Plus/minus what? Let's talk about uncertainty

Sigrid Keydana, Trivadis
2017/22/11

About me & my employer

Trivadis

- DACH-based IT consulting and service company, from traditional technologies to big data/machine learning/data science

My background

- from psychology/statistics via software development and database engineering to data science and ML/DL

My passion

- machine learning and deep learning
- data science and (Bayesian) statistics
- explanation/understanding over prediction accuracy

Where to find me

- blog: **http://recurrentnull.wordpress.com**
- twitter: @zkajdan

In this world nothing can be said to be certain, except death and taxes

Welcome to everything else!

- How many super cool iphone lookalikes will we sell next quarter?
- When should we invest in more powerful servers?
- How many boxes of super healthy energy bars should we keep in stock?
- How long does it take to run this batch job?
- How much time do you need for that report?

Our job: sports forecasting

Our task today: forecast men's 400m Olympic winning times

It's 2000, just before the Olympics. This is the data we have:

Expected time in 2000: 42.3 seconds? Or 42.1?

Linear regression says 42.33.

Whatever we say, it's pretty likely to be wrong…

How do we deal with this?

Let's better not commit to a point estimate…

Prediction intervals to the rescue!

Prediction intervals - linear regression

Wait: prediction intervals? Wasn't that called "confidence intervals?"

Let's take the example of linear regression.

```
(fit <- lm(seconds ~ year, male400_1996))
```

```
Call:
lm(formula = seconds ~ year, data = male400_1996)

Coefficients:
(Intercept)         year
  207.46609     -0.08257
```

Here's the point prediction:

```
# this would yield the same result
# fit$coefficients[1] + fit$coefficients[2] * 2000
fit %>% predict(newdata = data.frame(year = c(2000)))
```

```
       1
42.33243
```

Let's try getting those (confidence? prediction?) intervals!

Confidence intervals:

```
fit %>% predict(newdata = data.frame(year = c(2000)), interval = "confidence")
```

```
       fit     lwr      upr
1 42.33243 41.2646 43.40026
```

Prediction intervals:

```
fit %>% predict(newdata = data.frame(year = c(2000)), interval = "prediction")
```

```
       fit      lwr      upr
1 42.33243 39.48191 45.18295
```

Quite a difference! So which one do we take?

One step back: sampling variation and standard errors

- In single-variable linear regression we estimate an *intercept* and a *slope*

$$\beta_0 \quad y \quad \beta_1 x \qquad \beta_1 \quad cor\ y, x \ \frac{sd\ y}{sd\ x}$$

  - … that together make up the equation of a line

$$y_i \quad \beta_0 \quad \beta_1 x_i \quad \epsilon_i$$

- These estimates vary depending on the *sample* they are estimated from
- The statistical method gives us *standard errors* for these estimates

$\sigma_{\beta_0} \quad \sigma^2 \ \frac{1}{n} \qquad \frac{x^2}{\sum_{i\ 1}^{n} x_i \quad x\ ^2} \qquad \sigma_{\beta_1} \quad \sigma^2 \qquad \frac{n}{\sum_{i\ 1} x_i \quad x\ ^2}$ with

$$\sigma^2 \quad \frac{1}{n\ 2} \sum_{i\ 1}^{n} e_i^2$$

From standard errors, we can construct confidence intervals

- for the parameters
- for the line overall

Let's do this manually for the parameters. A 95% confidence interval for the intercept would look like this:

```
intercept_est <- summary(fit)$coefficients[1,1]
intercept_se <- summary(fit)$coefficients[1,2]
(conf_interval <- intercept_est + c(-1, 1) * qt(.975, df = fit$df) * intercept_se)
```

```
[1] 174.3053 240.6269
```

Same procedure for the slope:

```
slope_est <- summary(fit)$coefficients[2,1]
slope_se <- summary(fit)$coefficients[2,2]
(conf_interval <- slope_est + c(-1, 1) * qt(.975, df = fit$df) * slope_se)
```

```
[1] -0.09960579 -0.06552787
```

Which means we can say

"with 95% confidence, we estimate that having 4 years pass results in a decrease in the men's 400m Olympic winning times of 0.07 to 0.1 seconds"

