Applied Inductive Learning
Project 2 - Bias and Variance Analysis

Javaux Raphael & Javaux Maxime
University of Liege, Applied sciences

November 22, 2015

---

1 Theoretical questions

1.1 Bayes model and residual error in classification

a) We are studying a classification problem so the Bayes model $h_b(x_1, x_2)$ correspond to: $h_b(x_1, x_2) = arg \max_c P(y = c|x_1, x_2)$

We are going to use the Bayes theorem to change the conditional probability $P(y = c|x_1, x_2)$. We have:

$$h_b(x_1, x_2) = arg \max_c P(y = c|x_1, x_2) \tag{1}$$
$$= arg \max_c P(y = c \cap (x_1, x_2))/P(x_1, x_2) \tag{2}$$
$$= arg \max_c P((x_1, x_2)|y = c)P(y = c)/P(x_1, x_2) \tag{3}$$

In our case, the y class is drawn uniformly at random from {0,1} and so:

$$h_b(x_1, x_2) = arg \max_c P((x_1, x_2)|y = c) \tag{4}$$

We can simplify the calculation because the information is given as polar coordinates and x_1, x_2 are Cartesian coordinates linked to this radius r_i and angle α_i. In addition, α_i is uniformly distributed and so, the only variable which is relevant is the radius.

$$h_b(x_1, x_2) = arg \max_c P((x_1, x_2)|y = c) \tag{5}$$
$$= arg \max_c P((r_i, \alpha_i)|y = c) \tag{6}$$
$$= arg \max_c P(r_i|y = c) = h_b(r_i) \tag{7}$$

We have to compute the two probabilities $P(r_i|y = 0)$ and $P(r_i|y = 1)$. As r_i can take continuous values, we will compute a probability density which will represent each probability

of the Bayes model.

$$h_b(r_i) = arg \max_c P(r_i|y=c) \tag{8}$$

$$= arg \max_c f(r_i|y=c) \tag{9}$$

For $f(r_i|y=0)$, we have:

$$f(r_i|y=0) = \frac{1}{2}\mathcal{N}(R_1, \sigma^2) + \frac{1}{2}\mathcal{N}(2*R_1, \sigma^2) \tag{10}$$

$$= \frac{1}{2\sigma\sqrt{2\pi}} \left(e^{\frac{-1}{2\sigma^2}(r_i - R_1)} + e^{\frac{-1}{2\sigma^2}(r_i - 2*R_1)} \right) \tag{11}$$

By the same way, we have:

$$f(r_i|y=1) = \frac{1}{2\sigma\sqrt{2\pi}} \left(e^{\frac{-1}{2\sigma^2}(r_i - R_2)} + e^{\frac{-1}{2\sigma^2}(r_i - 2*R_2)} \right) \tag{12}$$

Both $f(r_i|y=0)$ and $f(r_i|y=1)$ can be seen in the figure 1. To find the right Bayes model, we have to choose the output that gives the highest probability density for each radius and so, find intersections between curves.

We can calculate intersections analytically or numerically.

- Analytically: Find exacts intersections is difficult but we can use some assumptions; if $\sigma << R_1$, $\sigma << R_2$ and $\sigma << |\frac{R_1+R_2}{2}|$. In this case, the effect of both normal distributions of each $f(r_i|y=c)$ is limited and do not change the value of the other. Due to that, we can compute intersections by taking into account the increasing sequence of R_1,$2R_1$,R_2. In our case, we have $R_1 < R_2 < 2R_1 < 2R_2$. So, there is an intersection between R_1, R_2, one other between $R_2, 2R_1$ and a last one between $2R_1, 2R_2$. Those intersections are extremely close to $\frac{R_1+R_2}{2} = 0.6$, $\frac{R_2+2R_1}{2} = 0.85$ and $\frac{2R_1+2R_2}{2} = 1.2$
- Numerically: We can use a solver that converge to intersections. In our case, the three intersections are the same as computed in the simplified analytic way.

As the two curves intersect three times at $r_i = 0.6,\ 0.85,\ 1.2$ we have four spaces: $]-\infty,\ 0.6[$, $]0.6,\ 0.85[$, $]0.85,\ 1.2[$ and $]1.2,\ \infty[$.

