Data Mining
Tree ensembles:
Bagging, Boosting and Random Forests

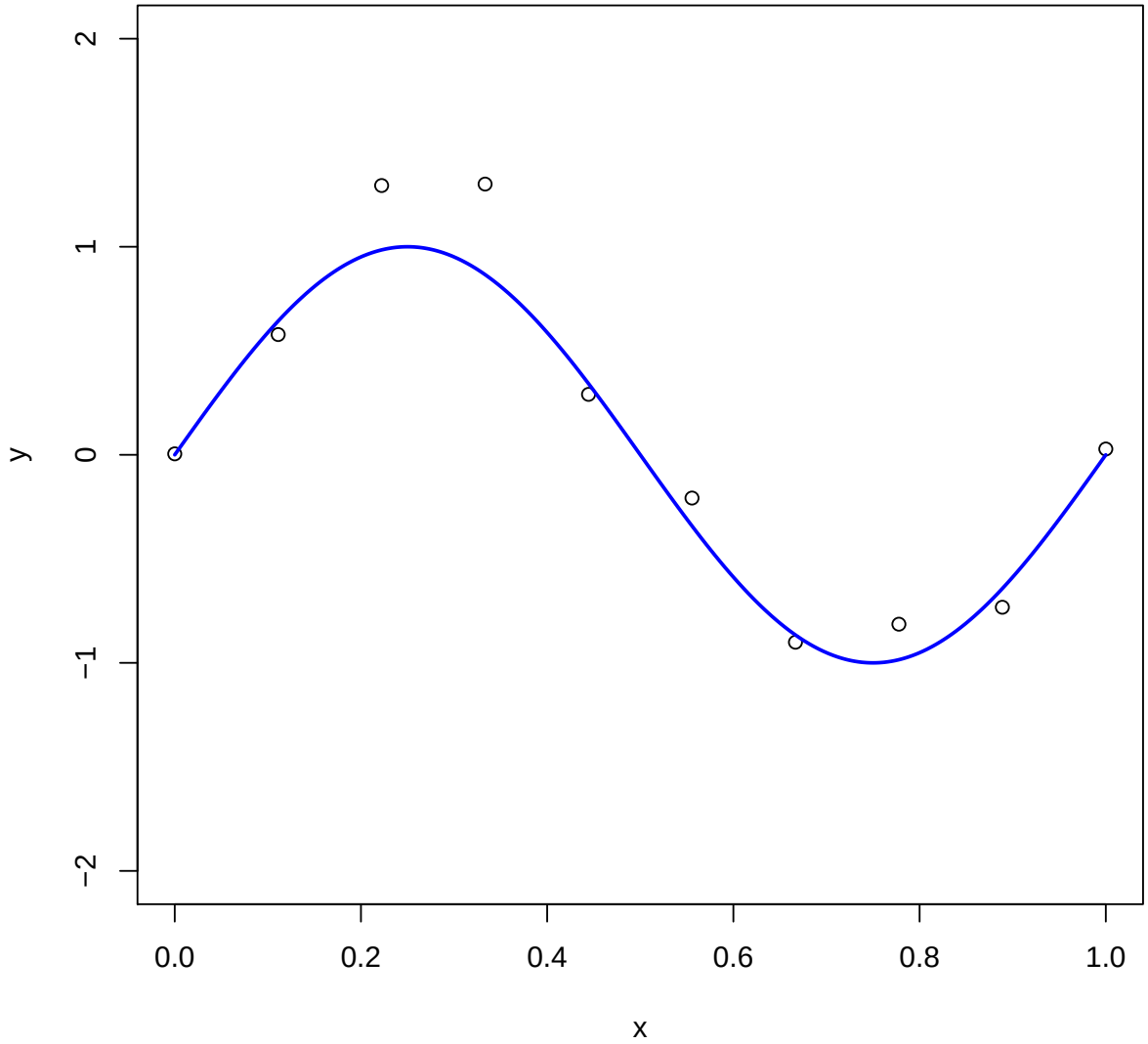
Ad Feelders

Universiteit Utrecht

Introduction

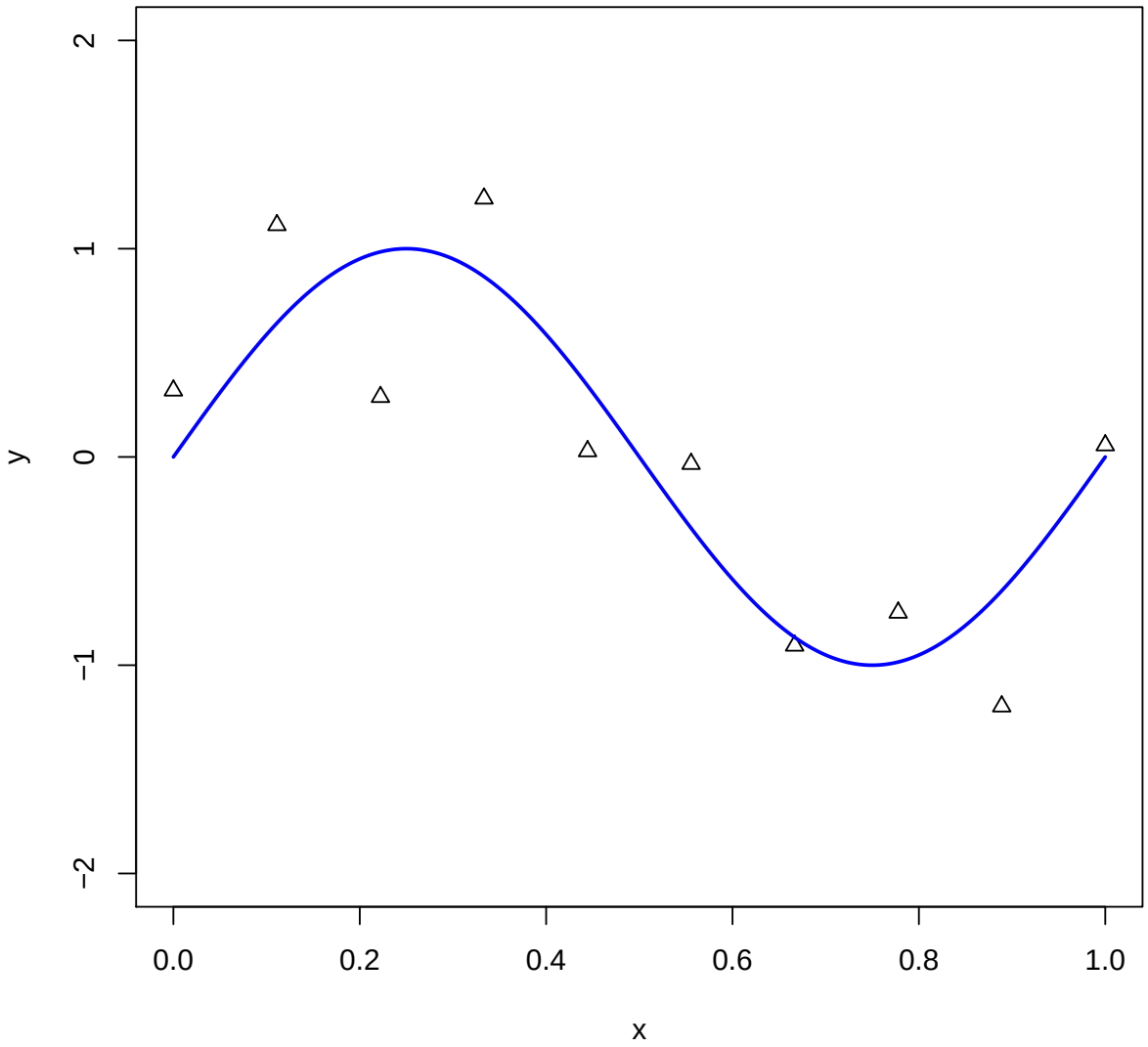
- Bad news: single trees are usually not among the top predictors.
- Good news: *ensembles* or *committees* of trees typically perform much better.
- Why does averaging the predictions of multiple trees help to reduce error?
- To answer this question we first study the bias-variance decomposition of prediction error.