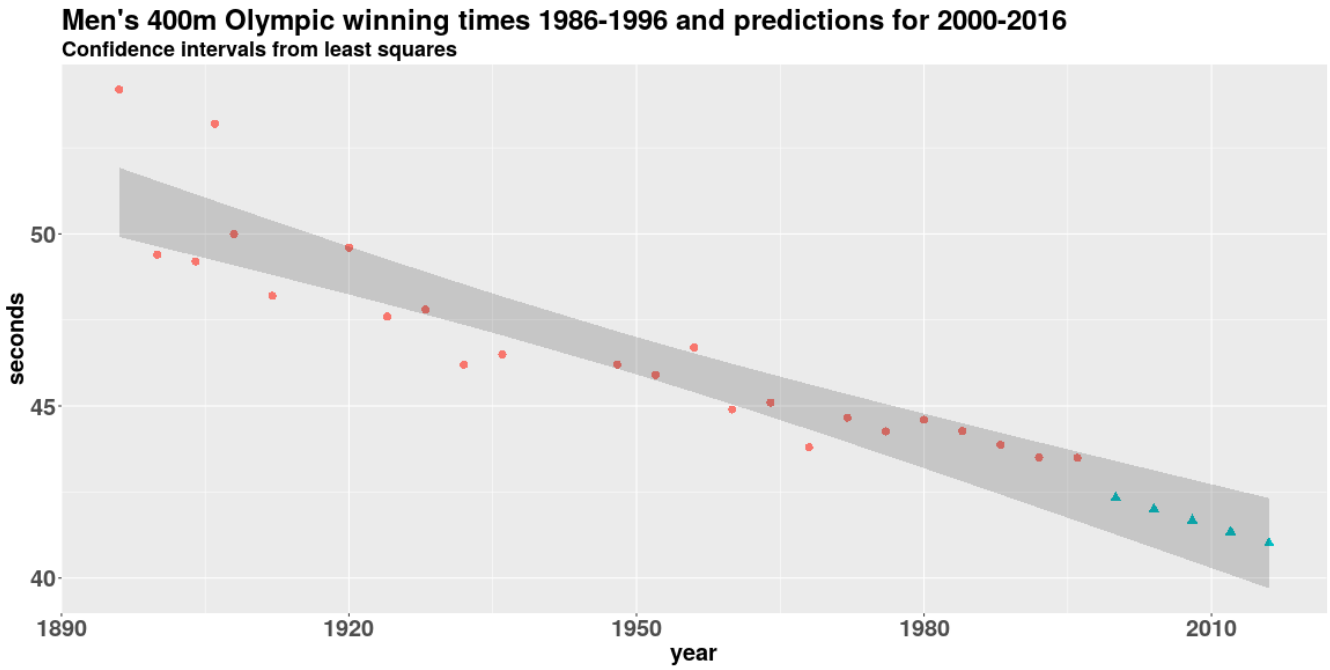
… which reflects our uncertainty about the slope, not the points on the line.

A confidence interval for the regression line

We need the standard error for a point x_0 on the regression line:

$$\sigma \quad \frac{1}{n} \quad \frac{x_0 \quad x}{\sum_{i \quad 1}^{n} x_i \quad x}^{2}$$

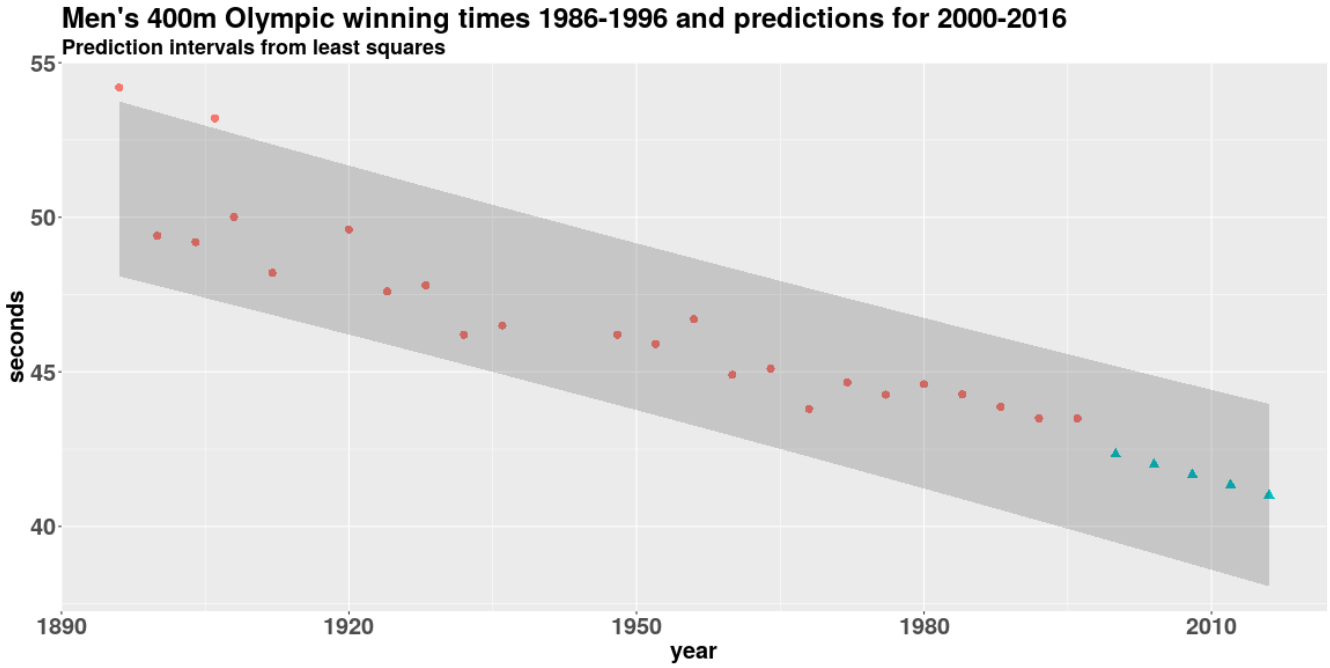
This reflects the amount of uncertainty due to our estimates being based on *sample variation*.
Uncertainty is smallest near the mean of the predictor (x).



