# GLM & GLMNET

Dr Liew How Hui

Jan 2021

## Announcements and Revision

- All physical tutorial class are converted to OTL.
- Physical practical class can stay physical but for this subject, it will be changed to OTL. Anyone disagree must voice it out now.
- Week 1: CRISP-DataMining (Modelling + Evaluation), Classification (Confusion Matrix) vs Regression (SSE/RSS, RMSE, $R^2$)
- Week 2: kNN Classifier vs kNN Regressor
- Week 3: Weighted kNN + Logistic Regression (No regressor!)
- Week 5: One hour quiz (or 1 week later?) in the second part of the lecture?

## Outline

# LinR and Bayesian LinR

There are two general approaches to statistical inferences:

**Frequentist approach**: the data sampled from the population is considered as random and the population parameter values, known as null hypothesis, as fixed (but unknown). To estimate thus this null hypothesis we look for the sample parameters that maximize the likelihood of the data. However, the data at hand, even it is sampled randomly from the population, it is fixed now, so how can we consider this data as random. The answer is that we assume that **the population distribution is known** and we work out the **maximum likelihood of the data using this distribution**. If we get very small value for the likelihood of the data which is known as p-value we tend to reject the null hypothesis.

The main problem is the misunderstanding and misusing of this p-value when we decide to reject the null hypothesis based on some threshold, from which we wrongly interpreting it as the probability of rejecting the null hypothesis.

# LinR and Bayesian LinR (cont)

**Bayesian approach**: provides true probabilities to quantify the uncertainty about a certain hypothesis, but requires the use of a first belief about how likely this hypothesis is true, known as **prior**, to be able to derive the probability of this hypothesis after seeing the data known as posterior probability. If a have population parameter to estimate $\theta$ , and we have some data sampled randomly from this population $D$, the posterior probability thus will be

$$\overbrace{p(\theta|D)}^{Posterior} = \frac{\overbrace{p(D|\theta)}^{Likelihood} \cdot \overbrace{p(\theta)}^{Prior}}{\underbrace{p(D)}_{Evidence}}.$$

The Evidence is the probability of the data at hand regardless the parameter $\theta$.

## LinR

Linear regression:

$$y_i = \mathrm{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_i, \quad i = 1, \ldots, n \tag{1}$$

where $\boldsymbol{\beta}$ is a $p \times 1$ vector, and the $\varepsilon_i \sim N(0, \sigma^2)$ are iid. The likelihood function is

$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \mathrm{x}_i^{\mathrm{T}}\boldsymbol{\beta})^2\right)$$

The frequentist approach assumes that there are **enough measurements** to say something meaningful about $\boldsymbol{\beta}$.

## LinR Example

```
### Linear Regression
set.seed(1) # For rnorm
n = 101
the.sigma = 3.1    # large noise
a = 1.5
b = 2.3
x = seq(-1,5,length=n)
y = a * x + b + rnorm(n,0,the.sigma)
lr_m = lm(y~., data.frame(x=x,y=y))
plot(x,y)
abline(b,a)
print(summary(lr_m))  # includes anova(lr_m)
```

## LinR Example (cont)

```
Call:
lm(formula = y ~ ., data = data.frame(x = x, y = y))

Residuals:
    Min      1Q  Median      3Q     Max
-7.2797 -1.8900 -0.0098  1.8240  7.1566

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.7043     0.4221   6.407 5.03e-09 ***
x             1.4554     0.1588   9.163 7.33e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.792 on 99 degrees of freedom
Multiple R-squared:  0.4589,    Adjusted R-squared:  0.4534
F-statistic: 83.96 on 1 and 99 DF,  p-value: 7.329e-15
```

Compare to $a = 1.5$ and $b = 2.3$, the estimate is “sort of” OK.

