Plus/minus what? Let’s talk about uncertainty

Sigrid Keydana
Trivadis
München

Keywords
Data Science, Statistics, Bayesian Statistics, Machine Learning

Introduction

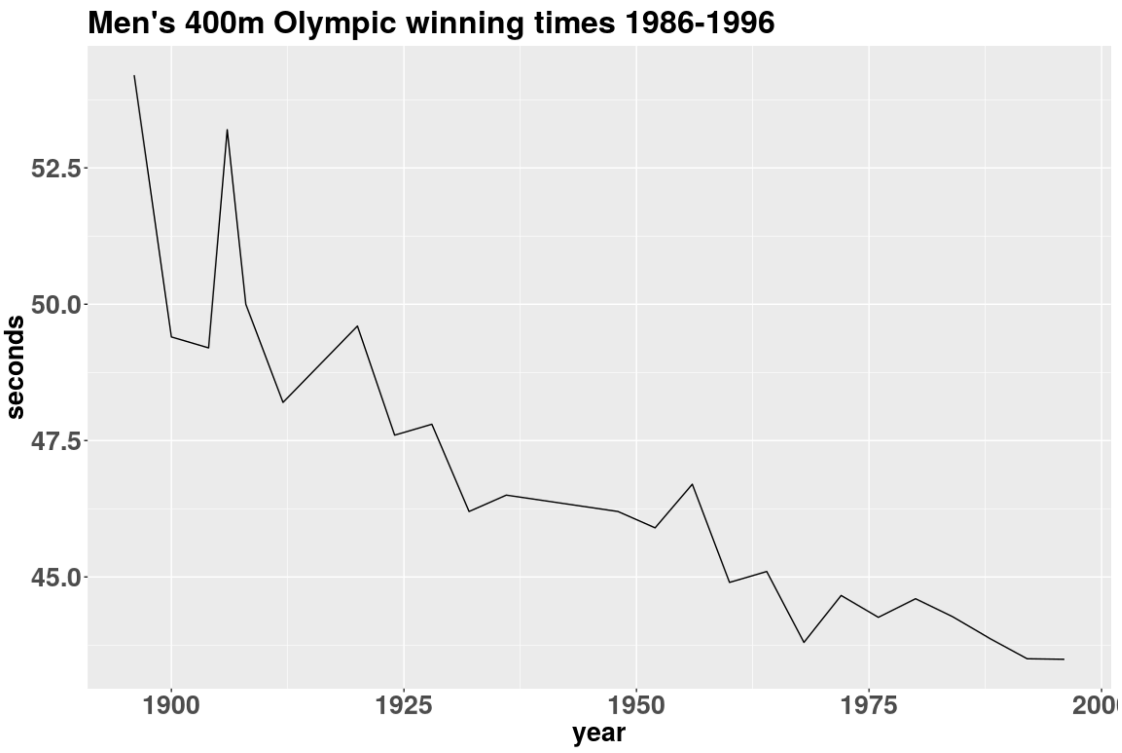
Forecasting is something we do every day. Will it rain? Will I finish this task in time? Will <fill in your favorite sports team here> win the next <fill in corresponding sports> match?

In our work life, it's often numbers we have to forecast: How many products will we sell next quarter? When should we buy more storage for our big data system? At what intervals do we have to get new supply?

Whenever we reply with a specific number, we're pretty likely to be wrong. What can we do? We need to make sure we communicate *intervals*, not point estimates. That's what this presentation is about.

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

It's 2000, just before the Olympics. We're tasked with forecasting the men's 400m winning times. This is the data we have:



So what should we say? 42.3 seconds? 42.1? Let's try linear regression:

```
fit <- lm(seconds ~ year, male400_1996)
fit %>% predict(newdata = data.frame(year = c(2000)))

##        1
## 42.33243
```

Whatever the authority of linear regression, it does not feel safe to just say "42.3". Fortunately, from linear regression we can also get something else: prediction intervals.

Wait: prediction intervals? Wasn't that called "confidence intervals?" Well, there's both. Let's take a look:

First, confidence intervals. We see that the 95% confidence interval ranges from 41.3 to 43.4 seconds.

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

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

And here are prediction intervals. They are a lot wider: from 39.4 to 45.2.

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

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

So which one should we report? Let's first find out what they actually mean.