Sample 1 and the True Regression Function



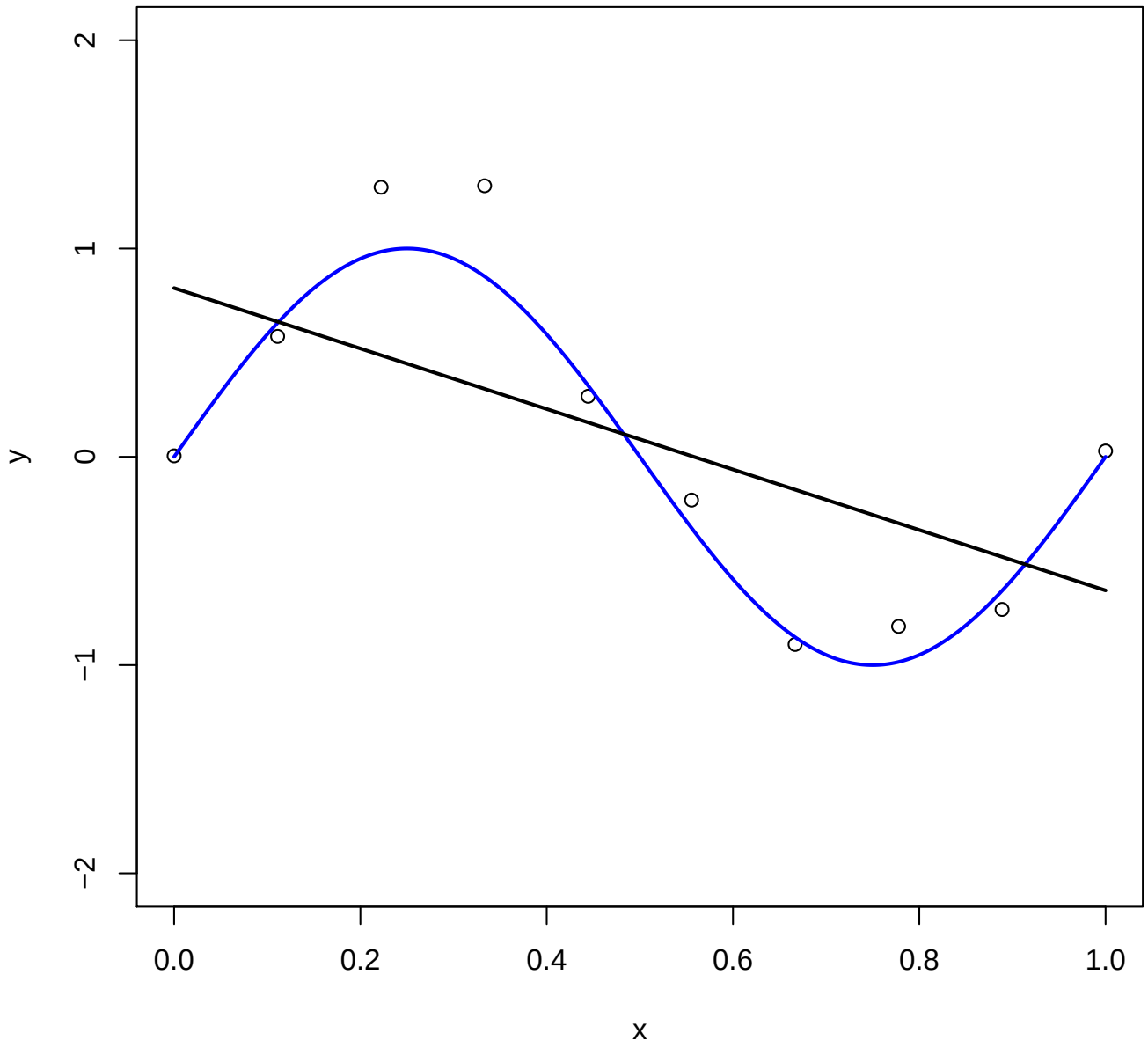
$f(x) = \sin(2\pi x)$ $\quad$ $y_i = f(x_i) + \varepsilon_i$ with $\varepsilon_i \sim N(0, \sigma = 0.3)$