As seen in the figure 1, the Bayes model will predict:

$$h_b(r_i) = arg \max_c f(r_i| = c) \tag{13}$$

$$= \begin{cases} 1, & \text{if } r_i \in\ ]-\infty,\ 0.6[\cap\ ]0.85,\ 1.2[\\ 0, & \text{if } r_i \in\ ]0.6,\ 0.85[\cap\ ]1.2,\ \infty[\end{cases} \tag{14}$$

b) The generalization error is the error made by the Bayes model. We can see that the smallest probability density in each four spaces that we declared at the point a) is not zero. So that, there is a given probability to be in those spaces with the wrong prediction which can be computed by the integral of the smallest probability density in those spaces times the probability to have the linked y. In our case, we have: [1]

[1] $[]_{y=c}$ denote the union of spaces where the y=c is predicted.

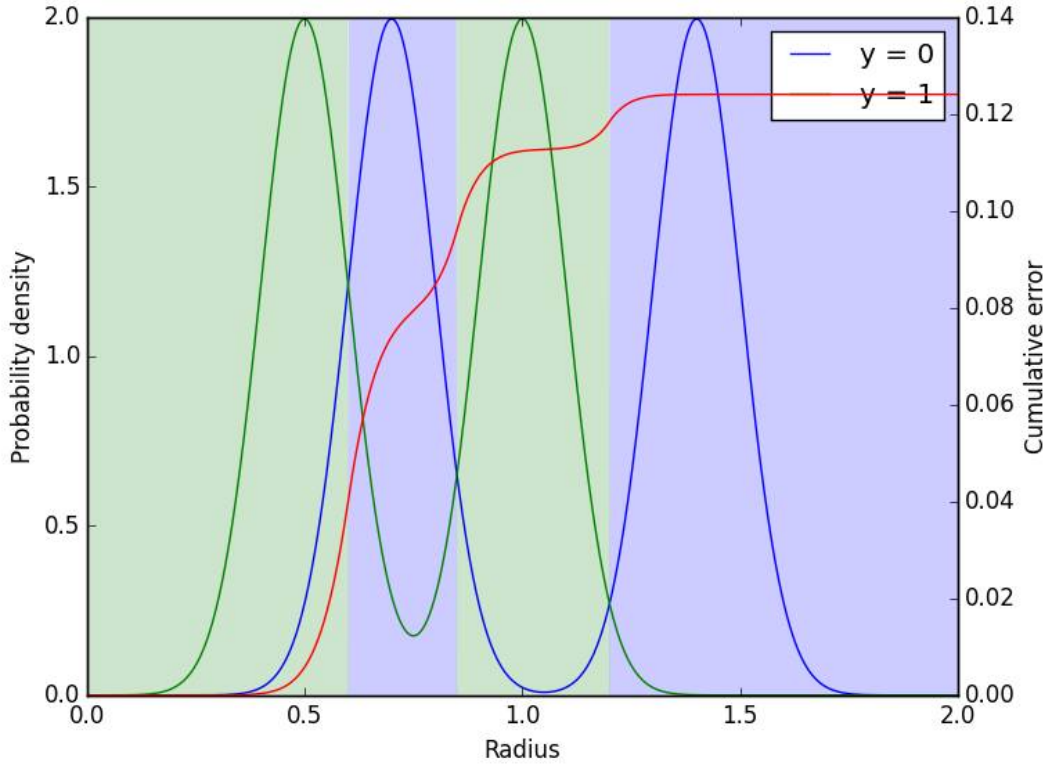


Figure 1: $f(r_i|y=0)$ and $f(r_i|y=1)$

$$
\begin{aligned}
E_{x_1,x_2,y}\{1(y \neq h_b(x_1,x_2)\} =& E_{r_i,y}\{1(y \neq h_b(r_i)\} && (15)\\
=& \int_{[]_{y=1}} f(r_i|y=0)P(y=0)\,dr_i + \int_{[]_{y=0}} f(r_i|y=1)P(y=0)\,dr_i && (16)\\
=& \frac{1}{2}\int_{-\infty}^{0.6} f(r_i|y=0)\,dr_i + \frac{1}{2}\int_{0.6}^{0.85} f(r_i|y=1)\,dr_i && (17)\\
&+\frac{1}{2}\int_{0.85}^{1.2} f(r_i|y=0)\,dr_i + \frac{1}{2}\int_{1.2}^{\infty} f(r_i|y=1)\,dr_i && (18)\\
=& \frac{1}{2}(0.079+0.113+0.045+0.011) && (19)\\
=& 0.124 \simeq 12\% && (20)
\end{aligned}
$$

The red curve in the figure 1 represents the evolution of the cumulative error with the radius. This curve can be computed with integrals (17) limited with r_i.