Sampling variation and standard errors

In single-variable linear regression we estimate an *intercept*

$$\overset{¿}{\beta}_0 = \overline{y} - \overset{¿}{\beta}_1 \overline{x}$$

and a *slope*

$$\overset{¿}{\beta}_1 = cor(y, x) \frac{sd(y)}{sd(x)}$$

that together make up the equation of a line:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

These estimates vary depending on the *sample* they are estimated from. Linear regression gives us *standard errors* for these estimates:

$$¿ x_i - \overline{x} ¿^2$$

$$(¿¿)$$

$$\sum_{i=1}^{n} ¿$$

$$\frac{1}{n} + \frac{\overline{x}^2}{¿}$$

$$\sigma_{\overset{¿}{\beta}_0} = \overset{¿}{\sigma}^2 ¿$$

$$¿\, x_i - \bar{x}\, ¿^2$$
$$¿$$
$$\sigma_{\overset{¿}{\beta}_1} = \overset{¿}{\sigma}^2 / \sum_{i=1}^{n} ¿$$

with $\overset{¿}{\sigma}^2 = \frac{1}{n-2} \sum_{i=1}^{n} e_i^2$

Confidence intervals in linear regression

From the parameters' standard errors, we can construct confidence intervals for them. We can even do this manually. 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
```

And here is a confidence interval 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
```

With these confidence intervals for the parameters, we can say something like

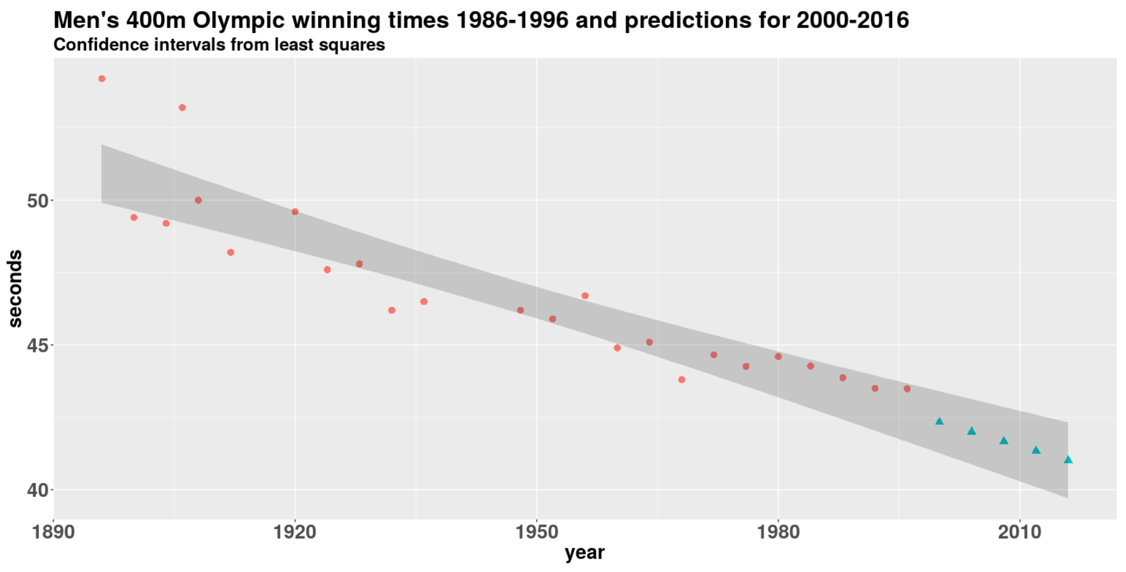
with 95% confidence, we estimate that having 4 years pass results in a decrease in the men's 400m Olympic winning times of 0.07 to 0.1 seconds

... which reflects our uncertainty about the slope, not the points on the line. We need something more.

So what we need is the standard error for a point x_0 on the regression line:

$$x_0 - \bar{x}\, ¿^2$$
$$¿$$
$$¿\, x_i - \bar{x}\, ¿^2$$
$$¿$$
$$¿$$
$$\sum_{i=1}^{n} ¿$$
$$¿$$
$$\frac{1}{n} + ¿$$
$$\overset{¿}{\sigma} \sqrt{¿}$$

This reflects the amount of uncertainty due to our estimates being based on *sample variation*. Uncertainty is smallest near the mean of the predictor $\bar{x}$.



But uncertainty based on sample variation is not everything. There's an additional source of uncertainty!

