Prediction!

Contents

Ok, so I think I start off a bit loose to get people comfortable and bring out some playing cards.

- I'll point to someone and ask them to guess the suit of the next card I show.
- Reshuffle and repeat giving them five cards/guesses.
- Then I'll repeat with another three-four students.
- # correctly guessed will be noted somewhere

What's the point? How can I best predict the number of card suits the next person will get right?

Prediction

Goal: Predict a new value of a variable

- Ex: Another student will be guessing. Define $Y = \#$ of card suits guessed correctly from the five. What should we guess/predict for the next value of Y?
- Chat with them.
- Lead them to talk about a sample.
- Simulate values of Y using an app, placing them on a histogram or dot plot.
- Lead them to ideas of using something like the sample mean or median as the predicted value.
- Why something like the sample mean or sample median? What are we really trying to do? Find a value that is 'close' to the most values, i.e. something in the center being the most logical thing to do.

Loss function

Let's assume we have a sample of n people that each guessed five cards. Call these values y_1, y_2, ..., y_n.

Need: A way to quantify how well our prediction is doing... Suppose there is some best prediction, call it c. How do we measure the quality of c?

- Using the idea that we want something 'close' to all points, we find a way to compare each point to our prediction.
- Think about things like:

$$y_1 - c, (y_1 - c)^2, |y_1 - c|$$

$$\sum_{i=1}^{n}(y_i - c), \sum_{i=1}^{n}(y_i - c)^2, \sum_{i=1}^{n}|y_i - c|$$

- Quick app to look at how the measures work on the data set already simulated - give the ability to pick a c and at least these metrics, but maybe allow any metric to be typed in...
- In the end an objective function must be created to minimize that uses a 'Loss function', and we'll talk about why we'll use the common squared error loss:

$$g(y_1, ..., y_n) = \sum_{i=1}^{n} L(y_i, c) = \sum_{i=1}^{n} (y_i - c)^2$$

Can we choose an 'optimal' value for c to minimize this function? Calculus to the rescue!

Steps to minimize a function with respect to c:

1. Take the derivative with respect to c
2. Set the derivative equal to 0
3. Solve for c to obtain the potential maximum or minimum
4. Check to see if you have a maximum or minimum (or neither)

Answer comes out to be $\bar{y}$ as the minimizer.

Big wrap: This means that the sample mean is the best prediction when using squared error loss.

Using a Population Distribution

Rather than using sample data, suppose we think about the theoretical distribution for $Y = \#$ of card suits guessed correctly from the five. What might we use here? What assumptions do we need to make this distribution reasonable?

- $Y \sim Bin(5, 0.25)$ assuming we have independent and identical trials
- This gives

$$p_Y(y) = P(Y = y) = \binom{n}{y} p^y (1-p)^{n-y} = \binom{5}{y} 0.25^y 0.75^{5-y}$$

  for $y = 0, 1, 2, ..., n$ or $y = 0, 1, 2, 3, 4, 5$

Is there an optimal value c for the **expected value** of the loss function?

That is, can we minimize (as a function of c) $E\left[(Y-c)^2\right]$?

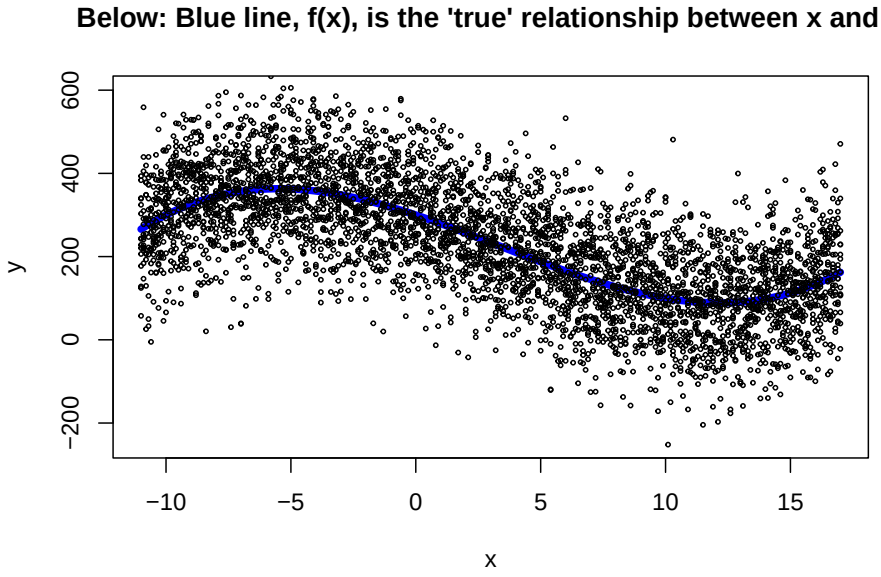
- Maybe visualize this in the same app as above (visualize the quadratic function weighted by the distribution and allow for c to be chosen)
- I think I'll start them out on this one as theory leads to more difficult ideas and I'm not sure if you did general expected values.
- In the end though we end up with np or 1.25.
- I'll show/discuss that this works generally for any distribution $p_Y(y)$ that has a mean
- Discuss the relationship with this and a sample (1/n weight for each point vs $p_Y(y)$ weight for each.)
- **Big idea:** This implies that μ is the best predictor to use if you are considering minimizing the expected squared error loss.