## Bayesian LinR

In the Bayesian Linear Regression, the data are supplemented with a prior probability distribution. Since the posterior distribution has no analytical expression for arbitrary prior, the conjugate prior (an inverse-gamma distribution) is used:

$$\text{``}\mathbb{P}(\theta)\text{''} = (\sigma^2)^{-\frac{v_0}{2}-1} \exp\left(-\frac{v_0 s_0^2}{2\sigma^2}\right).$$

where $vs^2 = (\mathrm{y} - \mathrm{X}\hat{\beta})^{\mathrm{T}}(\mathrm{y} - \mathrm{X}\hat{\beta})$, $v = n - p$, $p$ is the number of regression coefficients.

## Bayesian LinR (cont)

The prior and the likelihood below

$$\text{“}\mathbb{P}(D|\theta)\text{”} = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2}(\mathrm{y} - \mathrm{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathrm{y} - \mathrm{X}\boldsymbol{\beta})\right)$$

gives the posterior distribution with $D = \mathrm{y}, \mathrm{X}$, $\theta = \boldsymbol{\beta}, \sigma^2$ as follows according to Wikipedia:

$$\mathbb{P}(\boldsymbol{\beta}, \sigma^2 \mid \mathrm{y}, \mathrm{X}) \propto \mathbb{P}(\boldsymbol{\beta} \mid \sigma^2, \mathrm{y}, \mathrm{X})\mathbb{P}(\sigma^2 \mid \mathrm{y}, \mathrm{X}),$$

which is a product of two densities: $\mathcal{N}\left(\boldsymbol{\mu}_n, \sigma^2\boldsymbol{\Lambda}_n^{-1}\right)$ and Inv-Gamma$(a_n, b_n)$ distributions, with the parameters:

$$\boldsymbol{\Lambda}_n = (\mathrm{X}^{\mathrm{T}}\mathrm{X} + \Lambda_0), \quad \boldsymbol{\mu}_n = (\boldsymbol{\Lambda}_n)^{-1}(\mathrm{X}^{\mathrm{T}}\mathrm{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\Lambda}_0\boldsymbol{\mu}_0),$$

$$a_n = a_0 + \frac{n}{2}, \quad b_n = b_0 + \frac{1}{2}(\mathrm{y}^{\mathrm{T}}\mathrm{y} + \boldsymbol{\mu}_0^{\mathrm{T}}\boldsymbol{\Lambda}_0\boldsymbol{\mu}_0 - \boldsymbol{\mu}_n^{\mathrm{T}}\boldsymbol{\Lambda}_n\boldsymbol{\mu}_n).$$

## Bayesian LinR Example

With the same data as the LinR Example:

```
# https://www.r-bloggers.com/2020/04/bayesian-linear-regression/
library(rstanarm)  # more than 78 dependecies; rstan -> V8 nodejs lib
model_bayes = stan_glm(y~., data=data.frame(x=x,y=y), seed=111)
print(model_bayes, digits = 4)

# Analyse Posterior (rediculously many libraries to be called)
bayesplot::mcmc_dens(model_bayes, pars = c("x"))
bayestestR::describe_posterior(model_bayes)
post <- insight::get_parameters(model_bayes)
print(purrr::map_dbl(post, bayestestR::map_estimate),digits = 4)
print(purrr::map_dbl(post, mean),digits = 4)
```

## Bayesian LinR Example (cont)

```
SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.9e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition woul
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.022774 seconds (Warm-up)
Chain 1:                0.02603 seconds (Sampling)
Chain 1:                0.048804 seconds (Total)
Chain 1:
```

# Bayesian LinR Example (cont)

```
SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 5e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition woul
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.020656 seconds (Warm-up)
Chain 4:                0.026521 seconds (Sampling)
Chain 4:                0.047177 seconds (Total)
Chain 4:
```

## Bayesian LinR Example (cont)

There are four MCMC simulation! For slightly more complex model, there will be much more and longer computer simulation.