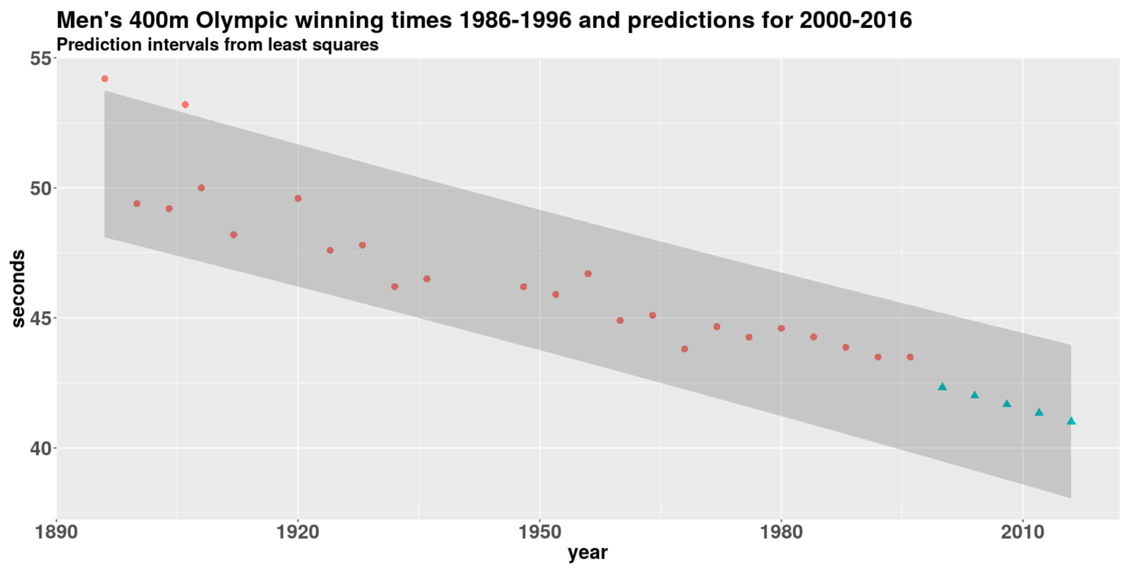
For sure the predictor x (the year we're in) cannot be held 100% responsible for the outcome y (the 400m winning time).

Prediction intervals in linear regression

The standard error for an actual *prediction* is:

$$x_0 - \bar{x}¿^2$$
$$¿$$
$$¿\, x_i - \bar{x}¿^2$$
$$¿$$
$$¿$$
$$\sum_{i=1}^{n} ¿$$
$$¿$$
$$1+\frac{1}{n}+¿$$
$$\overset{¿}{\sigma}\sqrt{¿}$$

... which leads to the following *prediction intervals*



These prediction intervals are what we report with our forecasts.

You might say that's all fine, but these formulae are specific to linear regression. For time series, we might be using ARIMA though, for example. Or say we use a method that does not conveniently come with prediction/confidence intervals. What do we do? Let's look at ARIMA first.

Prediction intervals for ARIMA

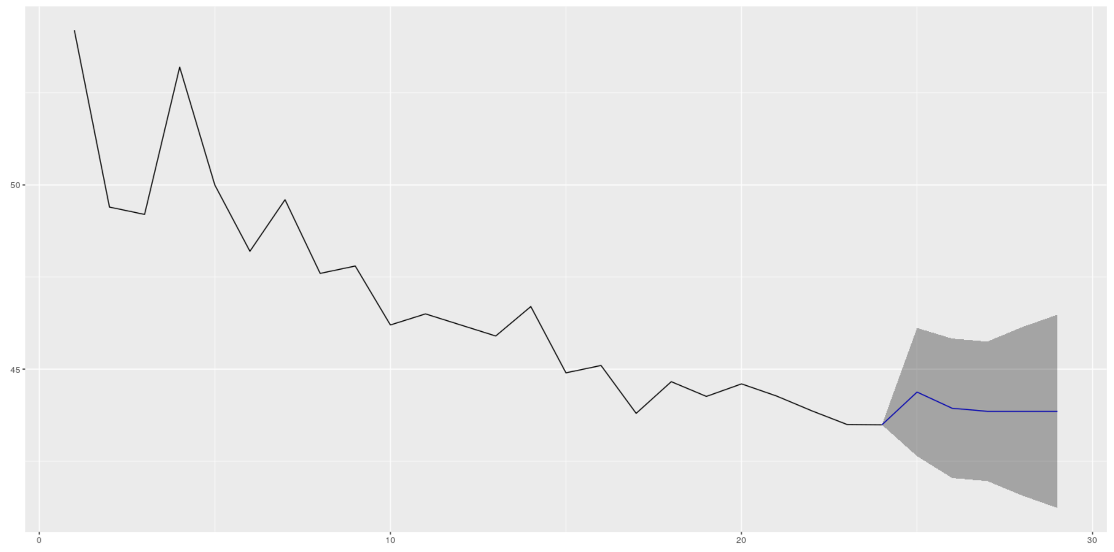
Here we apply ARIMA to the men's 400m data.

```
arima_fit <- auto.arima(male400_1996$seconds, allowdrift = FALSE)
arima_fit$coef

##        ma1        ma2        ma3
## -0.5744779 -0.3777209  0.6914891
```