This would be the 2nd (short day) material.

- Recap the big idea of prediction:
  - Need to quantify how well we are doing (squared error loss)
  - Sample mean is optimal if we have a sample
  - Given a theoretical distribution, expected squared error loss is optimized at the mean of the distribution
- Introduce next material with minesweeper, because it is now browser based, nostalgia on my part, and it seems somewhat fun https://minesweeper.online/game/938135731
  - Each student will be assigned a certain number of mines for the board (15x40).
  - They'll click on the first square just below the smiley face. They'll continue to click down one block at a time until they hit a bomb.
  - They'll record in a shared spreadsheet their number of blocks down the first bomb appears.
  - Each person should play 10 games and put their data in.
- Now we'll discuss how we could predict the number of blocks until the first bomb as a function of the number of bombs.
- We'll read in the data to R and do some plotting (I'm not sure what the relationship will be exactly but I'd guess not super linear).

Relating Explanatory Variables in Prediction

Harder (and more interesting) problem is to consider predicting a (response) variable Y as a function of an explanatory variable x.



Y is a random variable and we'll consider the x values fixed (we'll denote this as $Y|x$). We hope to learn about the relationship between Y and x. If we consider squared error loss, we know that $E(Y) = \mu$ minimizes

$$E\left[(Y-c)^2\right]$$

as a function of c. Given data, we used $\hat{\mu} = \bar{y}$ as our prediction.

Now that we have an x, $E(Y|x)$ will still minimize this. We can call this true unknown value

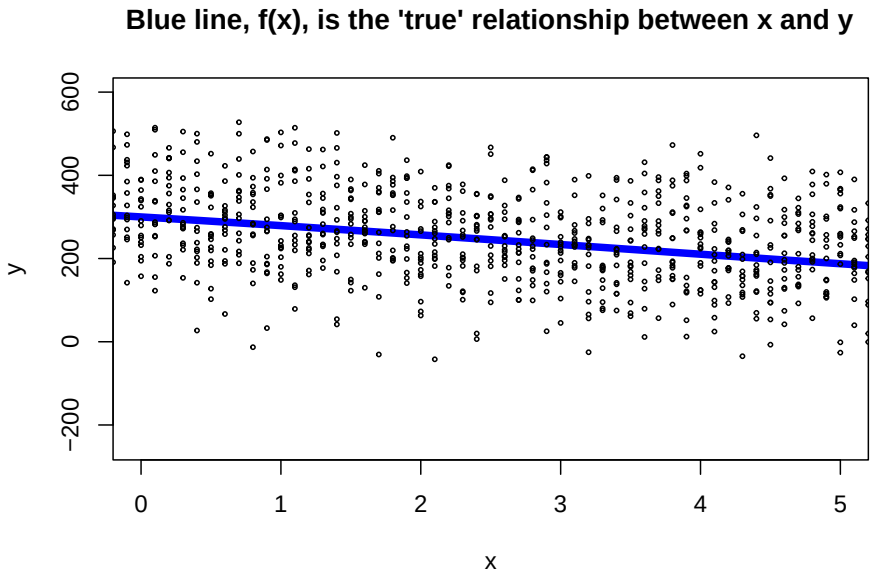
$$E(Y|x) = f(x)$$

Given observed Y's and x's, we can approximate this function and denote that estimate as $\hat{f}(x)$. This $\hat{f}(x)$ will minimize