Bayesian methods are complex but the estimation of $a = 1.5$, $b = 2.3$, $\sigma = 3.1$ are similar to linear regression:

```
stan_glm
 family:       gaussian [identity]
 formula:      y ~ .
 observations: 101
 predictors:   2
------
            Median MAD_SD
(Intercept) 2.7053 0.4220
x           1.4521 0.1532

Auxiliary parameter(s):
      Median MAD_SD
sigma 2.8045 0.1990

------
# Description of Posterior Distributions
```

## Bayesian LinR Example (cont)

```
Parameter    | Median |          89% CI |       pd |        89% ROPE | %
Rhat |       ESS
-------------------------------------------------------------------------
(Intercept) |  2.705 | [2.035, 3.346] | 100.00% | [-0.378, 0.378] |
0 | 1.000 | 3863.152
x           |  1.452 | [1.217, 1.713] | 100.00% | [-0.378, 0.378] |
0 | 1.000 | 4035.144

(Intercept)           x
      2.643       1.449
(Intercept)           x
      2.713       1.453
```

## Outline

# Generalised Linear Model (GLM)

In the early 1970s Nelder and Wedderburn (1972) identified a class of models that generalises the multiple linear regression (1) as the *generalised linear models (GLMs)*:

$$g(\mathbb{E}[Y_i]) = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} \tag{2}$$

Here $g$ is called a *link function* and the *variance function* is assumed to depend on the mean $\mathbb{E}[Y_i]$:

$$\mathrm{Var}(Y_i) = V(\mathbb{E}[Y_i]).$$

The parameters $\beta_i$, $i = 0, 1, \ldots, p$ are **unknown** and need to be estimated. $\beta_0$ is the *intersect* / *bias*.

## GLM (cont)

### GLM is supported in R

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, singular.ok = TRUE, contrasts = NULL, ...)
```

'family': gaussian (for linear regression), binomial (for logistic regression), Gamma, inverse.gaussian, poisson, quasi, quasibinomial, quasipoisson.

The theory and applications should be covered in "Applied Regression Analysis".

## Regression with GLM (cont)

When $Y_i \sim Normal(\mu, \sigma^2)$, $\mathbb{E}[Y_i] = \mu$, $\text{Var}(Y_i) = \sigma^2$ with $g(\mu) = \mu$, $V(\mu) = \sigma^2$. We obtain the *multiple linear regression* model (1) above.

Logistic regression (LR) model

When $Y_i \sim Binomial(n_i, p_i)$, $\mathbb{E}[Y_i] = n_i p_i$, $\text{Var}(Y_i) = p_i(1 - p_i)$. The GLM becomes a LR model

$$\ln \frac{P(Y = 1|X)}{1 - P(Y = 1|X)} = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi}$$

with the link function and variance function

$$g(\mu_i) = \underbrace{\ln \frac{\mu_i}{1 - \mu_i}}_{\text{logit}}, \quad V(\mu_i) = \mu_i(1 - \mu_i).$$

## GLM (cont)

### Poisson regression model

If the response (e.g. number of claims) $Y_i \sim \mathit{Poisson}(\lambda_i)$, for $i = 1, \cdots, n$, then $\mathbb{E}[Y_i|X_i] = \lambda_i = \mathrm{Var}(Y_i)$ with $g(\mu) = \ln\mu$, $V(\mu) = \mu$ and we have

$$\ln\mathbb{E}[Y_i|X_i] = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_p X_{pi}.$$

```
R:
  glm(x, y, family=poisson)

Python:
  statsmodels.genmod.generalized_linear_model.GLM(endog, exog,
    family=None, offset=None, exposure=None, freq_weights=None,
    var_weights=None, missing='none', **kwargs)
  family = Binomial, Gamma, Gaussian, InverseGaussian,
    NegativeBinomial, (Poisson), Tweedie

  sklearn.linear_model.PoissonRegressor(alpha=0)
  alpha>0 => GLMNET
```

## GLM (cont)

Poisson regression model (cont)

Example: The number of people in line in front of you at the grocery store. Predictors may include the number of items currently offered at a special discounted price and whether a special event (e.g., a holiday) is a few days/weeks away.

## GLM (cont)

### Exponential/Gamma regression model

If $Y_i \sim Exponential(\lambda_i)$, $\mathbb{E}[Y_i] = \frac{1}{\lambda_i}$, $Var[Y_i] = \frac{1}{\lambda_i^2}$ with $g(\mu) = -\frac{1}{\mu}$, $V(\mu) =$? and

$$-\frac{1}{\mathbb{E}[Y_i|X_i]} = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_p X_{pi}.$$

