better than the logistic model (see below).

The function **qeLin** wraps the ordinary **lm**. It mainly just calls the latter, but does some little fixess.

Logistic model

This is a generalization of the linear model, developed by statisticians and economists.

This model is only for classification settings. As noted, since Y is now 1 or 0, its mean becomes the probability of 1. Since m(t) is now a probability, we need it to have values in the interval [0,1]. This is achieved by feeding a linear model into the *logistic function*, l(u) = $(1 + \exp(-u))^{-1}$, which does take values in (0,1).
So for instance, to predict whether a player is a catcher (Y = 1 if yes, Y = 0 if no), we fit the model

P(catcher | height, weight, age) = m(height,weight,age) = 1 / [1 + exp{-(β_0 + β_1 height + β_2 weight + β_3 age)}]

The β_i are estimated from the sample data, using a technique called *iteratively reweighted least squares*.

The function **qeLogit** wraps the ordinary R function **glm**, but adds an important feature: **glm** only handles the 2-class setting, e.g. catcher vs. non-catcher. The **qeLogit** handles the c-class situation via the *One vs. All* method (applicable to any ML algorithm):

it calls **glm** one class at a time, generating c **glm** outputs. When a new case is to be predicted, it is fed into each of the c **glm** outputs, yielding c probabilities. It then predicts the new case as whichever class has the highest probability.

Here is an example using the UCI vertebrae data;

```
> library(fdm2id)
> data(spine)
> str(spine)
'data.frame':   310 obs. of  8 variables:
 $ V1      : num  39.1 53.4 43.8 31.2 48.9 ...
 $ V2      : num  10.1 15.9 13.5 17.7 20 ...
 $ V3      : num  25 37.2 42.7 15.5 40.3 ...
 $ V4      : num  29 37.6 30.3 13.5 28.9 ...
 $ V5      : num  114 121 125 120 119 ...
 $ V6      : num  4.56 5.99 13.29 0.5 8.03 ...
```

```
   $ Classif2: Factor w/ 2 levels "AB","NO": 1 1 1 1 1 1 1 1 1 1 ...
   $ Classif3: Factor w/ 3 levels "DH","NO","SL": 1 1 1 1 1 1 1 1 1 1 ...
  > spine <- spine[,-7]  # skip 2-class example
  > u <- qeLogit(spine,'Classif3')
  > u$testAcc
  [1] 0.1935484
  > u$baseAcc
  [1] 0.5053763

  > table(spine$Classif3)

   DH  NO  SL
   60 100 150
```

If we were to not use the body measurements V1 etc., we would always guess SL, resulting in an error rate of about 50%. By making use of the measurement data, we can reduce the misclassification rate to about 19%.

Let's try a prediction. Consider someone like the first patient in the dataset but with V6 = 6.22. What would our prediction be?

```
  > newx <- spine[1,-7]  # omit "Y"
  > newx$V6 <- 6.22
  > predict(u,newx)
  $predClasses
  [1] "DH"

  $probs
              DH        NO         SL
  [1,] 0.7432193 0.2420913 0.01468937
```

We would predict the DH class, as our estimated probability for the lass is 0.74, the largest among the three classes.

Some presentations describe the logistic model as "modeling the logarithm of the odds of Y = 1 vs. Y = 0 as linear." While this is correct, it is less informative, in our opinion. Why would we care about a logarithm being linear? The central issue is that the logistic function models a probability, just what we need.

Polynomial-linear models

Some people tend to shrink when they become older. Thus we may wish to model a tendency for people to gain weight in middle age but then lose weight as seniors, a nonlinear relation. We could try a quadratic model:

$m(\text{height},\text{age}) = \beta_0 + \beta_1 \text{ height} + \beta_2 \text{ age} + \beta_3 \text{ age}^2$

where presumably $\beta_3 < 0$.

We may even include a height X age product term, allowing for interactions. Polynomials of degree 3 and so on could also be considered. The choice of degree is a hyperparameter; in **qeSVM** it is named, of course, **degree**.

This would seem nonlinear, but that would be true only in the sense of being nonlinear in age. It is still linear in the β_i -- e.g. if we double each β_i in the above expression, the value of the expression is doubled -- so **qeLin** can be used, or **qeLogit** for classification settings.

Forming the polynomial terms by hand would be tedious, especially since we would also have to do this for predicting new cases. Instead, we use **qePolyLin** (regression setting) and **qePolyLog** (classification). They make use of the package **polyreg**.

Polynomial models can in many applications hold their own with the fancy ML methods. One must be careful, though, about overfitting, just as with any ML method. In particular, polynomial functions tend to grow rapidly near the edges of one's dataset, causing both bias and variance problems.

Shrinkage methods for linear/generalized linear models

Some deep mathematical theory implies that in linear models it may be advantageous to shrink the estimated b_i. *Ridge* regression and the LASSO do this in a mathematically rigorous manner. Each of them minimizes the usual sum of squared prediction errors, subject to a limit being placed on the size of the b vector; for ridge, the size is defined as the sum of the b_i^2, while for the LASSO it's the sum of {b_i|.