But... there's an additional source of uncertainty!

For sure the predictor x (the year we're in) cannot be held 100% responsible for the outcome y (the 400m winning time).

The standard error for an actual *prediction* is: $\sigma \quad 1 \quad \frac{1}{n} \quad \frac{x_0 \quad x^{\,2}}{\sum_{i \quad 1}^{n} x_i \quad x^{\,2}}$



Fine, but... that's all specific to linear regression...

- What if we were using, say, ARIMA for forecasting that time series?
- What if we used some custom method that does not come complete with standard errors / confidence intervals and all?

Prediction intervals - ARIMA

Let's try ARIMA on the 400m winning times!

```
arima_fit <- auto.arima(male400_1996$seconds)
arima_fit$coef
```

```
      ma1     drift 
-0.7998509 -0.3807447 
```

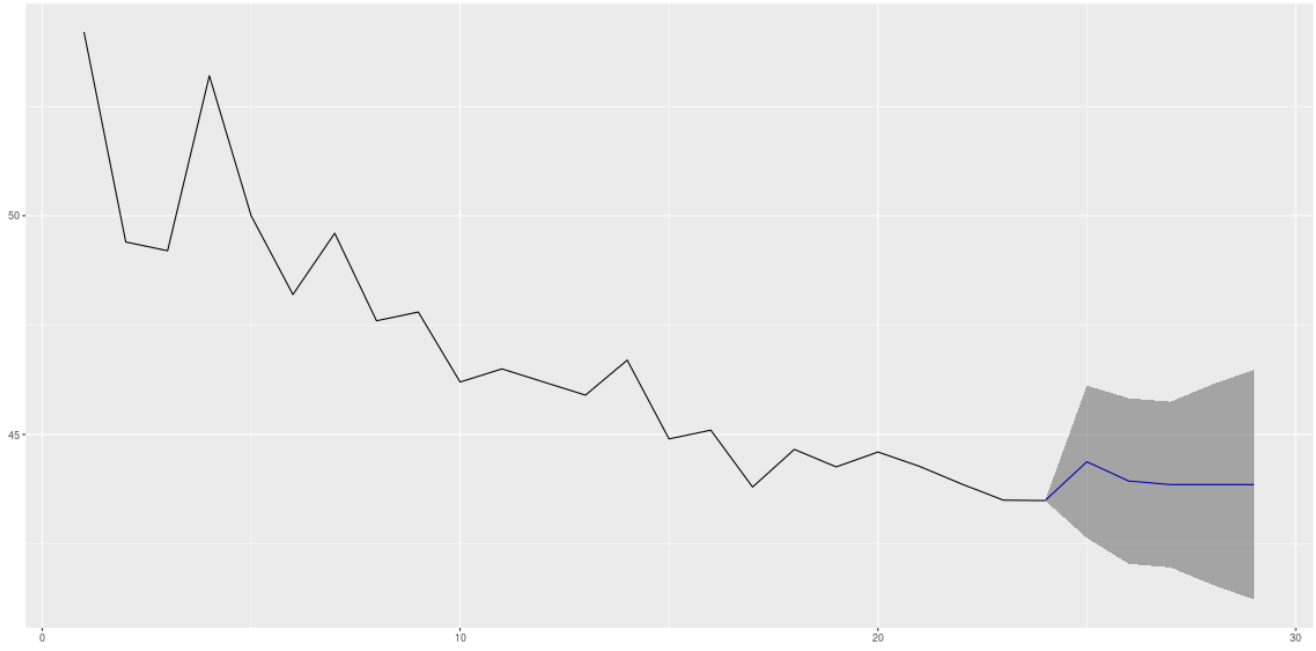
We get standard errors for the parameter estimates:

```
sqrt(diag(vcov(arima_fit)))
```

```
      ma1       ma2       ma3 
0.2524413 0.3333542 0.3169874 
```