```
R:
  glm(x, y, family=Gamma)

Python:
  statsmodels.genmod.generalized_linear_model.GLM(endog, exog,
    family=None, offset=None, exposure=None, freq_weights=None,
    var_weights=None, missing='none', **kwargs)
  family = Gamma

  sklearn.linear_model.GammaRegressor(alpha=0)
```

## Regression with GLM (cont)

### Inverse gaussian regression model

If the response $Y_i$, for $i = 1, \cdots, n$, follows an exponential distribution, then

$$\frac{1}{\mathbb{E}[Y_i|X_i]^2} = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_p X_{pi}.$$

```
R:
  glm(x, y, family=inverse.gaussian)

Python:
  statsmodels.genmod.generalized_linear_model.GLM(endog, exog,
    family=None, offset=None, exposure=None, freq_weights=None,
    var_weights=None, missing='none', **kwargs)
  family = InverseGaussian

  sklearn.linear_model.TweedieRegressor(power=3.0, alpha=0)
```

# GLM (cont)

Other models? The General Exponential Family?

- To be added in future: Quasi?, Tweedie?

Statistics report on GLM:

- $p$=number of parameters in GLM, $n$=number of observations.
- Akaike Information Criterion (AIC$=-2l(\beta)+2p$) and Bayesian Information Criterion (BIC$=-2l(\beta)+\log(n)p$): Both measures are based on the log-likelihood, but penalize for the number of parameters included in the model.

## GLM (cont)

Parameter estimation of $\beta_i$

MLE + Newton's method: A Fisher's scoring method:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + \mathcal{I}^{-1}(\boldsymbol{\beta}^{(t)})u(\boldsymbol{\beta}^{(t)}),$$

where $\mathcal{I}(\boldsymbol{\beta}^{(t)})$ is the Fisher information matrix. Note that if the canonical link function is used, then they are the same.

## Outline

## GLMNET = GLM via Penalized Maximum Likelihood

The R package `glmnet` solves the problem

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^{N} w_i \ell(y_i, \beta_0+\beta^T x_i)+\lambda\left[(1-\alpha)\frac{\|\beta\|_2^2}{2}+\alpha\|\beta\|_1\right], \tag{3}$$

over a grid of values of $\lambda$ covering the entire range. Here $l(y,\eta)$ is the negative log-likelihood contribution for observation $i$; e.g. for the Gaussian case it is $\frac{1}{2}(y-\eta)^2$. The elastic-net penalty is controlled by $\alpha$, and bridges the gap between lasso ($\alpha = 1$, the default) and ridge ($\alpha = 0$). The tuning (hyper)parameter $\lambda$ controls the overall strength of the penalty.

## Regression with GLM (cont)

ElasticNet Linear Regression

$$\min_{\beta} \left\{ \frac{1}{2n} \|X\beta - y\|_2^2 + \lambda \left[ \alpha \|\beta\|_1 + (1-\alpha) \frac{\|\beta\|_2^2}{2} \right] \right\} \quad (4)$$

where $n$ is number of samples, $0 \le \alpha \le 1$ is the elastic mixing parameter.

```
R:
  glmnet(x, y, family="gaussian"/"mgaussian", alpha, 0<=lambda)

Python:
  sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5,
    fit_intercept=True, normalize=False, precompute=False,
    max_iter=1000, copy_X=True, tol=0.0001, warm_start=False,
    positive=False, random_state=None, selection='cyclic')
```

# Regression with GLM (cont)

## Lasso regression

When $\alpha = 1$ in (4), we have the *lasso*,

$$\min_{\beta} \left\{ \frac{1}{2n} \|X\beta - y\|_2^2 + \lambda \|\beta\|_1, \right\} \tag{5}$$

a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer non-zero coefficients, effectively reducing the number of features upon which the given solution is dependent $\rightarrow$ compressed sensing.