The function **qeLASSO** wraps **cvglmnet** in the **glmnet** package. The main hyperparameter **alpha** specifies ridge (0) or the LASSO (1, the default).

There are various other shrinkage methods, such as Minimax Convex Penalty (MCP). This and some others are available via **qeNCVregCV**, which wraps the **ncvreg** package.

Shrinkage methods are often also applied to other ML algorithms.

The LASSO tends to produce solutions in which some of the b_i are 0. Thus it is popular as a tool for predictor variable selection.

LASSO for feature selection

The LASSO tends to produce solutions in which some of the b_i are 0. Thus it is popular as a tool for predictor variable selection.

Here is an example using **pef**, a dataset from the US Census, included with **qeML**. The features consist of age, education, occupation and gender. Those last three are categorical variables, which are converted to indicator varaibles. Let's predict wage income.

```
> w <- qeLASSO(pef,'wageinc')
> w$coefs
11 x 1 sparse Matrix of class "dgCMatrix"
                   s1
(Intercept) -8983.986
age           254.475
educ.14      9031.883
educ.16      9182.592
occ.100     -2707.388
occ.101     -1029.592
occ.102      3795.166
occ.106         .
occ.140         .
sex.1        4472.672
wkswrkd      1178.889
```

There are six occupations (this dataset is just for programmers and engineers),

```
> levels(pef$occ)
[1] "100" "101" "102" "106" "140" "141"
```

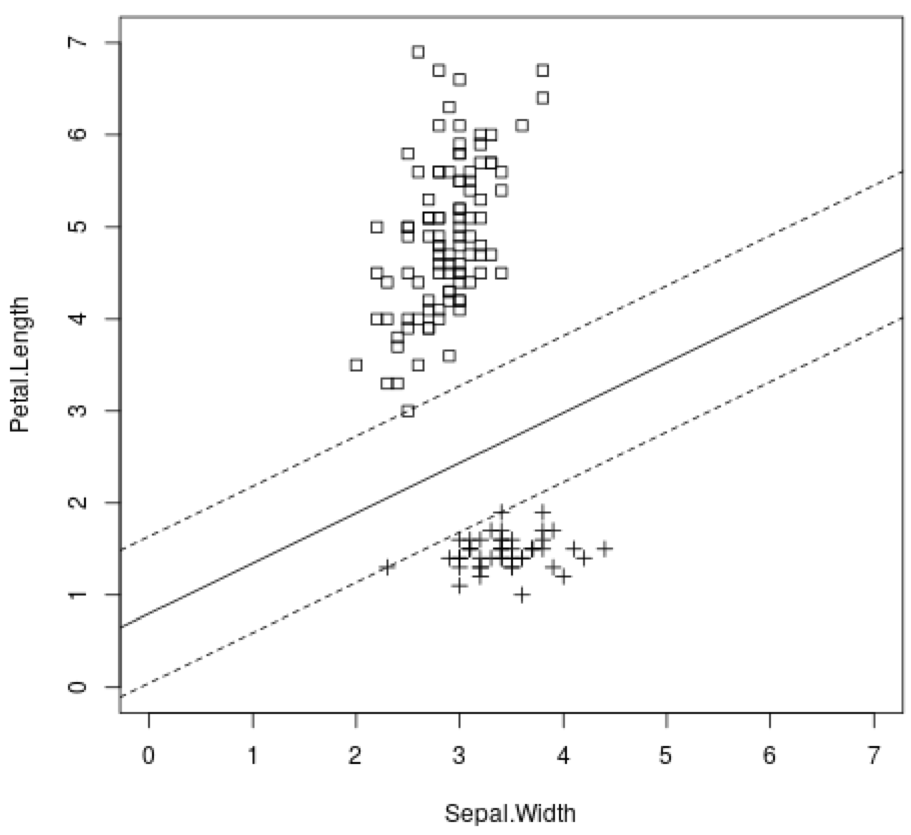
thus five indicator variables. The LASSO gave coefficients of 0 for occupations 106 and 140, so we might decide not to use them, even if we utimately use some other ML algorithm.

Support Vector Machines

These were developed originally in the AI community, and later attracted interest among statisticians. They are used mainly in classification settings.

Say in the baseball data we are predicting catcher vs. non-catcher, based on height and weight. We might plot a scatter diagram, with height on the horizontal axis and weight on the vertical, using red dots for the catchers and blue dots for the non-catchers. We might draw a line that best separates the red and blue dots, then predict new cases by observing which side of the line they fall on. This is what SVM does (with more predictors, the line become a plane or hyperplane).

Here is an example, using the Iris dataset built in to R:



There are 3 classes, but we are just predicting setosa species (shown by + symbols) vs. non-setosa (shown by boxes) here. Below the solid line, we predict setosa, otherwise non-setosa.

SVM philosophy is that we'd like a wide buffer separating the classes, called the *margin*, denoted by the dashed lines. Data points lying on the edge of the margin are termed *support vectors*, so called because if any other data point were to change, the margin would not change.