ARIMA prediction intervals

… which means we can get prediction intervals here, too:



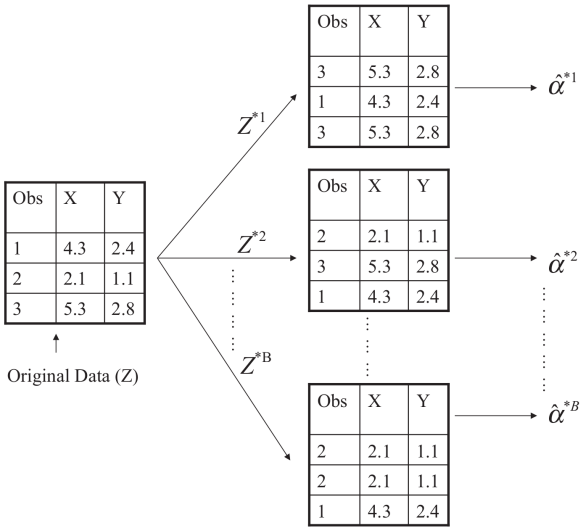
Fine. What if we're not automagically given those prediction intervals?

We'll probably need to pull ourselves up by our own bootstraps…

Generic solution: the bootstrap

The bootstrap

- Ideally, we'd compute a parameter's standard error from many repeated samples
- Unfortunately, most of the time we just have a single sample
- Idea: create synthetical samples from the original one, using *sampling with replacement*



Source: **James et al., Introduction to Statistical Learning**

So it all depends on the specifics of the method we're using...

Wouldn't it be nice if there was a unified, intuitive approach?

Well … there is.

Uncertainty for free: The Bayesian approach

Let's do it the Bayesian way!

In Bayesian statistics:

- the *data* is *given*, and the *parameters* are *random* (and thus have distributions, too!)
- as new data comes in, we update our expectations - according to famous *Bayes theorem*

$$P\ A\ B \qquad \frac{P\ B\ A\ *P\ A}{P\ B}$$

- uncertainty estimates are "for free", as we look at the complete posterior distribution

Bayes - in practice - using R

R packages for Bayesian modeling (using the ***Stan*** backend for MCMC sampling):

- *rethinking* (by R. McElreath, author of awesome *Statistical Rethinking*) - we'll use this one soon
- rstan (uses Stan's C-like DSL)
- rstanarm, brms (higher level interfaces in the style of R's usual model fitting syntax)

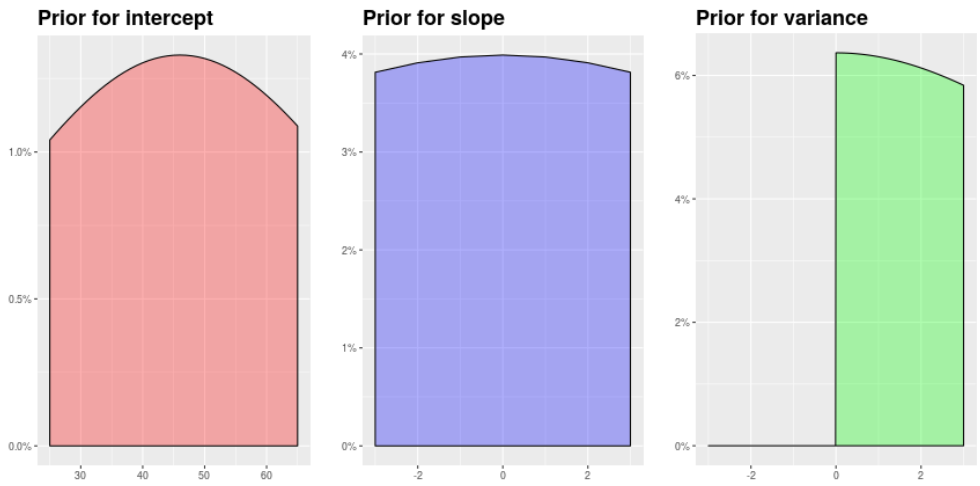
No Bayes without priors...