It's less convenient than with linear regression, but we can extract standard errors for the parameter estimates:

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

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

The existence of standard errors means prediction intervals are available too:



Fine. So what if prediction intervals are not available? A generic method we can always use is 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.

The idea of the bootstrap is to use the one sample we have and treat it as a population. We repeatedly draw samples from it, with replacement, and compute the parameter we're interested in. Then the standard error is estimated from the variability of those repeated estimations.

This approach is generally applicable whatever the parameter estimated.

If you're confused by now, that's understandable. So many different ways... Wouldn't it be nice if there were one unified framework within which to compute uncertainty?

Well - there is. In the Bayesian approach, there's a common philosophy underlying all models.

Going Bayesian

In Bayesian statistics, the *data* is *given*, and the *parameters* are *random* - as opposed to the often used frequentist paradigm. The parameters being random means they have *distributions*, as opposed to being point estimates.

We start with our prior belief about the parameters. Then, as new data comes in, we update our expectations - according to the famous *Bayes theorem*

$$P(A \vee B) = \frac{P(B \vee A) * P(A)}{P(B)}$$

By updating our prior belief with the likelihood of the data observed, we arrive at posterior estimates, which must be updated again if and when new data comes in.

As we're always working with complete distributions, in the Bayesian framework we get uncertainty estimates "for free".

The Bayesian approach itself is old, but it only recently gained popularity as modern hardware and software allowed for ways to approximate the evidence (the term in the formula's denominator).

There is an increasing number of ways to do Bayesian modeling in R. Most of them use the *Stan* backend for Markov Chain Monte Carlo sampling (Hamiltonian Monte Carlo, mostly).

Here, we're using R. McElreath's *rethinking* package that allows for model specification in the usual way.

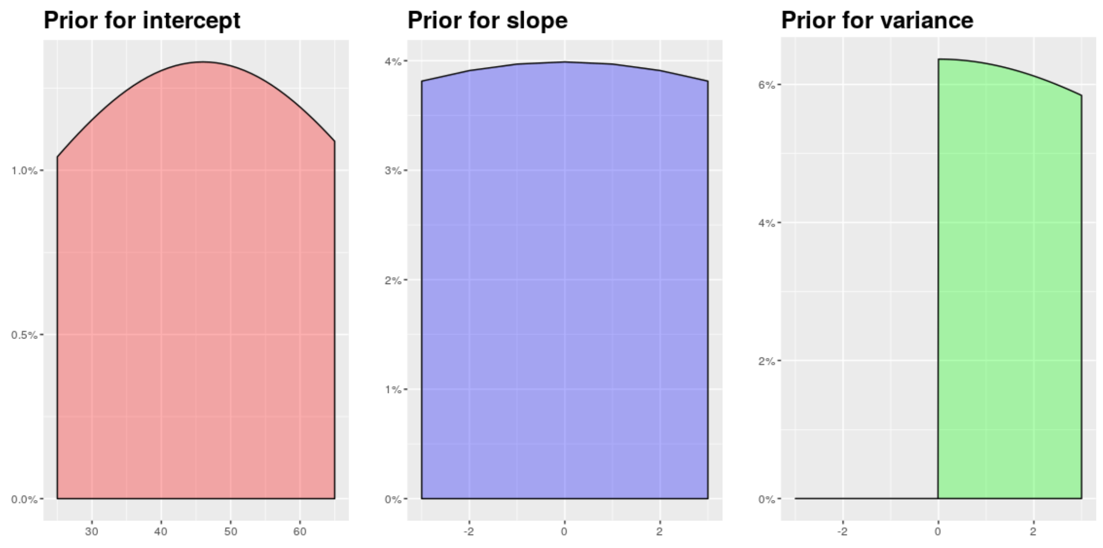
Before looking at the actual model, let's think about priors for our parameters: the intercept and the slope.