In most cases, the two classes are not linearly separable. So we allow curved boundaries, implemented through polynomial (or similar) transformations to the data. The degree of the polynomial is a hyperparameter.

Another hyperparameter is **cost:** Here we allow some data points to be within the margin. The cost variable is roughly saying how many exceptions we are willing to accept.

The **qeSVM** function wraps **svm** in the **e1071** package. The package also offers various other implementations.

Neural networks

These were developed almost exclusively in the AI community.

An NN consists of *layers*, each of which consists of a number of *neurons* (also called *units* or *nodes*). Say for concreteness we have 10 neurons per layer. The output of the first layer will be 10 linear combinations of the predictor variables/features, essentially 10 linear regression models. Those will be fed into the second layer, yielding 10 "linear combinations of linear combinations," and so on.

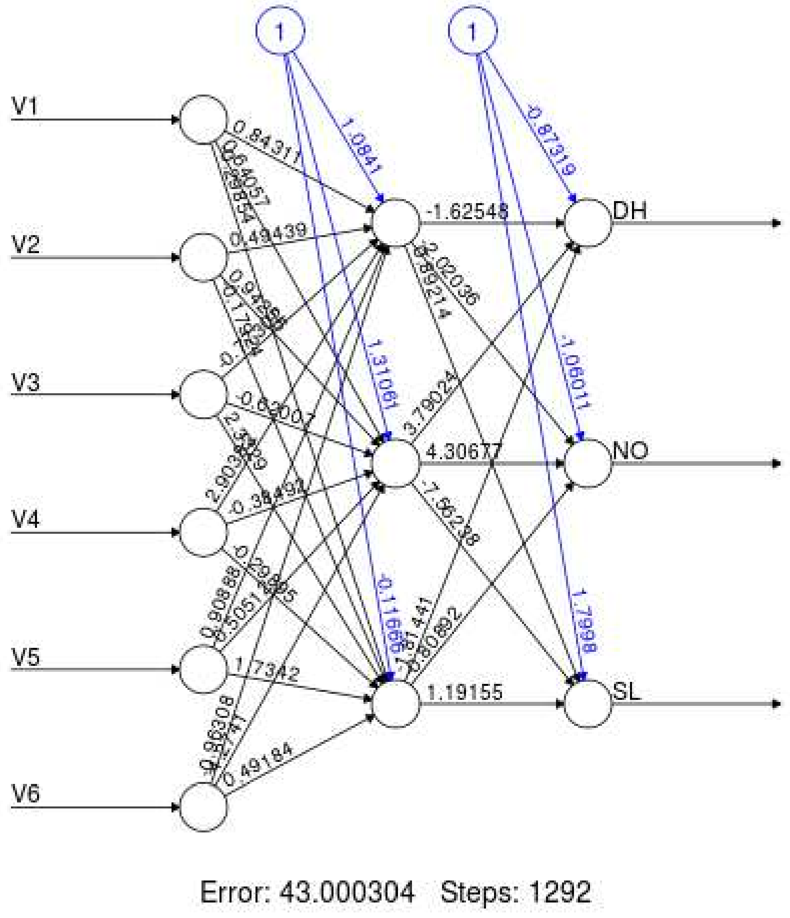
In regression settings, the outputs of the last layer will be averaged together to produce our estimated m(t). In the classification case with c classes, our final layer will have c outputs; whichever is largest will be our predicted class.

"Yes," you say, "but linear combinations of linear combinations are still linear combinations. We might as well just use linear regression in the first place." True, which is why there is more to the story: *activation functions*. Each output of a layer is fed into a function A(t) before being passed on to the next layer; this is for the purpose of allowing nonlinear effects in our model of m(t).

For instance, say we take A(t) = t^2 (not a common choice in practice, but a simple one to explain the issues). The output of the first layer will be quadratic functions of our features. Since we square again at the outputs of the second layer, the result will be 4th-degree polynomials in the features. Then 8th-degree polynomials come out of the third layer, and so on.

One common choice for A(t) is the logistic function l(u) we saw earlier. Another popular choice is ReLU, r(t) = max(0,t). No matter what we choose for A(t), the point is that we have set up a nonlinear model for m(t).

Here's an example, again with the Vertebrae data: The predictor variables V1, V2 etc. for a new case to be predicted enter on the left, and the predictions come out the right; whichever of the 3 outputs is largest, that will be our predicted class.



The first layer consists of V1 through V6. The second layer, our only *hidden* layer here, has three neurons. Entering on the left of each neuron is a linear combination of V1 through V6. The outputs are fed into A(t) and then to the third layer.

Hyperparameters for NNs include the number of layers, the number of units per layer (which need not be the same in each layer), and the activation function. Most NN software also includes various other hyperparameters.