- Common practice: use uniform priors (feigning total ignorance)
- But often, we have at least *some* information!

Choosing priors for the men's 400m

What do we know?

- mean is somewhere around 46 => have intercept centered at 46, with high variance
- want to conservatively estimate slope: center at 0, but use high variance
- noise variance can only be >= 0: use half-cauchy centered at 0



Model specification

```
require(rethinking)

model <- map2stan(
  alist(
    seconds ~ dnorm(mu, sigma),
    mu <- a + b*year,
    a ~ dnorm(46, 30),
    b ~ dnorm(0,10),
    sigma ~ dcauchy(0, 10)
  ),
  data = male400_1996,
  iter = 6000,
  chains = 4,
  verbose = FALSE
)
```

Model results

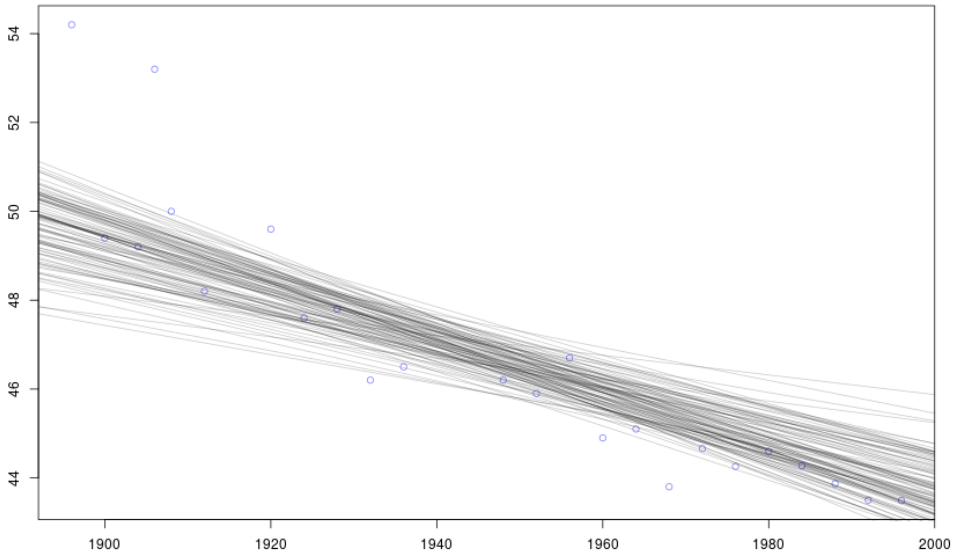
```
precis(model)
```

```
        Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
a     156.75  21.70     123.19     190.36  1695    1
b      -0.06   0.01      -0.07      -0.04  1693    1
sigma   1.62   0.35       1.06       2.07  1076    1
```

So, what about the regression line?

With all the sampling that's been going on, now we don't just have 1 line, but many!

We extract parameter values from the samples and construct one line per sample:

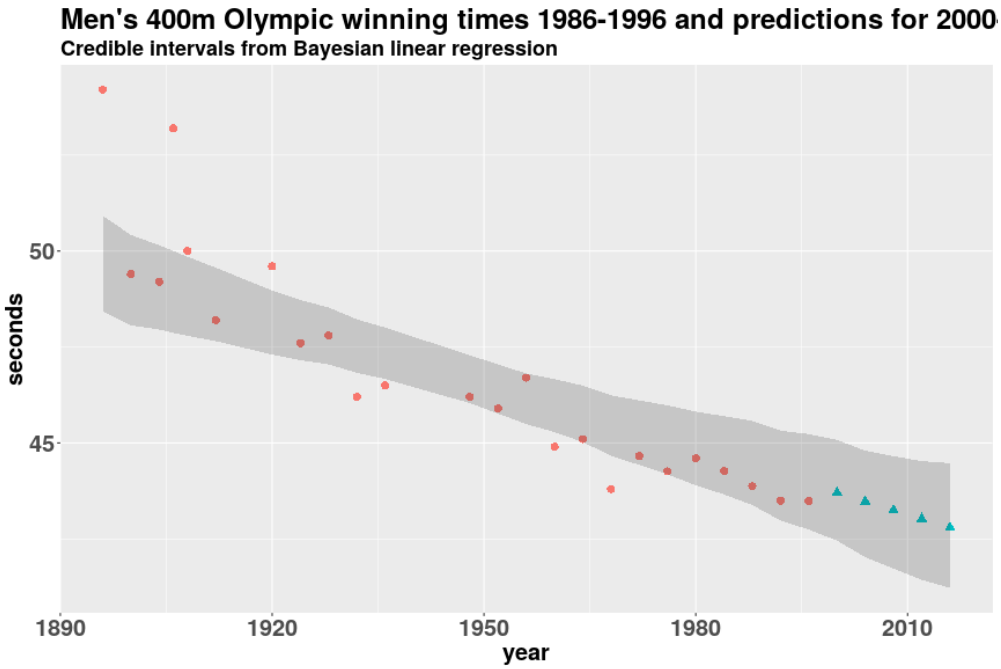


We can immediately *see* the uncertainty in our model!