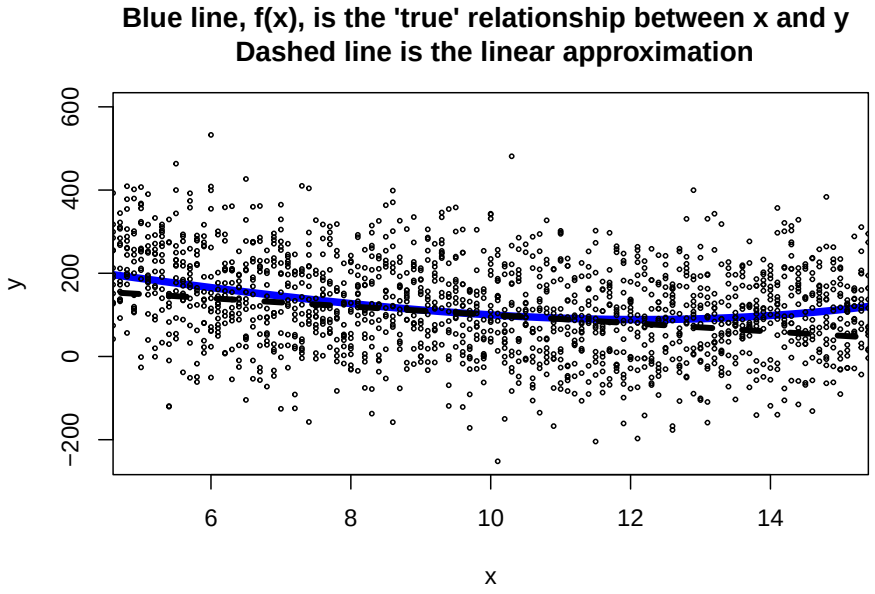
$$g(y_1, ..., y_n | x_1, ..., x_n) = \sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

Approximating $f(x)$

Although the true relationship is most certainly nonlinear, we may be ok approximating the relationship linearly. For example, consider the same plot but between 0 and 5 only:



That's pretty linear. Consider plot between 5 and 15:



Line still does a reasonable job and is often an ok approximation in simple cases.

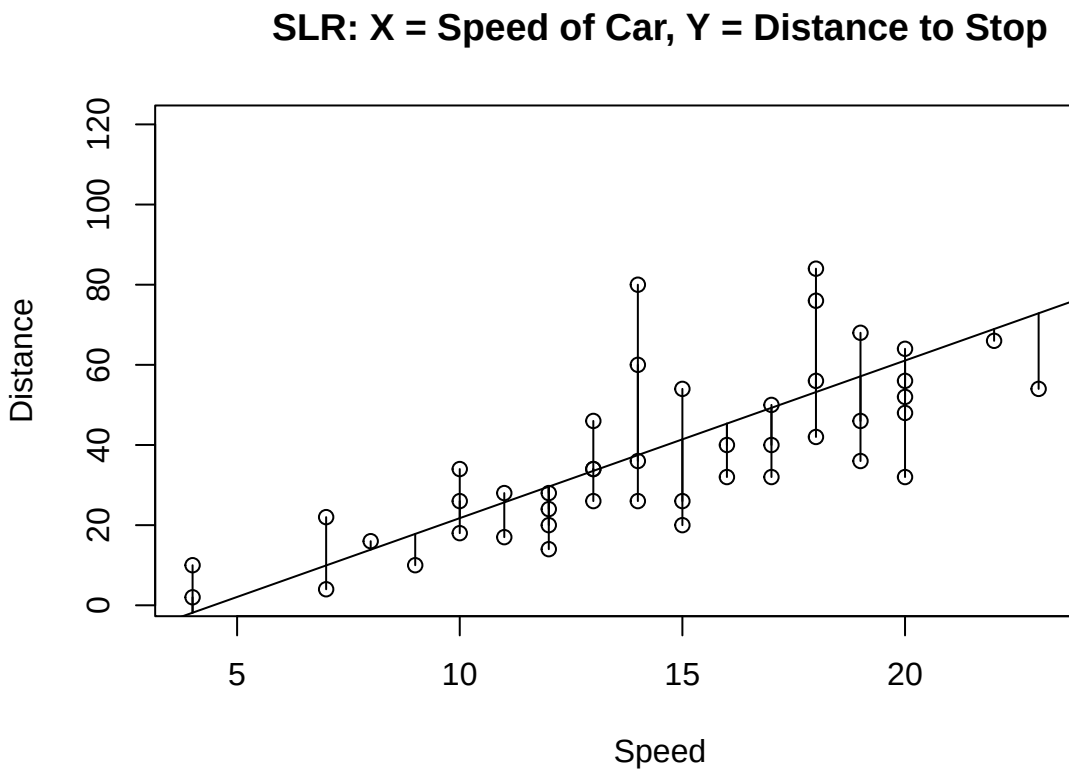
Linear Regression Model

The (fitted) linear regression model uses $\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$. This means we want to find the optimal values of $\hat{\beta}_0$ and $\hat{\beta}_1$ from:

$$g(y_1, ..., y_n | x_1, ..., x_n) = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

This equation is often called the 'sum of squared errors (or residuals)' or the 'residual sum of squares'.