The linear combination coefficients, shown as numeric labels in the picture, are known as *weights*. How are they calculated? Again least squares is used, minimizing

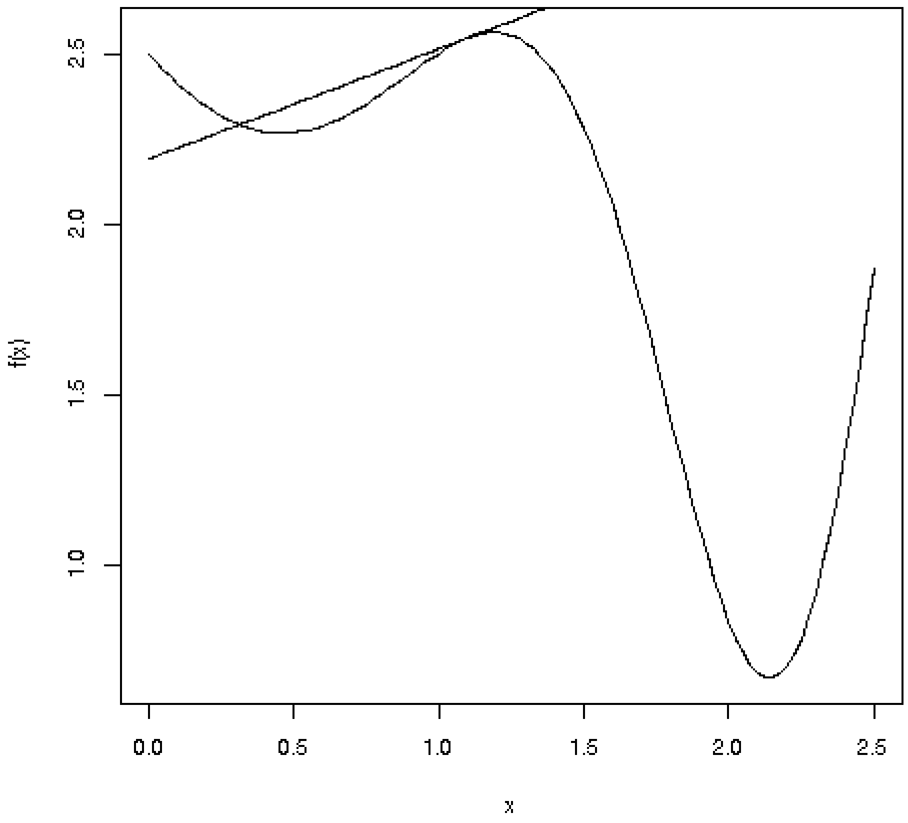
$\Sigma_i (Y_i - \text{finaloutput}_i)^2$

Let n_w denote the number of weights. This can be quite large, even in the millions. Moreover, the n_w equations we get by setting the partial derivatives to 0 are not linear.

Thus this is no longer a "simple" calculus problem. Iterative methods must be used, and it can be extremely tricky to get them to converge. Here's why:

Though far more complex than in the linear case, we are still in the calculus realm. We compute the partial derivatives of the sum of squares with respect to the n_w weights, and set the results to 0s. So, we are finding roots of a very complicated function in n_w dimensions, and we need to do so iteratively.

A simplified version of the iteration process is as follows. Consider the function f graphed below:



There is an overall minimum at approximately x = 2.2. This is termed the *global minimum*. But there is also a *local minimum*, at about x = 0.4; that term means that this is the minimum value of the function only for points near---"local to"--- 0.4. Let's give the name x_0 to the value of x at the global minimum.

Denote our guess for x_0 at iteration i by g_i. Say our initial guess g_0 = 1.1.

The tangent line is pointing upward to the right, i.e. has positive slope, so it tells us that by going to the left we will go to smaller values of the function. We do want smaller values, but in this case, the tangent is misleading us. We should be going to the right, towards 2.2, where the global minimum is.

You can see that if our current guess were near 2.2, the tangent line would guide us in the right direction. But we see above that it can send us in the wrong direction. One remedy (among several typically used in concert) is to not move very far in the direction the tangent line sends us. The idea behind this is, if we are going to move in the wrong direction, let's limit our loss. The amount we move is called the *step size* in general math, but the preferred ML term is the *learning rate*. And, this is yet another hyperparameter.

So, NNs are arguably the most complex of all the methods described here, and tend to use huge amounts of computing time, even weeks!

The **qeNeural** function allows specifying the numbers of layers and neurons per layer, and the number of iterations. It wraps **krsFit** from the **regtools** package, which in turn wraps the R **keras** package (and there are further wraps involved after that).

Since any continuous activation function can be approximated by a polynomial, the outputs of the layers are approximately polynomials, of higher and higher degree with each layer. Thus they are subject to the same issues on the edges of the dataset as were described for polynomial models above.

Overfitting

Up to a point, the more complex a model is, the greater its predictive power. "More complex" can mean adding more predictor variables, using a higher-degree polynomial, adding more layers etc.

As we add more and more complexity, the *model bias* will decrease, meaning that our models become closer to the actual m(t), in principle. But the problem is that at the same time, the *variance* of a predicted Y_{new} is increasing.

Say again we are predicting human weight from height, with polynomial models. With a linear model, we use our training data to estimate two coefficients, b_0 and b_1. With a quadratic model, we estimate three, and estimate four in the case of a cubic model and so on. But it's same training data in each case, and intuitively we are "spreading the data thinner" with each more complex model. As a result, the standard error of our predicted Y_{new} (estimated standard deviation) increases.