Posterior intervals for the regression line

In the Bayesian framework, the equivalent of a confidence interval is a *credible interval*. Equivalent? Well… except

- credible intervals really have an intuitive interpretation
- credible intervals really contain the highest density values

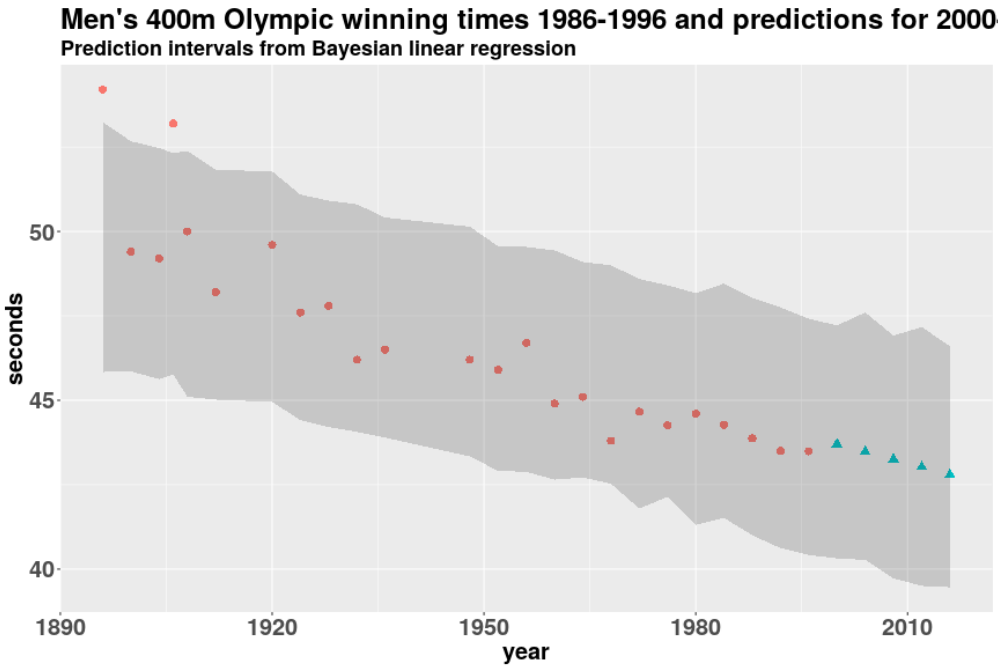


Credible intervals are easily constructed using existing samples from the parameters' posterior.

Prediction intervals

Of course, here again we're not really interested in predicting averages, but individual examples.

We want *prediction intervals*:



Prediction intervals are computed from the complete *posterior predictive* distribution.

So that's uncertainty intervals...

… didn't you say we'd look at *complete distributions* for our predictions?