Sample 2 and the True Regression Function

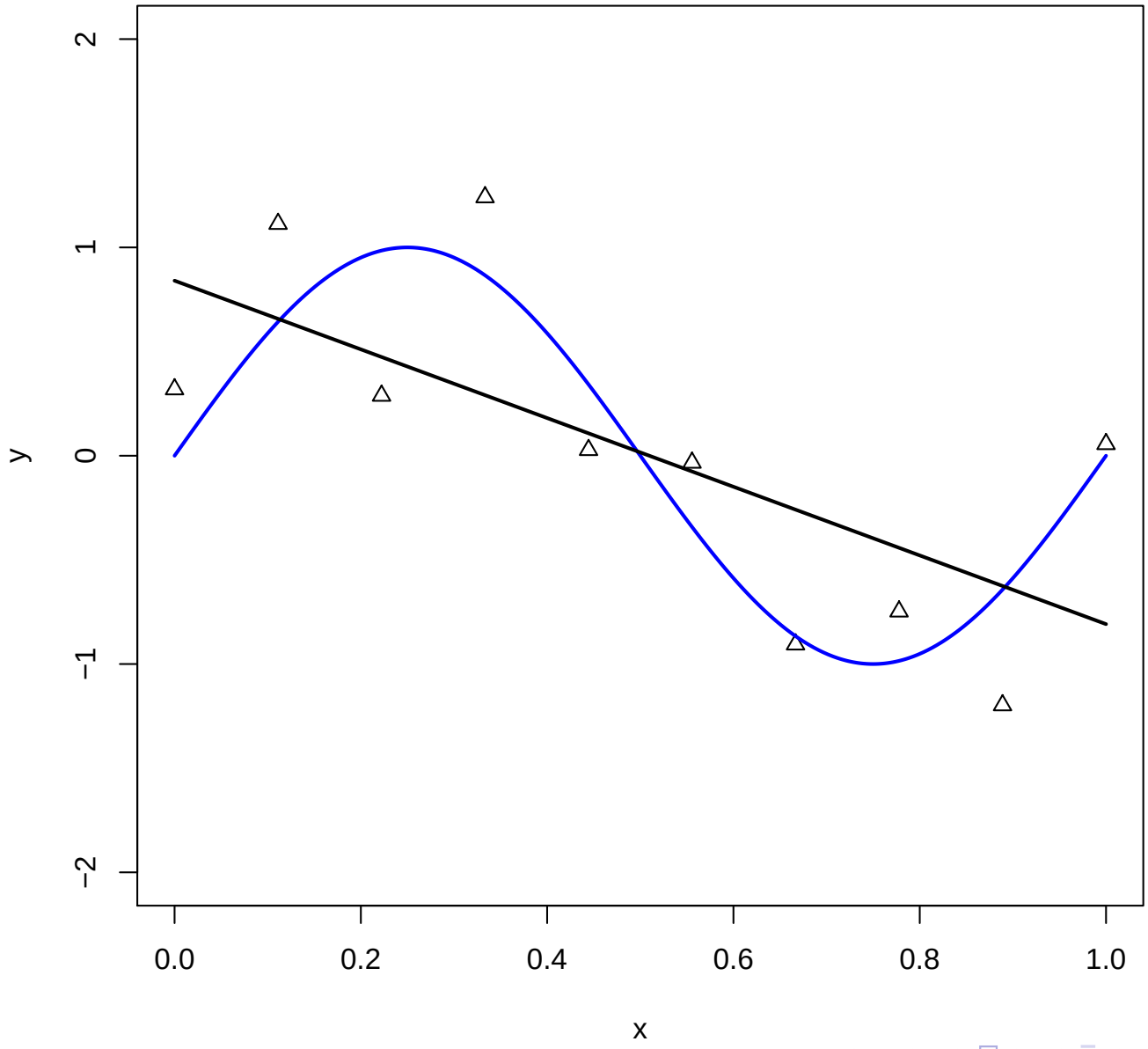


$f(x) = \sin(2\pi x)$ $\qquad$ $y_i = f(x_i) + \varepsilon_i$ with $\varepsilon_i \sim N(0, \sigma = 0.3)$