Hence the famous Bias-Variance Tradeoff. If we use too complex a model, the increased variance overwhelms the reduction in bias, and our predictive ability suffers. We say we have *overfit*.

So there is indeed a "Goldilocks" level of complexity, an optimal polynomial degree, optimal number of nearest neighbors, optimal *network architecture* (configuration of layers and neurons), and so on. How do we find it?

Alas, **there are no magic answers here**. But various approaches have been developed that one can try. See the **qeML** vignettes **Feature_Selection** and **Overfitting**.

Which ML method to use?

The usual answer given for this question is, "The best method is very dependent on which dataset you have." True, but are there some general principles?

ML people often distinguish between "tabular" and "nontabular" data. A disease diagnosis application may be considered tabular--rows of patients, and columns of measurements such as blood glucose, body temperature and so on. On the other hand, a facial recognition application is considered nontabular, which is confusing, since it too has the form of a table--one row per picture, and one column per pixel position.

Where the two kinds of applications may differ, though, is whether there are general monotonic relationships between Y and the features. In predicting diabetes, say, the higher the glucose level, the more likely the patient is diabetic. Though monotonic relations may exist in image classification, they maybe more localized.

The point then, is that generally "monotonic" applications may be best served by ML methods that have either linear components or low-degree polynomials, such as linear or generalized linear models, SVM and polynomial versions of these. For other applications, one might try k-NN, NNs etc.

Visualising Categorical Data

To visualise categorical (or qualitative) variables, we will be using the Berkeley Admissions dataset.This dataset is included in the R datasets package. It contains graduate school applicants to the six largest departments at University of California, Berkeley in 1973. To learn more about the dataset, use the help function.

Why is this dataset interesting? This dataset is useful for demonstrating how we can visualise categorical data using fourfold plots and cotab plots. It is also an example of Simpson's paradox. This can be explained as a phenomenon where a trend appears in groups of data but disappears or reverses when combined with another group of data.

In the UCBAdmissions dataset, when we look at the **Admit** and **Gender** variables, there appears to be bias towards the number of men being admitted, with women having a lower acceptance rate overall. When we compare **Admit** and **Gender** with **Dept**, this bias disappears and we can see that the admission rates are similar for males and females in most departments, except A.

We will explore the data using the **vcd** (Visualising Categorical Data) package.

```
# Load the dataset
data(UCBAdmissions)

# Help page
?UCBAdmissions
```

Now let's look at the structure of the dataset.

```
dim(UCBAdmissions)
```

```
## [1] 2 2 6
```

```
dimnames(UCBAdmissions)
```

```
## $Admit
## [1] "Admitted" "Rejected"
##
## $Gender
## [1] "Male"   "Female"
##
## $Dept
## [1] "A" "B" "C" "D" "E" "F"
```

```
str(UCBAdmissions)
```

```
##  table [1:2, 1:2, 1:6] 512 313 89 19 353 207 17 8 120 205 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ Admit : chr [1:2] "Admitted" "Rejected"
##   ..$ Gender: chr [1:2] "Male" "Female"
##   ..$ Dept  : chr [1:6] "A" "B" "C" "D" ...
```

The applicants are classified by **Admit** (either Admitted or Rejected), **Gender** (either Male or Female) and **Department** (A to F). The data forms a 3-way table (2 x 2 x 6).

First, let's examine the relationship between **Admit** and **Gender** using a two-way frquency table.

```
UCB.GA <- margin.table(UCBAdmissions, c(1,2))
UCB.GA
```

```
##           Gender
## Admit      Male Female
##   Admitted 1198    557
##   Rejected 1493   1278
```

There seems to be a difference between the number of females and males that are admitted. Let's create a cross table using the **gmodels** package.

```
univ <- apply(UCBAdmissions, c(1,2), sum)
univ
```

```
##           Gender
## Admit      Male Female
##   Admitted 1198    557
##   Rejected 1493   1278
```

```
prop.table(univ, 2)
```

```
##           Gender
## Admit           Male    Female
##   Admitted 0.4451877 0.3035422
##   Rejected 0.5548123 0.6964578
```

We can see that the proportion of males admitted is 44.5%, compared to 30.4% of females. It seems there may be some bias here. Let's take a closer look at the odds ratio.

What is the odds ratio?