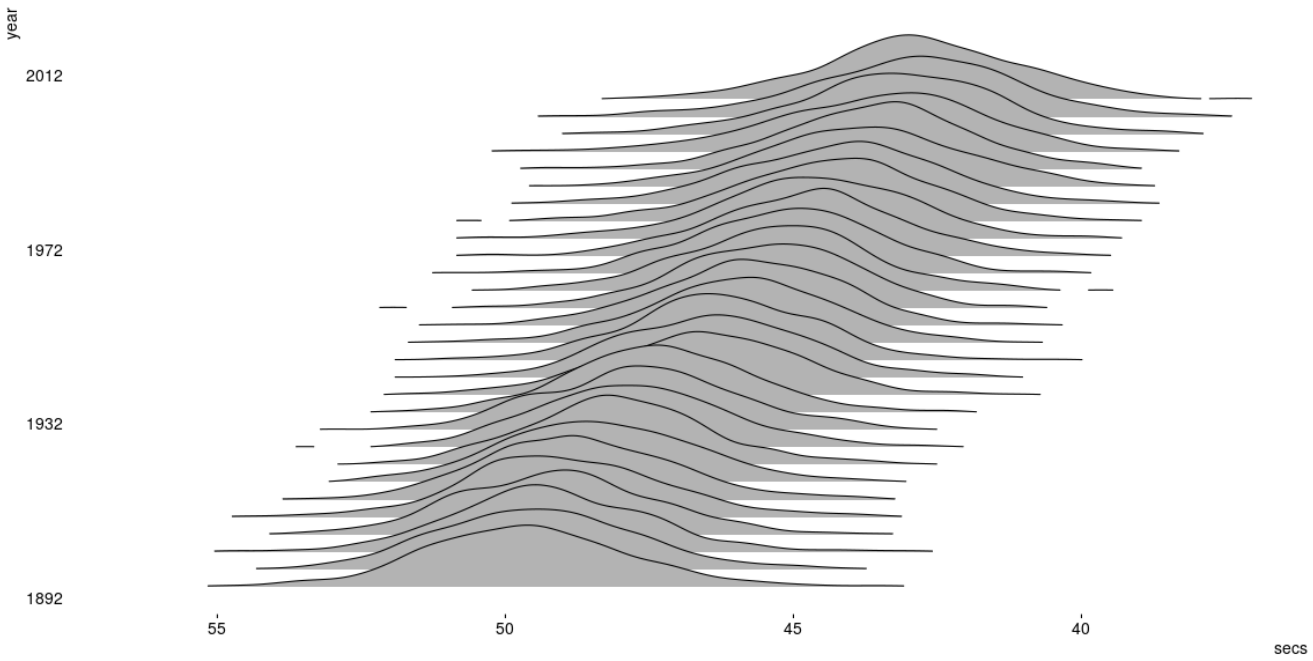
Yes. That's the *posterior predictive* distribution:

$$p\ y\ \ X, y, x, \theta \qquad\qquad p\ y\ \ x, \theta\ \ p\ \theta\ \ X, y\ \mathrm{d}\theta$$

The posterior predictive is a weighted average of predictions, computed over all possible parameter values.

Posterior predictive - the whole picture

We actually have a distribution of predictions for every point in time.



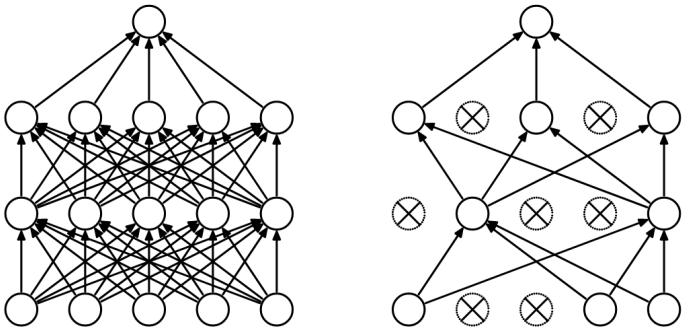
Fine. So Bayes solves it all...

… what if I'm using deep learning?

Uncertainty in deep neural networks

Uncertainty in deep learning

- Normally, in neural networks, the outputs are point predictions (possibly in the form of class probabilities)
- Looking at NN architecture, it is not self-evident how to extract confidence/prediction intervals from a network
- We could always apply the bootstrap approach, or even simple ensembling, to at least get a feel for the variability of the net's estimates
- Several other approaches have been suggested, among them the use of *dropout* to calculate uncertainty:



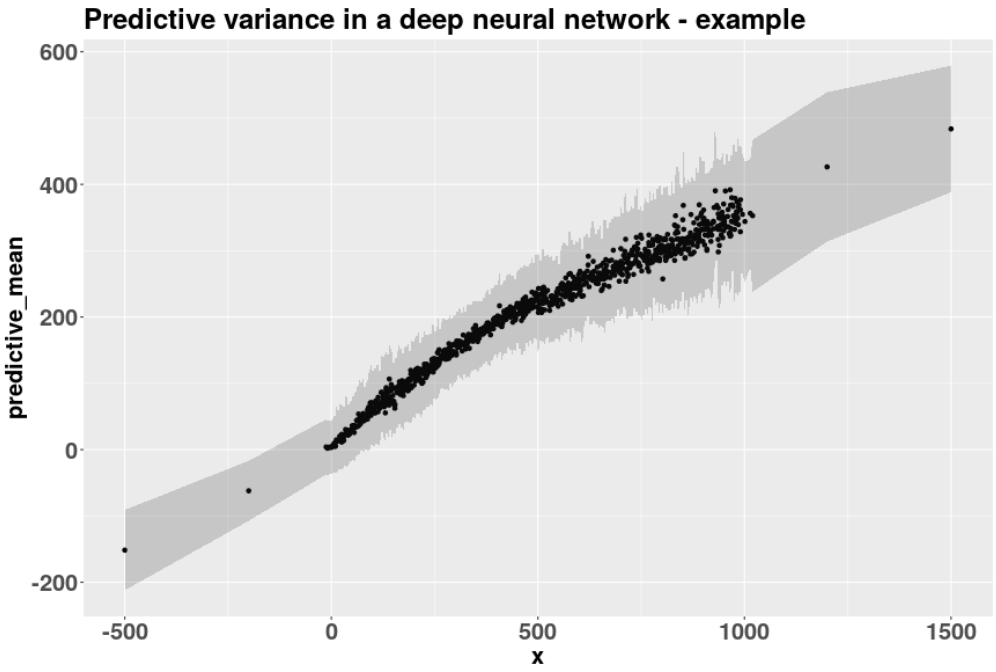
(a) Standard Neural Net (b) After applying dropout.