Update this with the data from minesweeper but show a plot like this or perhaps just over a portion that looks



like it could be modeled linearly.

Calculus allows us to find the 'least squares' estimates for β_0 and β_1 in a nice closed-form!

Do they know partial derivatives? I'm not sure. I think we'll be running low on time here anyway, so maybe I'll just talk about the idea of how to get them, set up the equations and then just give the answers.

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

Fitting a Linear Regression Model in R

Recap: Our goal is to predict a value of Y while including an explanatory variable x. We are assuming we have a sample of (x_i, y_i) pairs, $i = 1, ..., n$.

The Simple Linear Regression (SLR) model can be used:

$$\hat{f}(x_i) = \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

where - y_i is our response for the i^{th} observation
- x_i is the value of our explanatory variable for the i^{th} observation
- β_0 is the y intercept
- β_1 is the slope

The best model to use if we consider squared error loss has

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

Data Intro

Plots and such too.

'Fitting' the Model

- Basic *linear model* fits done with `lm()`

```
lm(dist ~ speed, data = cars)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

- Save as object

```
fit <- lm(dist ~ speed, data = cars)
coefficients(fit) #helper function
```

```
## (Intercept)       speed
##  -17.579095    3.932409
```

```
residuals(fit) #helper function
```

```
##          1          2          3          4          5          6          7
##   3.849460  11.849460  -5.947766  12.052234   2.119825  -7.812584  -3.744993
##          8          9         10         11         12         13         14
##   4.255007  12.255007  -8.677401   2.322599 -15.609810  -9.609810  -5.609810
##         15         16         17         18         19         20         21
##  -1.609810  -7.542219   0.457781   0.457781  12.457781 -11.474628  -1.474628
##         22         23         24         25         26         27         28
##  22.525372  42.525372 -21.407036 -15.407036  12.592964 -13.339445  -5.339445
##         29         30         31         32         33         34         35
```

```
## -17.271854  -9.271854   0.728146 -11.204263   2.795737  22.795737  30.795737
##         36         37         38         39         40         41         42
## -21.136672 -11.136672  10.863328 -29.069080 -13.069080  -9.069080  -5.069080
##         43         44         45         46         47         48         49
##   2.930920  -2.933898 -18.866307  -6.798715  15.201285  16.201285  43.201285
##         50
##   4.268876
```

```
summary(fit)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

- Add fit to our scatter plot

```
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point() +
  geom_smooth(method = "lm") +
```

- Now can predict y for a given x

```
predict(fit, newdata = data.frame(speed = c(42, 5)))
```

```
##          1          2
## 147.582073   2.082949
```

Error Assumptions

Not sure about adding this in but it would help connect things with what they are doing in class...

We often assume that we observe our response variable Y as a function of the line plus random errors:

$$Y_i = \beta_0 + \beta_1 x_i + E_i$$

where the errors come from a Normal distribution with mean 0 and variance σ^2 ($E_i \overset{iid}{\sim} N(0, \sigma^2)$)

If we do this and use probability theory/maximum likelihood, we will get the same estimates as above! If the normality assumption is reasonable, we then also gain knowledge of the distribution of our estimators of the intercept and slope ($\hat{\beta}_0$ and $\hat{\beta}_1$).

These allow us to create confidence intervals or conduct hypothesis tests. However, these assumptions aren't needed to simply do prediction (but are useful to get error bounds on our prediction

- Get SE for prediction

```
predict(fit, newdata = data.frame(speed = c(42, 5)), se.fit = TRUE)
```

```
## $fit
##          1          2
## 147.582073   2.082949
##
## $se.fit
##         1         2
## 11.264612  4.837825
##
## $df
## [1] 48
##
## $residual.scale
## [1] 15.37959
```

- Get confidence interval for mean response

```
predict(fit, newdata = data.frame(speed = c(42, 5)),
        se.fit = TRUE, interval = "confidence")
```

```
## $fit
##          fit       lwr       upr
## 1 147.582073 124.93305 170.23109
## 2   2.082949  -7.64415  11.81005
##
## $se.fit
##         1         2
## 11.264612  4.837825
##
## $df
## [1] 48
##
## $residual.scale
## [1] 15.37959
```