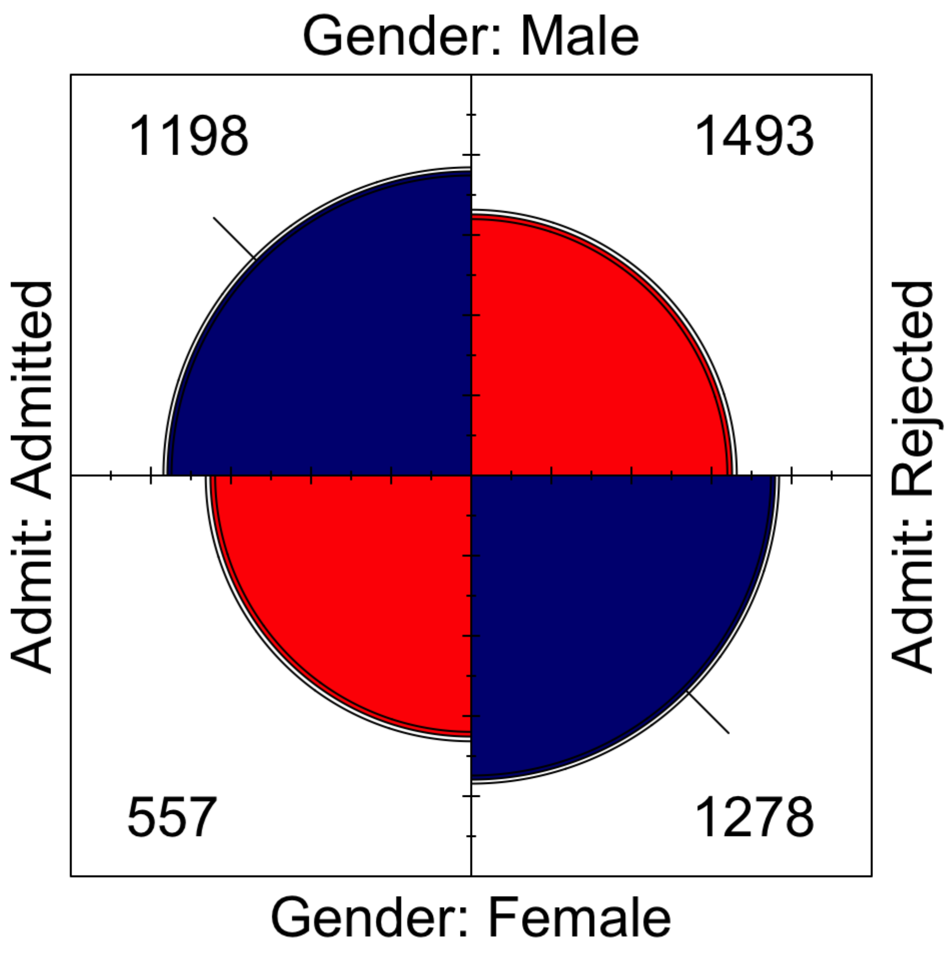
The odds ratio is the probability of success over failure. In this case, it is the probability that one is admitted versus the probability of being rejected by department, given their gender.

Let's look a two-way plot first of **Admit** and **Gender**, ignoring the department that they applied to.

```
# load the vcd package
library(vcd)
```

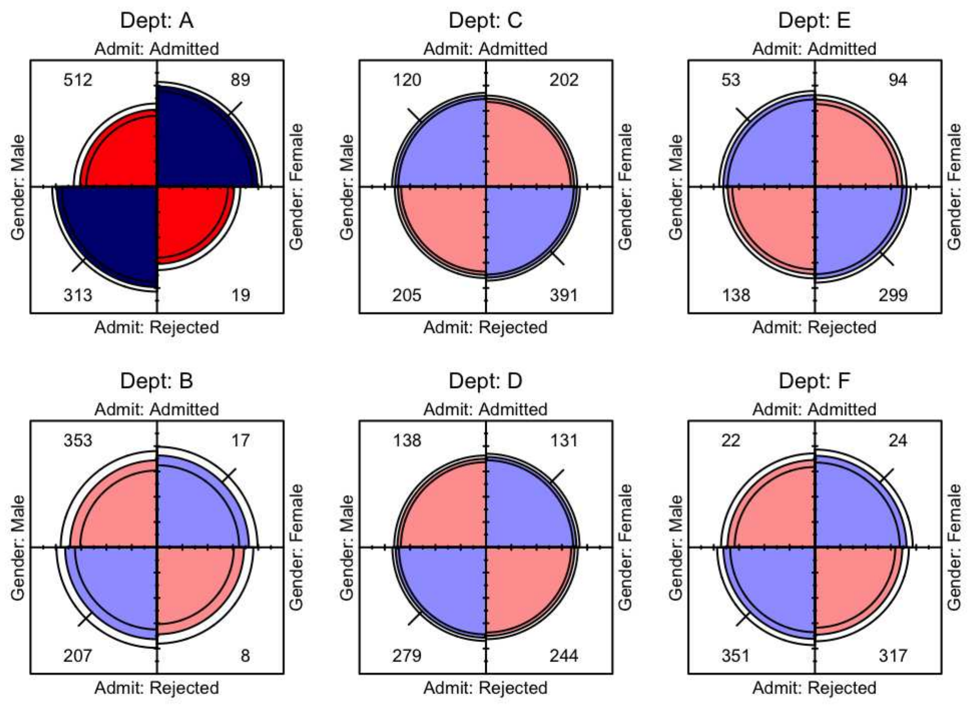
```
## Loading required package: grid
```

```
# fourfold plot
UCB <- aperm(UCBAdmissions, c(2,1,3))
fourfold(margin.table(UCB, c(1, 2)))
```



Now let’s look at a three-way plot.

```
fourfold(UCBAdmissions, mfrow=c(2,3))
```



As we can see, when department is excluded there is quite a different story being told. Admission is the response variable, whilst gender and department are explanatory variables.