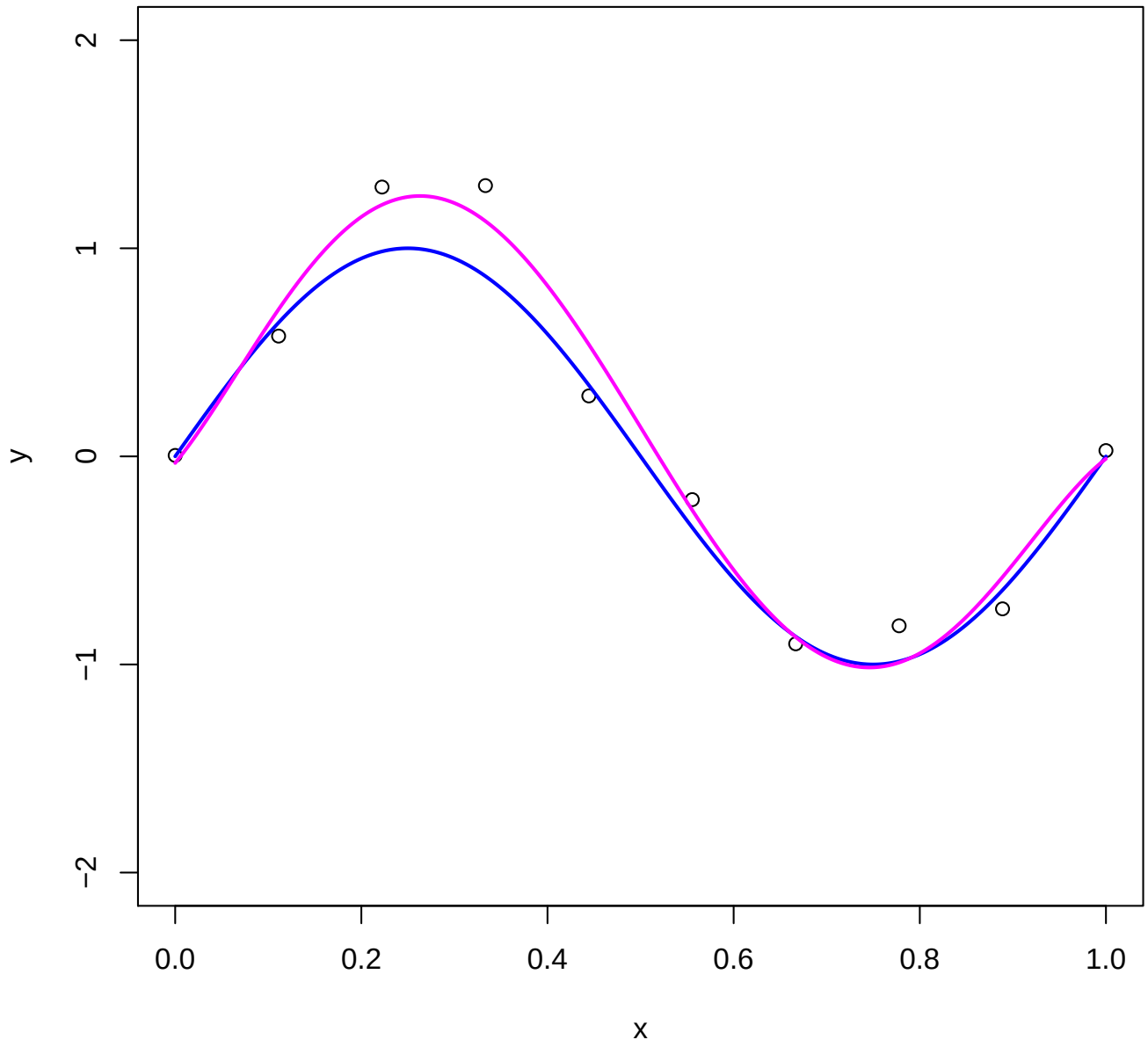
Linear Fit on Sample 1



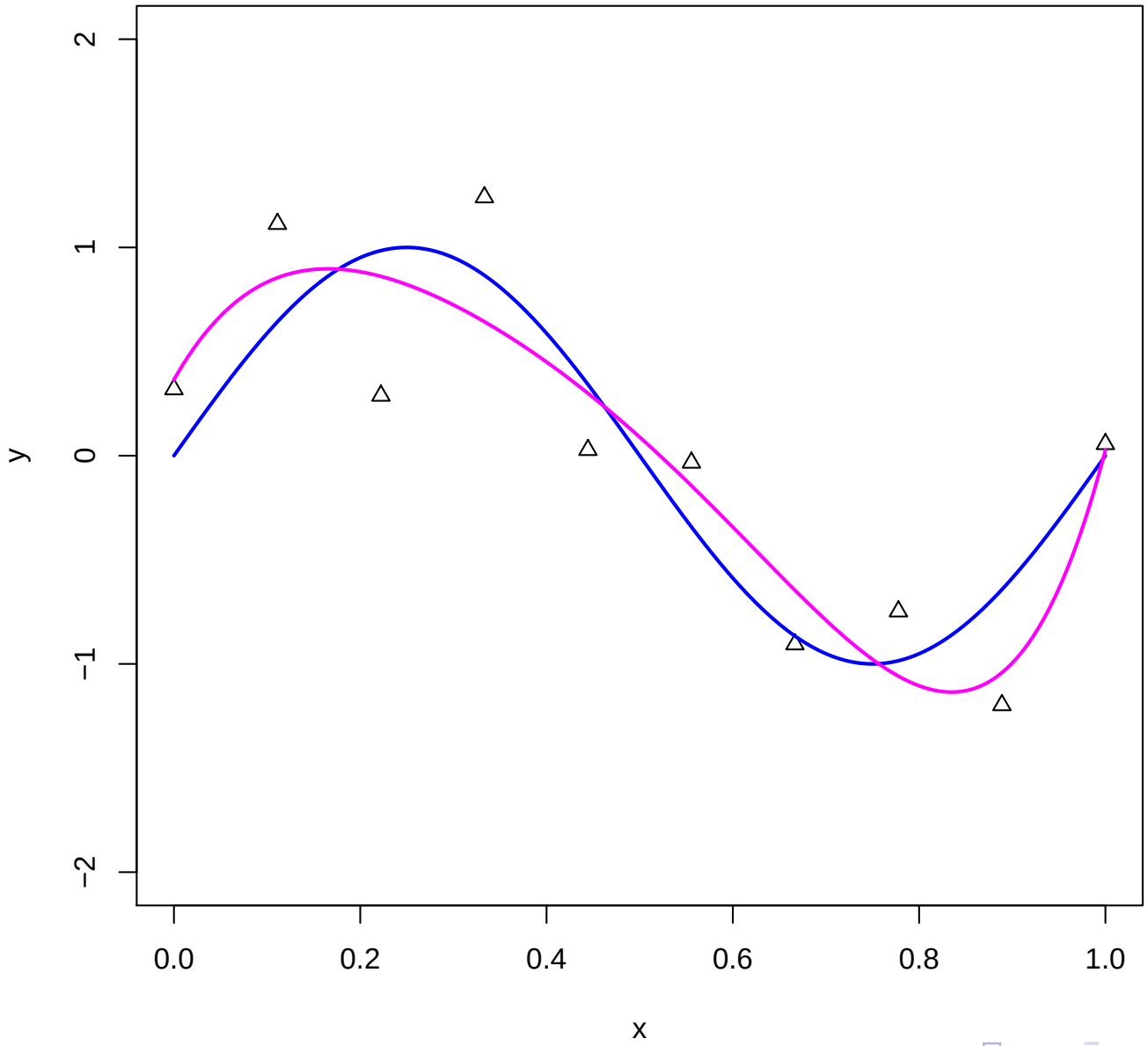