Source: a href="http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf">Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dropout networks as variational inference in Gaussian processes

"… We'll see that what we did above, averaging forward passes through the network, is equivalent to Monte Carlo integration over a Gaussian process posterior approximation."

Yarin Gal, ***What my deep model doesn't know***



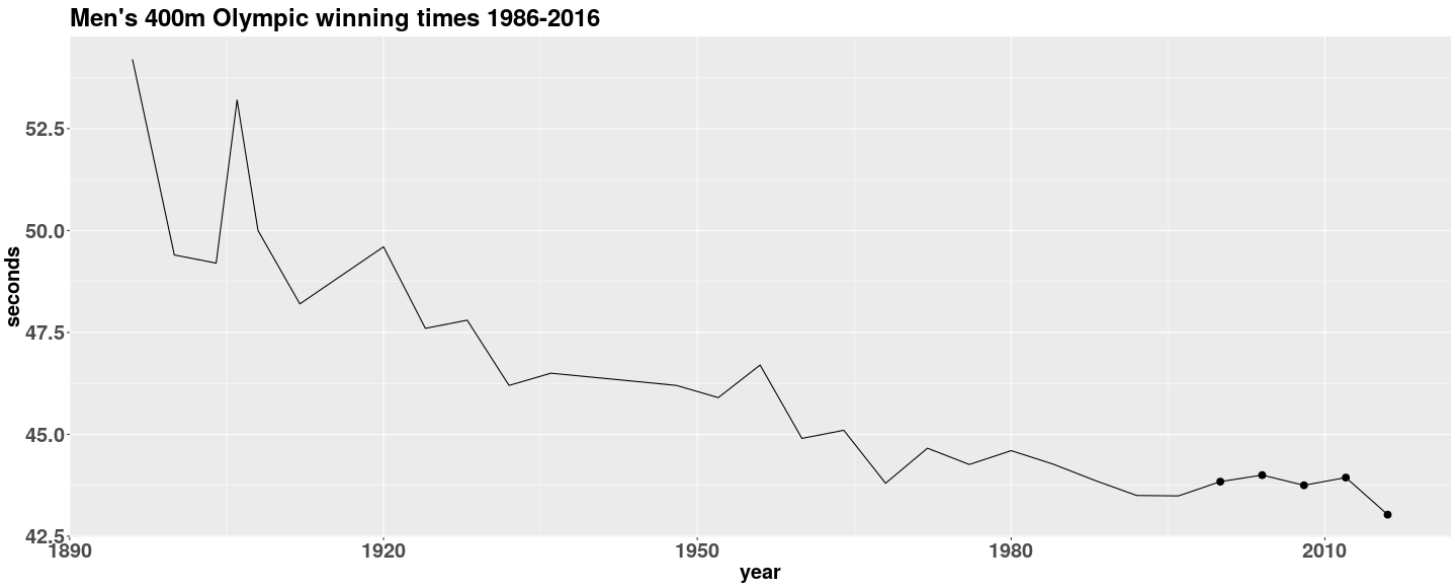
Now ... you never really showed us ...

… *how* wrong we'd have been with our point prediction…

In reality it's 2017 so we're so much wiser now…

So what *was* the men's 400m Olympic winning time in 2000?

The truth about the 400m - up until 2016 at least



Conclusion

- If you take away one thing from this talk, it's the *importance* of reporting uncertainty
- If your tool/your method doesn't seem to yield such information, dig deeper
  - if it's some vendor's licensed product, don't let them get away with it ;-)
  - if it's a method you've developed yourself, think how you can incorporate it
  - if you're using a tool (language) like R, check out it's fantastic packages, which provide *everything and more*
- If you have questions, don't hesitate to ask :-)

Questions?

Thank you!!