When we look at the admission of males and females by department, we can that the admission of males and females is quite similar, with the exception of department A. Within department A, more females are admitted than males. Proportionally, more females are also accepted with departments B, D and F.

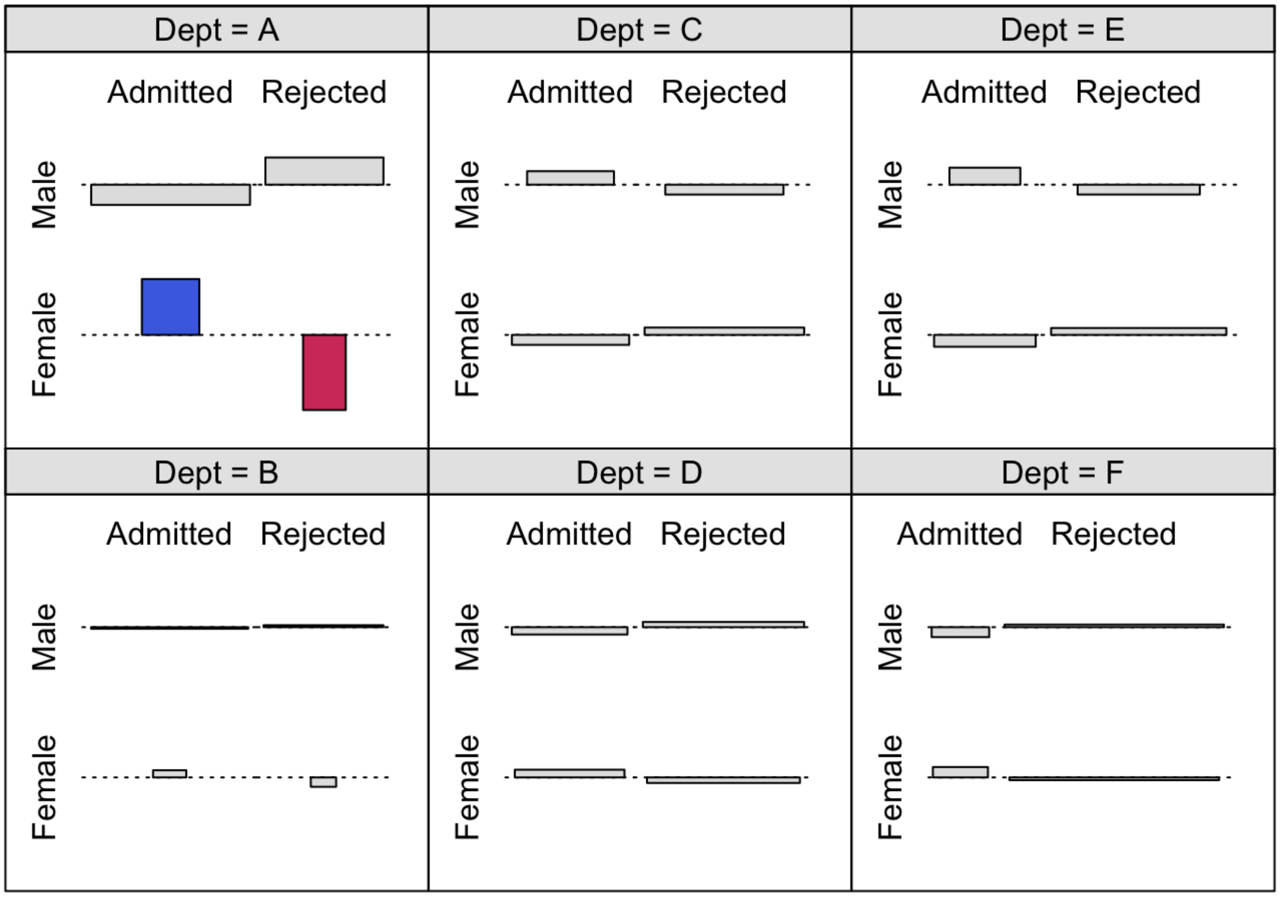
To check this, let's create another frequency table with the proportion of males and females accepted by department.

```
ftable(round(prop.table(UCBAdmissions, c(2,3)), 2),
       row.vars="Dept", col.vars = c("Gender", "Admit"))
```

```
##      Gender     Male              Female
##      Admit  Admitted Rejected Admitted Rejected
## Dept
## A               0.62     0.38     0.82     0.18
## B               0.63     0.37     0.68     0.32
## C               0.37     0.63     0.34     0.66
## D               0.33     0.67     0.35     0.65
## E               0.28     0.72     0.24     0.76
## F               0.06     0.94     0.07     0.93
```