This curve shows that the main error is realized when $r_i \in]0.4, 1.3[$. Out of this range, the error is nearly zero.

1.2 Bias and variance of the K-NN algorithm

a) As shown in the theoretical course, we have three terms that intervene in $E_{LS}\{E_{y|x}\{(y-\hat{y}(X;LS,k))^2\}\}$ which are the noise, the bias and the variance.

As $y = f(x) + \epsilon$, we can compute the Bayes model:

$$\begin{aligned} h_b(x) &= E_{y|x}\{y\} && (21)\\ &= E_{y|x}\{f(x) + \epsilon\} && (22)\\ &= E_{y|x}\{f(x)\} = f(x) && (23) \end{aligned}$$

In addition, we use the k-nearest neighbors algorithm and so:

$$\hat{y} = \frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)}) + \epsilon\big) \tag{24}$$

The **noise** quantifies how much the Bayes model varies from y: $noise(x) = E_{y|x}\{(y - h_b(x))^2\}$. In our case, we can use the following steps:

$$\begin{aligned} noise(x) =& E_{y|x}\{(y - h_b(x))^2\} && (25)\\ =& E_{y|x}\{(f(x) + \epsilon - f(x))^2\} && (26)\\ =& E_{y|x}\{\epsilon^2\} && (27)\\ =& var(\epsilon) = \sigma^2 && (28) \end{aligned}$$

The **bias** measures the error made between the Bayes model and the average model: $bias^2(x) = (h_b(x) - E_{LS}\{\hat{y}(x)\})^2$; In our case, we have:

$$\begin{aligned} bias^2(x) =& (h_b(x) - E_{LS}\{\hat{y}(x)\})^2 && (29)\\ =& \left(f(x) - E_{LS}\{\frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)}) + \epsilon\big)\}\right)^2 && (30)\\ =& \left(f(x) - E_{LS}\{\frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)})\} + \underbrace{E_{LS}\{\epsilon\}}_{=0}\right)^2 && (31)\\ =& \left(f(x) - \frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)})\right)^2 && (32) \end{aligned}$$

The **variance** quantifies how much the learned model varies from one learning sample

to another: $variance(x) = E_{LS}\{(\hat{y}(x) - E_{LS}\{(\hat{y}(x)\})^2\}$. This term will be in our case:

$$
\begin{aligned}
variance(x) =& E_{LS}\{(\hat{y}(x) - E_{LS}\{(\hat{y}(x)\})^2\} && (33)\\
=& E_{LS}\{\left(\frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)}) + \epsilon\big) - E_{LS}\{\frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)}) + \epsilon\big)\}\right)^2 && (34)\\
=& E_{LS}\{\left(\frac{1}{k}\sum_{l=1}^{k}\epsilon\right)^2\} && (35)\\
=& \frac{1}{k^2}E_{LS}\{\left(\sum_{l=1}^{k}\epsilon\right)^2\} && (36)\\
=& \frac{1}{k^2}var\left(\underbrace{\sum_{l=1}^{k}\epsilon}_{\text{Sum uncorrelated}}\right) && (37)\\
=& \frac{1}{k^2}\sum_{l=1}^{k}var(\epsilon) && (38)\\
=& \frac{\sigma^2}{k} && (39)\\
& && (40)
\end{aligned}
$$

Finally, we have the expected equation:

$$E_{LS}\{E_{y|x}\{(y - \hat{y}(X; LS, k))^2\}\} = \sigma^2 + \left(f(x) - \frac{1}{k}\sum_{l=1}^{k}\big(f(x_{(l)})\right)^2 + \frac{\sigma^2}{k} \quad (41)$$

b) As the first term is the **irreducible error**, it cannot be changed and so, k has no effect on it. Ideally, it is the minimum error that we can achieve.

The bias represents the square error done between the Bayes model and the average model. As nearest neighbors will have close value to the real, the bias will be smaller for small k and will increase after due to the fact that further neighbors can have any value.

The third term is the variance and describes how much the model will vary between different learning samples. By opposition to the bias, this term will decrease when we increase the k neighbors took into account. When k increases, we take more neighbors into account which leads to a decrease of the noise impact.

In general, the higher the complexity, the higher is the variance but the smaller is the bias. In this case, the complexity is set by the k parameter; when k increases, the complexity decreases and so, the bias increases and the variance decreases.