Sometimes in Bayesian inference, uniform priors are used, thus feigning total ignorance. But often, we have at least *some* information! In our case, we know that

1. the mean is somewhere around 46. Thus, we'll have the intercept centered at 46, but we choose a high variance for its prior.
2. the noise variance can only be >= 0, not negative. We'll use a half-cauchy centered at 0 as its prior.

Furthermore, we want to conservatively estimate the slope, while still giving the data the freedom to make us adjust our prior belief. Thus, we center the slope's prior distribution at 0, but have it vary around its mean a lot.

Here is a graphical display of the chosen priors:



Now we're ready to specify the model.

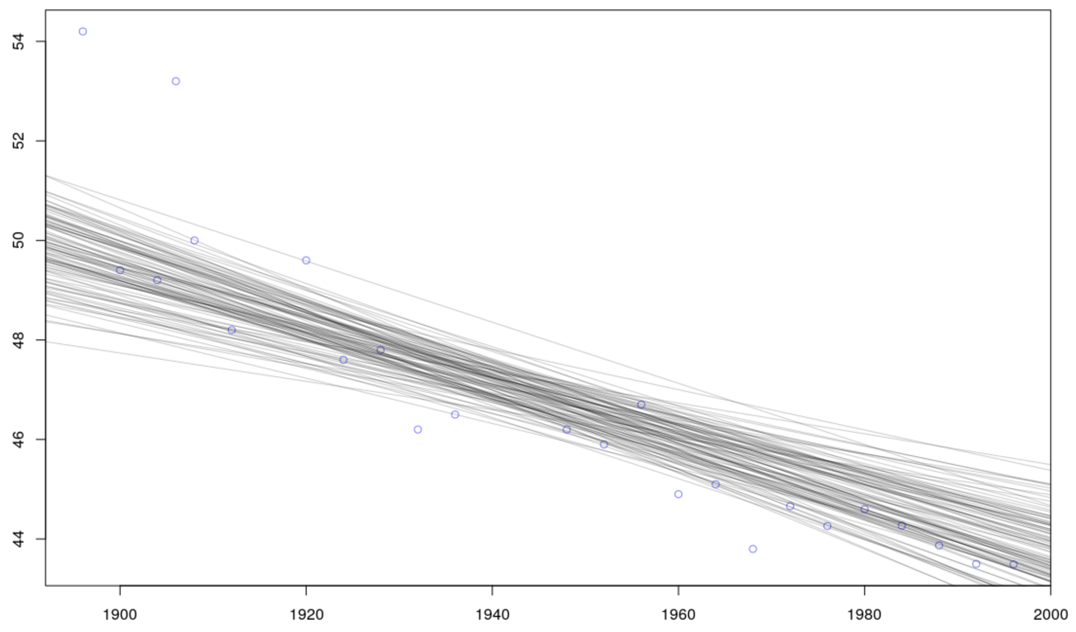
```
require(rethinking)

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
)
```

After some sampling time, this is the output we get - intercept, slope and noise variance from Bayesian linear regression.

```
precis(model)

##         Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
## a     156.52  22.17     123.11     192.78  1854 1.01
## b      -0.06   0.01      -0.08      -0.04  1853 1.01
## sigma   1.62   0.34       1.13       2.14  2165 1.00
```

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!

rethinking saves for us the list of sampled parameters. We can use them directly and draw one line per sample (we're drawing just 100 here):