- Get prediction interval for new response

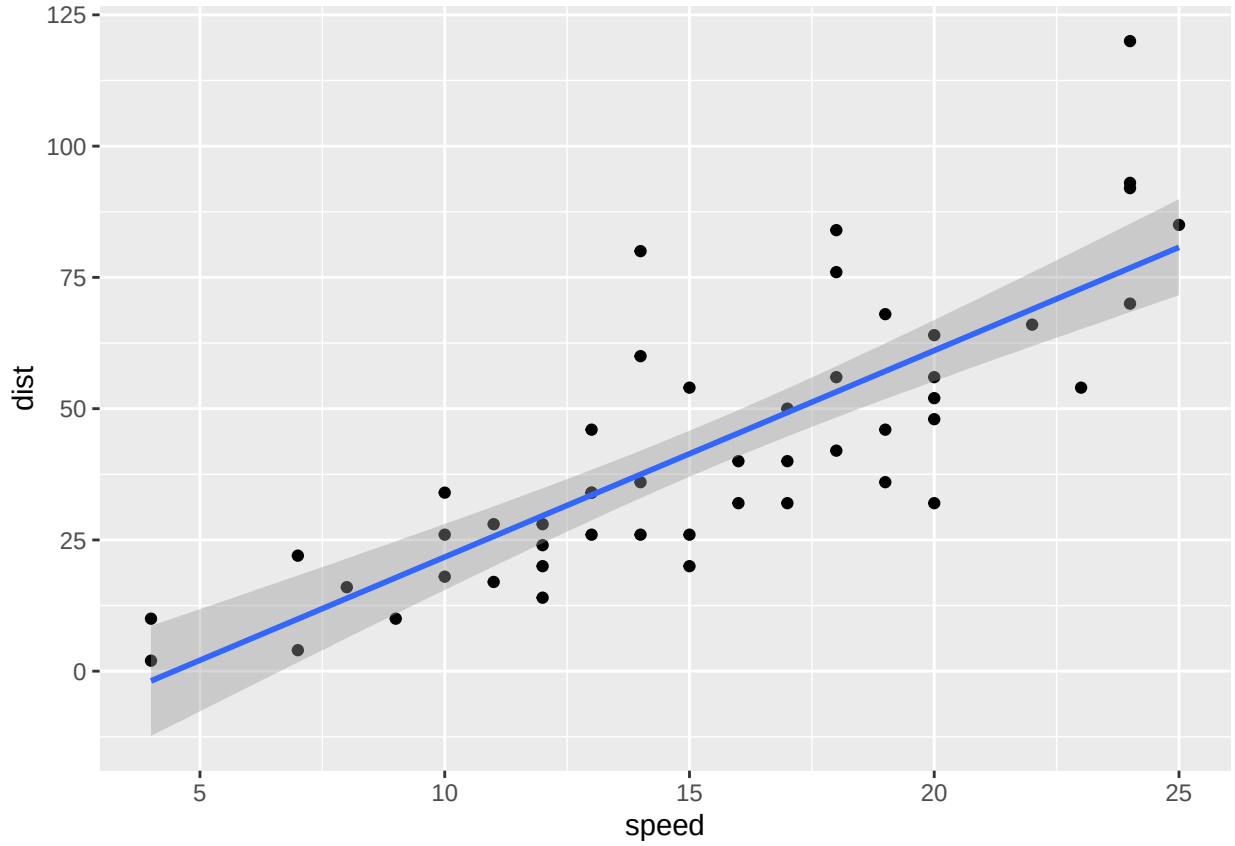
```
predict(fit, newdata = data.frame(speed = c(42, 5)),
        se.fit = TRUE, interval = "prediction")
```

```
## $fit
##          fit       lwr       upr
## 1 147.582073 109.25201 185.91213
## 2   2.082949 -30.33359  34.49948
##
## $se.fit
##         1         2
## 11.264612  4.837825
##
## $df
## [1] 48
##
## $residual.scale
## [1] 15.37959
```

- Confidence interval is automatically added to ggplot scatter plots with a `geom_smooth()` (called `se` unfortunately...)

```
library(ggplot2)
ggplot(cars, aes(x = speed, y = dist)) + geom_point() +
  geom_smooth(method = "lm", se = TRUE) #TRUE by default, 95% CI
```

```
## `geom_smooth()` using formula 'y ~ x'
```



- Add PI easily with packages (install `ciTools` package)

```
cars <- cars %>% ciTools::add_pi(fit, names = c("lower", "upper"))

ggplot(cars, aes(x = speed, y = dist)) +
  geom_point() +
    geom_smooth(method = "lm", fill = "Blue") +
    geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.3, fill = "Red") +
    ggtitle("Scatter Plot with 95% PI & 95% CI")
```

- Diagnostic plots easily found using `plot()`

```
plot(fit)
```

Evaluating Model Accuracy

Kaggle Competition

Data and they try

kNN as an alternative

kNN on Kaggle