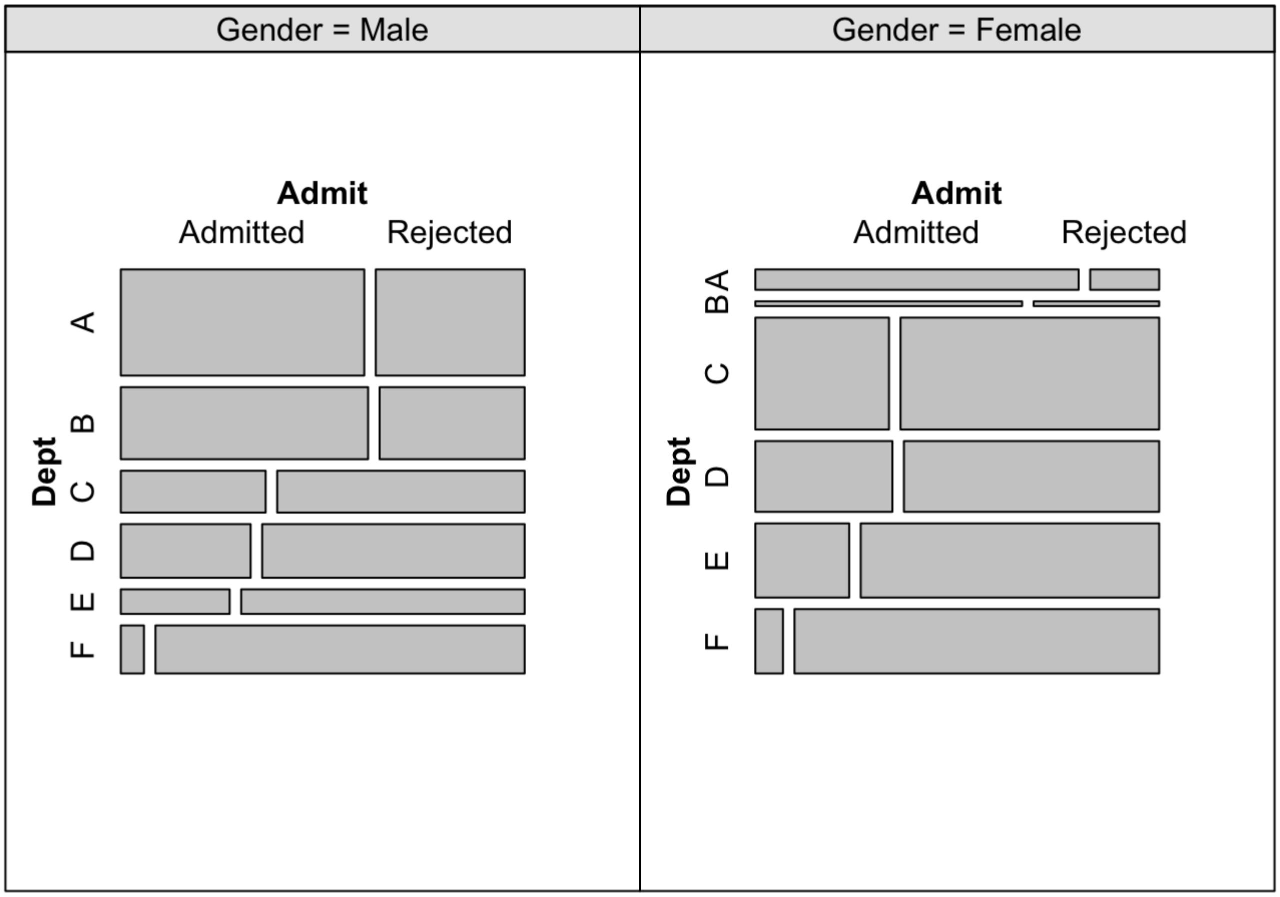
We can also look at the residuals by department using a cotabplot. This highlights the large residuals in department A.

```
cotabplot(aperm(UCBAdmissions, c(2,1,3)), panel = cotab_coindep, shade = TRUE,
          legend = FALSE,
          panel_args = list(type = "assoc", margins = c(2,1,1,2), varnames = FALSE))
```



Now, let’s look at this split by gender.

```
UCB_G <- structable(Gender ~ Dept + Admit, data =UCBAdmissions)
cotabplot(UCB_G)
```



This allows us to clearly see that there are more males who apply to departments A and B, where more females apply for departments C, D, E, and F. By visualising the data we are able to better understand the relationship between the three variables: Admit, Gender and Dept.

If we take another look at the proportional values, we can see that departments A and B admit approximately 2 in 3 applicants of either gender, whilst departments C:F admit 1 in 3 or fewer.

```
admit <- apply(UCBAdmissions, c(2,3,1), sum)
admit
```

```
## , , Admit = Admitted
##
##         Dept
## Gender     A   B   C   D  E  F
##   Male   512 353 120 138 53 22
##   Female  89  17 202 131 94 24
##
## , , Admit = Rejected
##
##         Dept
## Gender     A   B   C   D   E   F
##   Male   313 207 205 279 138 351
##   Female  19   8 391 244 299 317
```