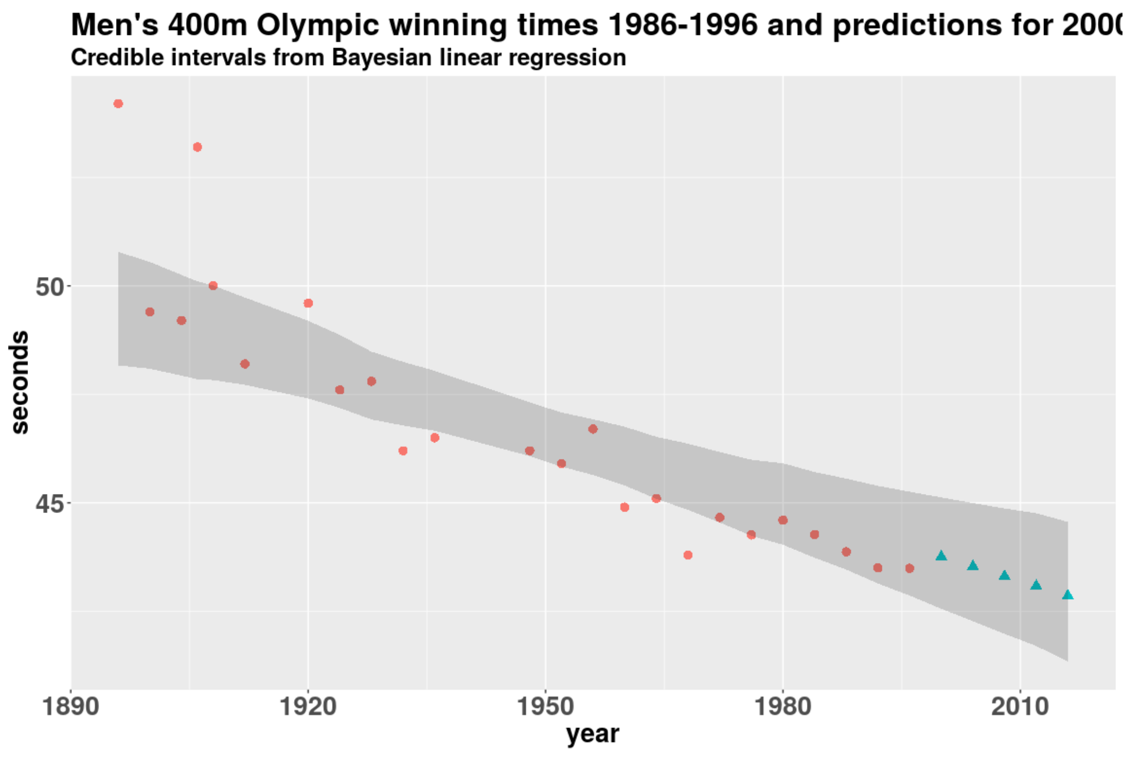


We can immediately *see* the uncertainty in our model!

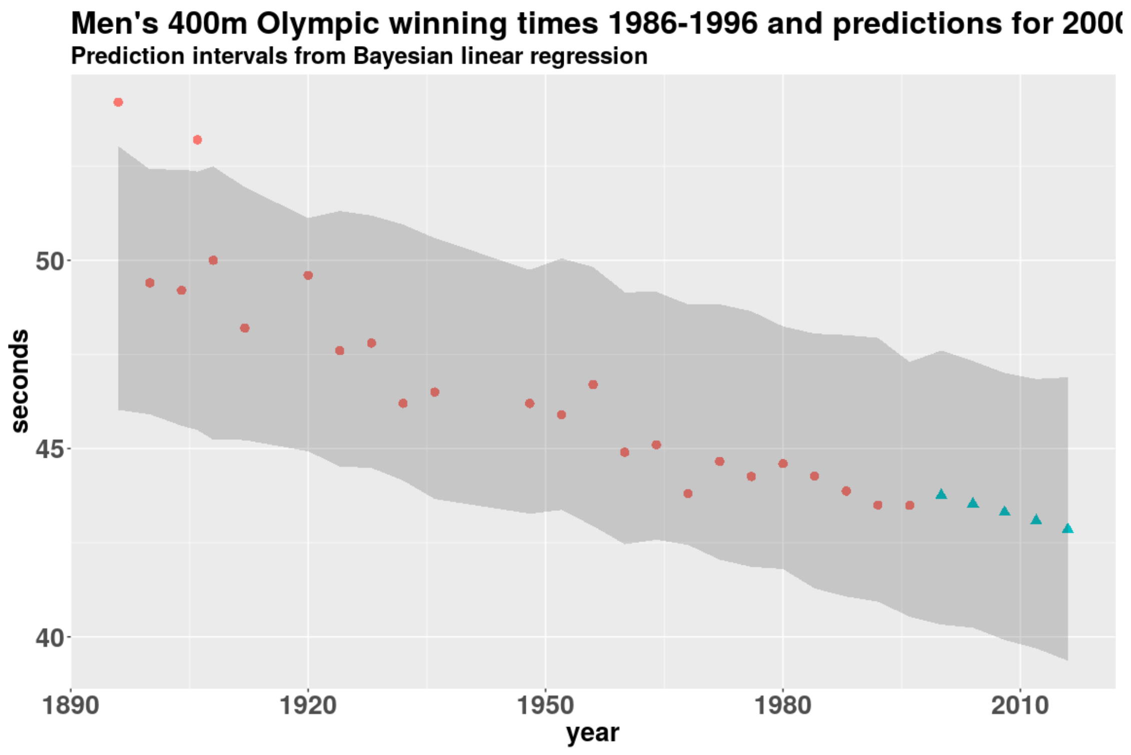
Bayesian credible intervals and prediction intervals

In the Bayesian framework, the equivalent of a confidence interval is a *credible interval*. Equivalent? Well... except that credible intervals really have an intuitive interpretation, and credible intervals really contain the highest density values as they don't have to be equi-spaced around the point estimate.