Linear Fit on Sample 2



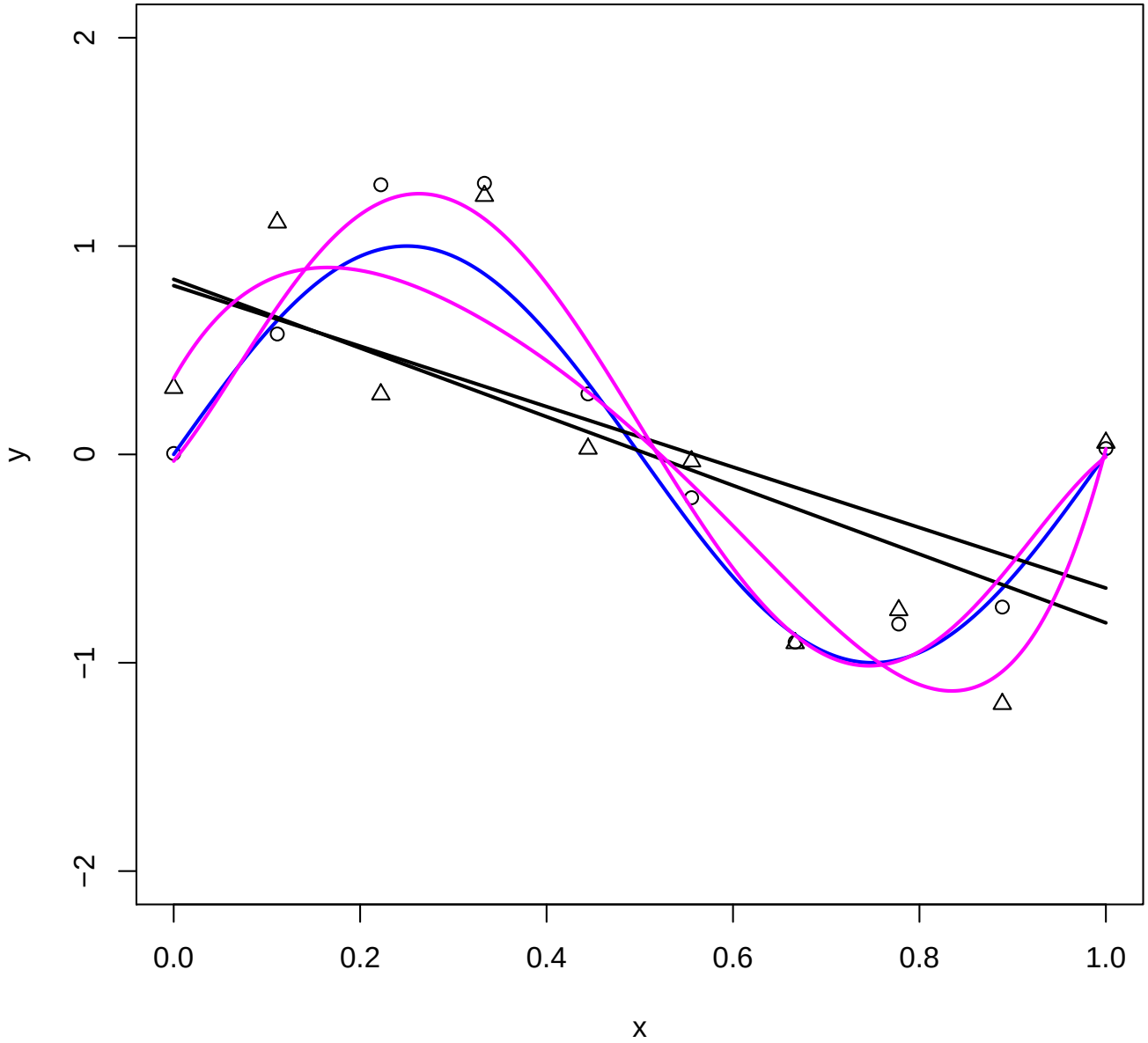