```
sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True,
  normalize=False, precompute=False, copy_X=True, max_iter=1000,
  tol=0.0001, warm_start=False, positive=False,
  random_state=None, selection='cyclic')
```

## Regression with GLM (cont)

Ridge regression

When $\alpha = 0$ in (4), we have the *ridge regression*:

$$\min_{\beta} \left\{ \frac{1}{2n} \|X\beta - y\|_2^2 + \frac{\lambda}{2} \|\beta\|_2 \right\} \tag{6}$$

where $\lambda$ is the Lagrange multiplier of the constraint.

```
sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True,
  normalize=False, copy_X=True, max_iter=None, tol=0.001,
  solver='auto', random_state=None)
```

alpha corresponds to $1/(2C)$ in other linear models such as LogisticRegression or LinearSVC.

# GLMNET (cont)

<u>Elastic multinomial LR</u>

A generalisation of the usual LR model is

$$\min_{(\beta_{0k},\beta_k)_1^K \in \mathbb{R}^{(p+1)K}} -\left[\frac{1}{n}\sum_{i=1}^n \left(\sum_{k=1}^K y_{ik}\cdot(\beta_{0k}+\beta_k^T x_i) - \ln(\sum_{k=1}^K e^{\beta_0+\beta^T x_i})\right)\right] + \lambda\left[(1-\alpha)\frac{\|\beta\|_F^2}{2} + \alpha\sum_{j=1}^p \|\beta_j\|_q\right] \tag{7}$$

where $q = 1$ (lasso penalty) or 2 (grouped-lasso penalty).

```
R:
  glmnet(x, y, family="binomial"/"multinomial", alpha, 0<=lambda)

Python:
  sklearn.linear_model.LogisticRegression(penalty='l2', *,
  dual=False, tol=0.0001, C=1.0, fit_intercept=True,
  intercept_scaling=1, class_weight=None, random_state=None,
  solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
  warm_start=False, n_jobs=None, l1_ratio=None)
```

# GLMNET (cont)

Note: Python's C=$\frac{1}{\lambda}$ $= 1/$(R's lambda), Python's l1_ratio $= \alpha =$ R's alpha.

## Elastic Poisson Regression Model

It should look something like this (not sure):

$$\min_{\beta_0,\beta} -\frac{1}{n}\sum_{i=1}^{n}\Big((y_i(\beta_0+\beta^T x_i) - e^{(\beta_0+\beta^T x_i)} - \ln(y_i!)\Big) + \lambda\left[(1-\alpha)\|\frac{\beta\|_2^2}{2} + \alpha\|\beta\|_1\right],$$

```
R:
  glmnet(x, y, family="poisson", 0<=alpha<=1, 0<=lambda)

Python:
  sklearn.linear_model.PoissonRegressor(*, alpha=1.0,
    fit_intercept=True, max_iter=100, tol=0.0001, warm_start=False,
    verbose=0)
```

# GLMNET (cont)

Elastic Exponential/Gamma Regression Model

It should look something like this (not sure):

$$\min_{\beta_0,\beta} -\frac{1}{n}\sum_{i=1}^{n}\left((y_i(\beta_0+\beta^T x_i) - e^{(\beta_0+\beta^T x_i)} - \ln(y_i!)\right) + \lambda\left[(1-\alpha)\|\frac{\beta\|_2^2}{2} + \alpha\|\beta\|_1\right],$$