Here we see credible intervals for the regression line. Credible intervals are easily constructed using existing samples from the parameters' posterior.



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.

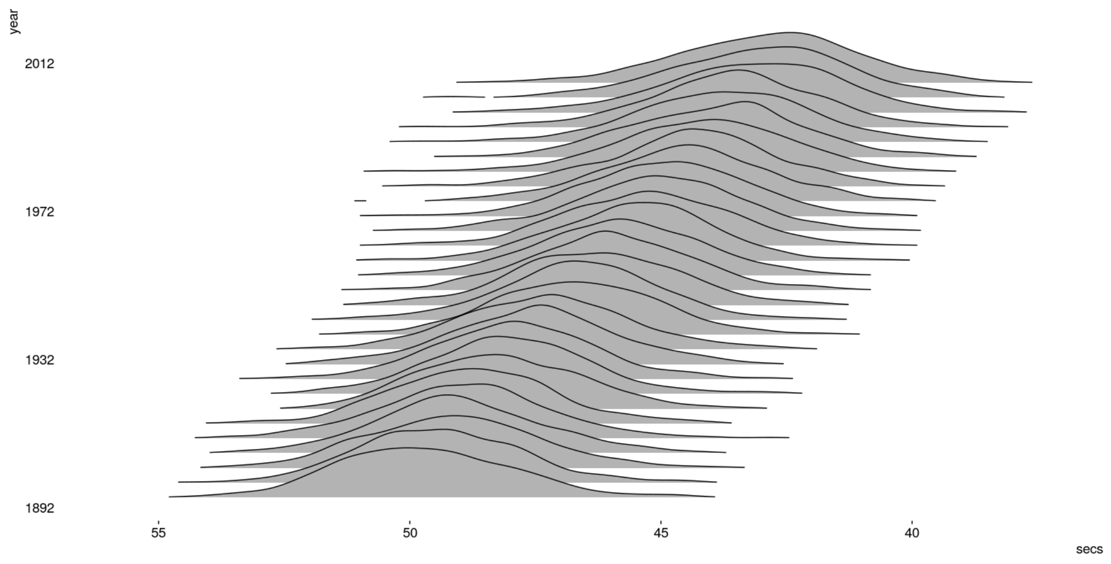
So that's uncertainty intervals... but really the Bayesian method gives us more: we have a distribution of predictions for each datapoint. This is called the *posterior predictive* distribution.

Posterior predictive - the whole picture

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

$$p(y \vee X, y, x, \theta) = \int p(y \vee x, \theta)\, p(\theta \vee X, y)\, d\theta$$

We have a distribution of predicted $\overset{\text{¿}}{y}$ s for every x :



This is an amazing amount of information we get when using the Bayesian approach!

However, there is one thing we haven't addressed yet: What if I use neural networks for prediction - what if use deep learning?

Uncertainty in deep neural networks

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.