Polynomial of Degree 5 on Sample 1



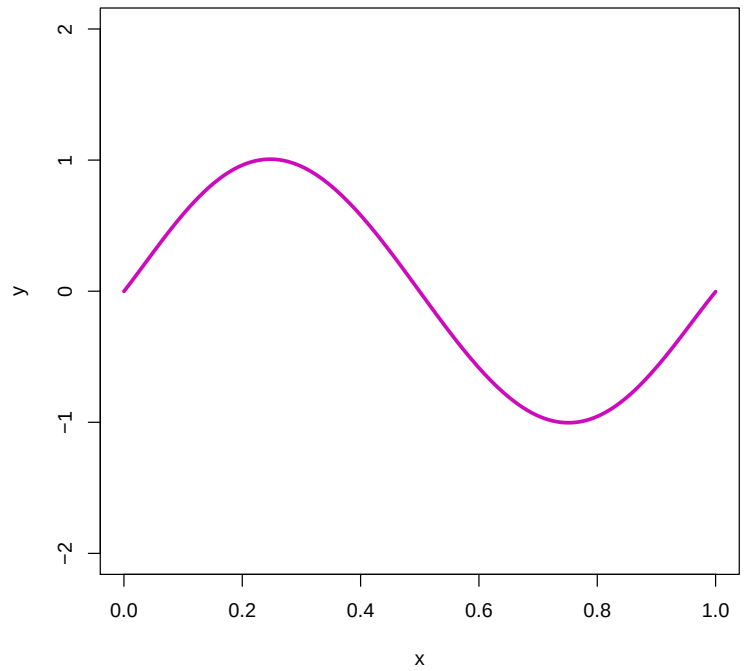