2 Empirical analysis

The given function $f(x) = (sin(x) + cos(x)) * x^2 + e$ can be used to generate a large number of input samples. These samples can be used to accurately estimate the residual error, the squared bias, the variance, and thus the expected error of any regression algorithm, by numerical computing these values.

Before estimating the error values, we must generate a large number of learning sets using $f(x)$, and we must train a regression for each of these sets:

```
def train_regressions():
    regressions = []
    for i in N_LS:
        X = np.random.uniform(-9, 9, LS_SIZE)
        y = f(X)

        regressions.append(trained regression on X and y)

    return regressions
```

To give accurate estimates, N_LS (the number of learning sets/trained regressions) and LS_SIZE (the size of the generated learning sets) should be sufficiently large numbers. We defined $N\_LS = 100$ and $LS\_SIZE = 1000$.

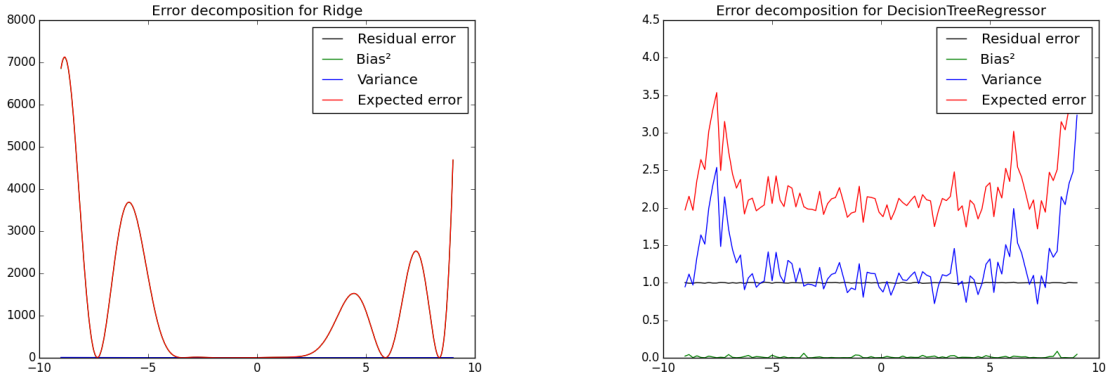
We can then estimate the error values for a given point x_0, and for the trained regressions, by generating a large number of samples of $f(x_0)$, and by evaluating the trained regressions on it:

```
def estimate_error(x_0):
    # Generates 'N_SAMPLES_ESTIMATE' samples on point 'x_0' with
    # the function 'f(x)'.
    X = np.empty(N_SAMPLES_ESTIMATE)
    X.fill(x_0)
    y = f(X)

    # Computes the values as predicted by the trained regressions
    # for 'x_0'.
    y_predict = np.array([r.predict(x_0) for r in regressions])

    # Estimates the residual error by computing the mathematical
    # variance of the sample set generated on point 'x_0'.
    residual_error = np.var(y)

    # Estimates the squared bias by computing the mean squared
    # difference of the mean of all output 'y' ('mu_y'), and the mean
    # of the predicted outputs 'y_predict' ('mu_y_predict').
```

```
    mu_y = np.mean(y)
    mu_y_predict = np.mean(y_predict)
    sq_bias = np.mean((mu_y - mu_y_predict)**2)

    # Estimates the variance by computing the mean squared difference
    # between the predicted outputs and the mean of the predicted
    # outputs.
    var = np.mean((y_predict - mu_y_predict)**2)

    # Mean square error (MSE).
    mse = residual_error + sq_bias + var

    return residual_error, sq_bias, var, mse
```

To give accurate estimates, $N\_SAMPLES\_ESTIMATE$ (the number samples generated for x_0) should be sufficiently large. A value of $10,000$ is sufficiently high for the mean square error as computed by this algorithm to be within 1% of the value returned by *mean_squared_error*() of the *Scikit-Learn* framework.

2.1 Error values as a function of x



(a) (b)

Figure 2a shows the evolution of the error components as estimated by our algorithm for the Ridge linear regression (*sklearn.linear_model.Ridge*), as a function of x.
The expected error is very high, and nearly only composed of the squared bias (the two curves are so closes that they are mostly indistinguishable in the graph). The very large value of the squared bias indicates that the regression is not able to correctly fit the learning sample. The $f(x)$'s slope can not be fitted correctly by the too simplistic linear regression.

Figure 2b shows the evolution of the error components as estimated by our algorithm for a trained decision tree (*sklearn.tree.DecisionTreeRegressor*), as a function of x.