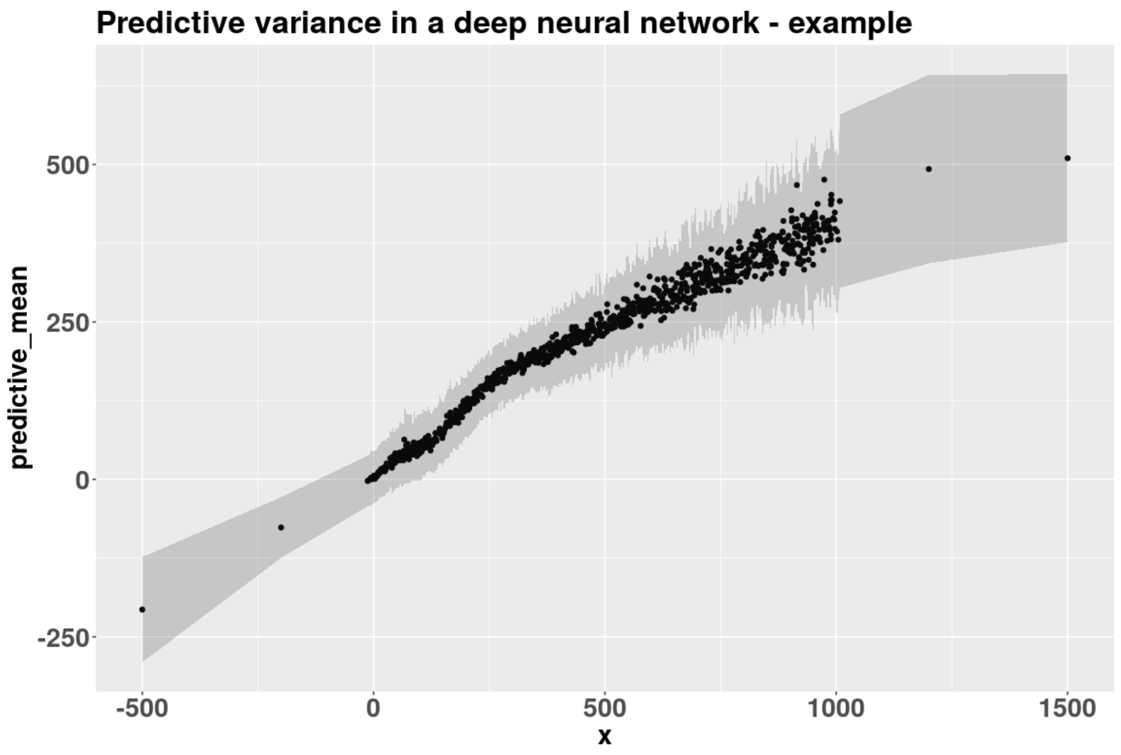
One the one hand, we could always apply the bootstrap approach, or even simple ensembling, to at least get a feel for the variability of the net's estimates. Among several other approaches that have been suggested, a prominent one is the use of *dropout* to calculate uncertainty.

Dropout networks as variational inference in Gaussian processes

Quoting from Yarin Gal's blog post *What my deep model doesn't know*,

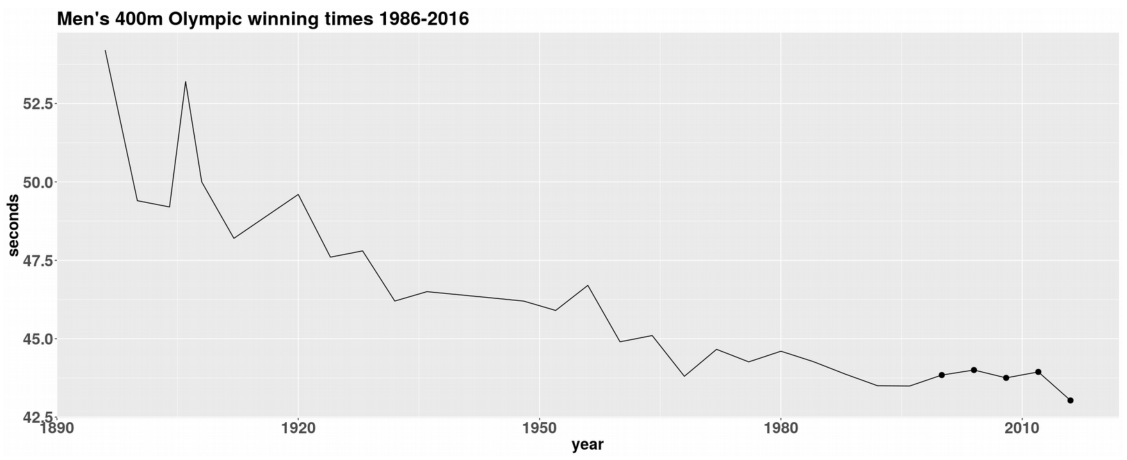
"... We'll see that what we did above, averaging forward passes through the network, is equivalent to Monte Carlo integration over a Gaussian process posterior approximation."

I've constructed a simple example to demonstrate this:



Ok. By now we've seen quite some methods and approaches, but ... we don't know *how* wrong we'd have been with our original point prediction (or any other one!). 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

As we have to stop at *some* time, here's the 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 just about everything.

Finally, if you have questions, don't hesitate to ask :-) Thank you!

Kontaktadresse:
Sigrid Keydana
Trivadis
Lehrer-Wirth-Strasse 4
D-81829 München

E-Mail	sigrid.keydana@trivadis.com
Internet:	www.trivadis.com