```
sklearn.linear_model.GammaRegressor(*, alpha=1.0,
  fit_intercept=True, max_iter=100, tol=0.0001,
  warm_start=False, verbose=0)
```

## Outline

## Locally Weighted Learning (LWL)

LWL or local regression is a generalisation of linear regression

$$\min_{\beta} \sum_{i} (\beta_0 + \beta_1 x_1^{(1)} + \cdots + \beta_p x_n^{(p)} - y_i)^2$$

to "something" different but with some similarity to kNN:

$$C(\mathrm{x}) = \sum_{i} L(f(\mathrm{x}_i, \beta), y_i) \cdot K(d(\mathrm{x}_i, \mathrm{x})) \tag{8}$$

where $K$ is the kernel, $f$ is a general model and $L$ is the loss function.

The $\beta$ is only estimated when x is given. This is similar to kNN, i.e. no training, only "predict".

## LWL (cont)

A special case of the LWL is the local (polynomial) regression (see `https://en.wikipedia.org/wiki/Local_regression`). Compare to kNN or weighted kNN, it is more complex and:

- It requires fairly large, densely sampled data sets in order to produce good models.
- It does not produce a regression function that is easily represented by a mathematical formula & computationally intensive.
- Prone to outliers.

## LWL (cont)

In R, local regression is available as

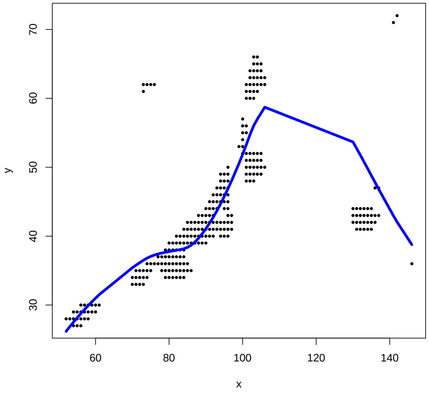
```
loess(formula, data, weights, subset, na.action, model = FALSE,
       span = 0.75, enp.target, degree = 2,
       parametric = FALSE, drop.square = FALSE, normalize = TRUE,
       family = c("gaussian", "symmetric"),
       method = c("loess", "model.frame"), ...)
```

An example can be found in
`https://stackoverflow.com/questions/15337777/fit-a-line-with-loess-in-r`

```
# https://stackoverflow.com/questions/15337777/fit-a-line-with-loess-
# load(url('https://www.dropbox.com/s/ud32tbptyvjsnp4/data.R?dl=1'))
load("data.R")
lw1 = loess(y ~ x, data)
plot(y ~ x, data=data, pch=19, cex=0.5)
j = order(data$x)
lines(data$x[j],lw1$fitted[j],col="blue",lwd=4)
```

## LWL (cont)

Like kNN, the fitted data could be obtained using predict(). However, in the example, lines() is used to plot rather than to “predict”.



## LWL (cont)

A special case of local regression is the Nadaraya-Watson kernel regression estimate (related to moving average), which is implemented in R:

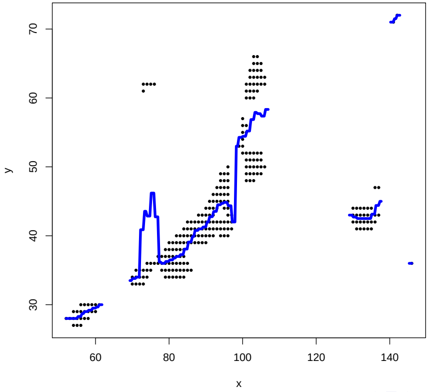
```
ksmooth(x, y, kernel = c("box", "normal"), bandwidth = 0.5,
        range.x = range(x), n.points = max(100L, length(x)), x.points
```

An example using the same data is

```
# https://stackoverflow.com/questions/15337777/fit-a-line-with-loess-
# load(url('https://www.dropbox.com/s/ud32tbptyvjsnp4/data.R?dl=1'))
load("data.R")
#ks = ksmooth(data$x, data$y, "normal", bandwidth=8)
ks = ksmooth(data$x, data$y, bandwidth=2)
plot(y ~ x, data=data, pch=19, cex=0.5)
j = order(data$x)
lines(ks, col="blue",lwd=4)
```

## LWL (cont)

It does not predict the parts without any data and is obviously prone to outliers (for the default box filter with bandwidth 2):



y
30
40
50
60
70
60
80
100
120
140
x