This algorithm performs far better than the linear regression. Its error is mainly determinated by an high variance. That means that the decision trees over-fit the training samples (the tree are complete). Applying some pruning, or using some ensemble methods (random forests), should greatly reduce the error of this algorithm.

In both plots, the residual error is equal to 1, for any value of x. This exact value should be expected, given the e noise variable that has been introduced in the formula.

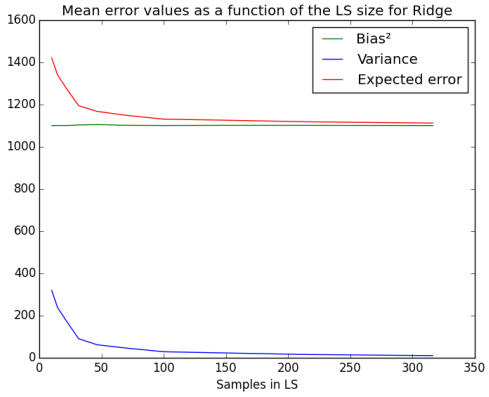
2.2 Mean error values over the input space

Table 1: Mean error rates over the input space

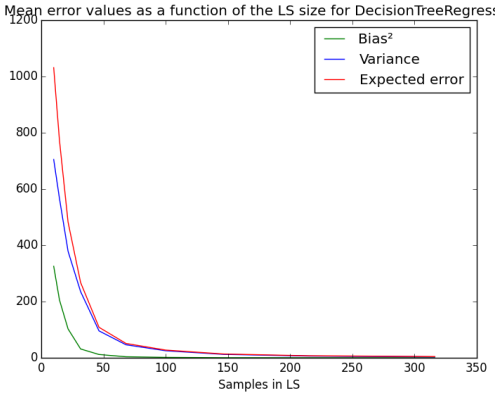
Algorithm	Res. error	Squared bias	Variance	MSE
Ridge	1.000	1082.348	3.747	1087.095
DecisionTreeRegressor	1.000	0.013	1.218	2.231
KNeighborsRegressor	1.000	0.007	0.519	1.526

Our error estimation algorithm can also be used to estimate the mean values of the error components by running it on several values of x_0, and by averaging the estimated errors. Table 1 exhibits the error values for three regression algorithms, obtained by averaging the values obtained over 1000 points uniformly distributed between -9 and 9.
The k-nearest neighbors algorithms performs the best, and its mean square error (MSE) is very close to the minimal attainable error rate. The algorithm seems to accommodate itself very well to a function that have very soft curves as $f(x)$ has.

2.3 Mean error values as a function of the learning set size



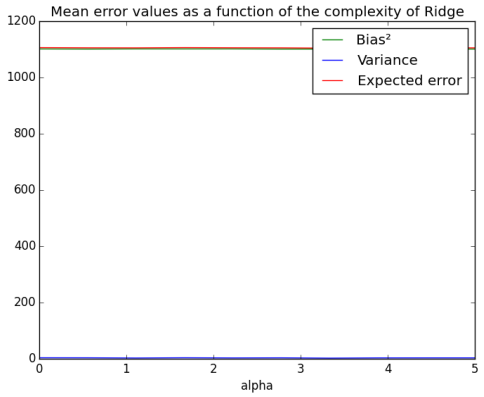
(a)



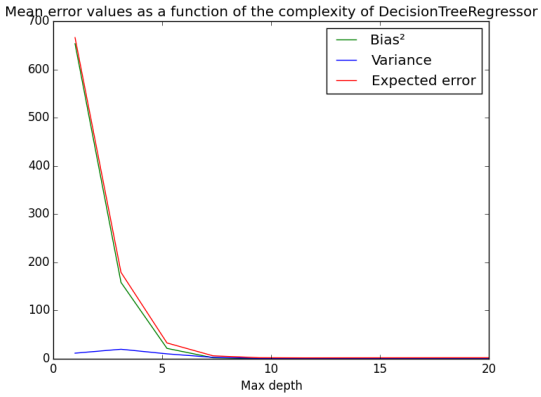
(b)

Figures 3a and 3b show the evolution of the error components when the size of the learning sets used to train the regressions increases.
In the case of the *Ridge* regression, increasing the number of samples in the learning set does not reduce the squared bias. The algorithm fails to learn more from the learning set.
In the case of the decision tree regression, the algorithm rapidly fits itself to the learning set (the squared bias decreases very early), and the error rate still decreases with larger learning sets.

2.4 Mean error values as a function of the model complexity



(a)

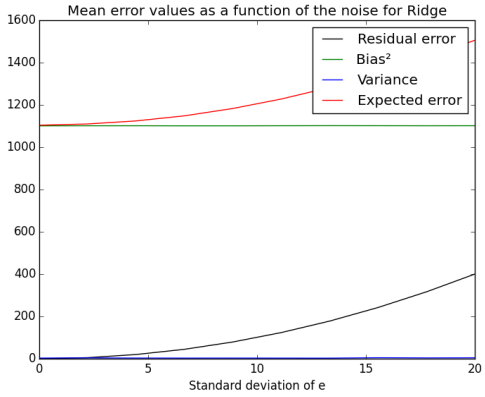


(b)

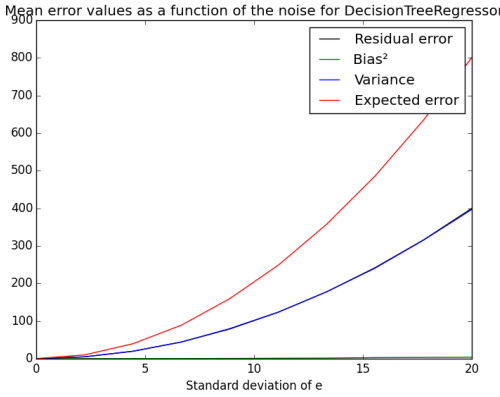
Figures 4a and 4b show the evolution of the error components when the complexity of the model is increased.
In the case of the *Ridge* regression, increasing the *alpha* factor has no effect on the error. This is because the *alpha* factor controls how much the regression weights input attributes, and it has no effect on a single dimension problem.
In the case of the decision tree regression, increasing the complexity increases the performances of the regression. Strangely, there is no visible artifact of over-fitting: the variance does not seem to increase on very-depth trees.

2.5 Mean error values as a function of the standard deviation of the noise

Figures 5a and 5b show the evolution of the error components when the standard deviation of the noise is increased.
In the case of the *Ridge* regression, increasing the standard deviation has the effect of only increasing the residual error. That is because the algorithm performs very poorly, and is thus relatively insensitive to noise.
In the case of the decision tree regression, the trees, being complete/unpruned, are very sensitive to noise. Both the residual error and the variance increase with the noise (both curves overlap).

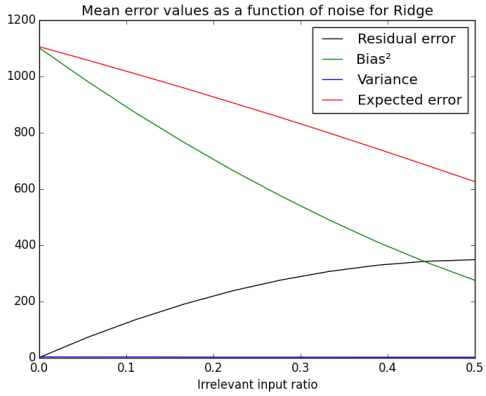


(a)

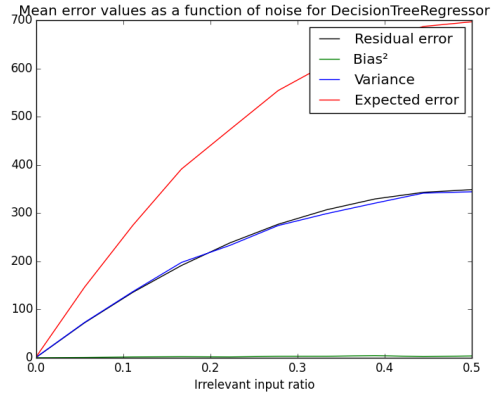


(b)

2.6 Mean error values as a function of the ratio of irrelevant data



(a)



(b)

Figures 6a and 6b show the evolution of the error components when the irrelevant data are returned by $f(x)$. The ratio indicates the probability of the $f(x)$ function to return an irrelevant value taken randomly in $U(-9, 9)$

In the case of the *Ridge* regression, the regression performs better with an high number of irrelevant input. That is because the regression is better at predicting these irrelevant input (randomly taken in $U(-9, 9)$) than at predicting the proper values of $f(x)$. The behavior of the decision tree regression is very similar to the one encountered in section 2.5. The trees are very sensitive to noise, and the variance increase at the same pace as the residual error.