Polynomial of Degree 5 on Sample 2

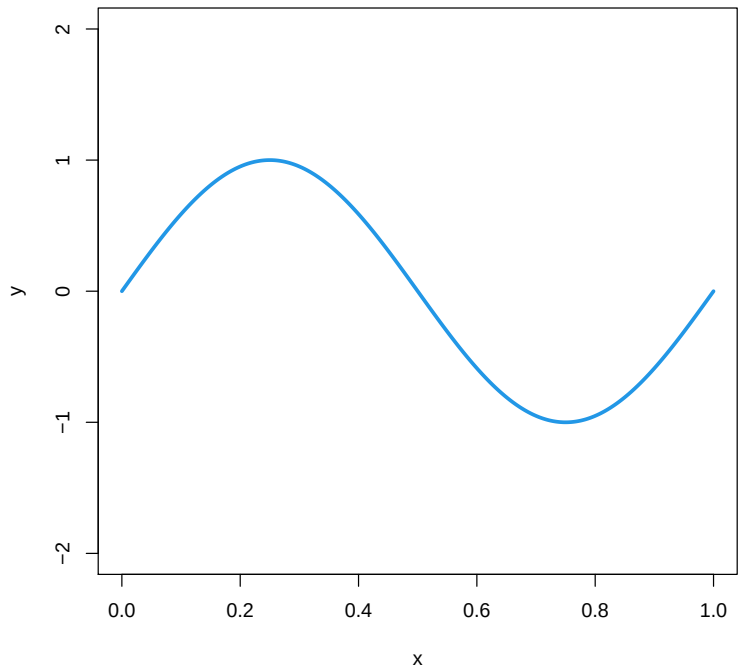


All Together Now



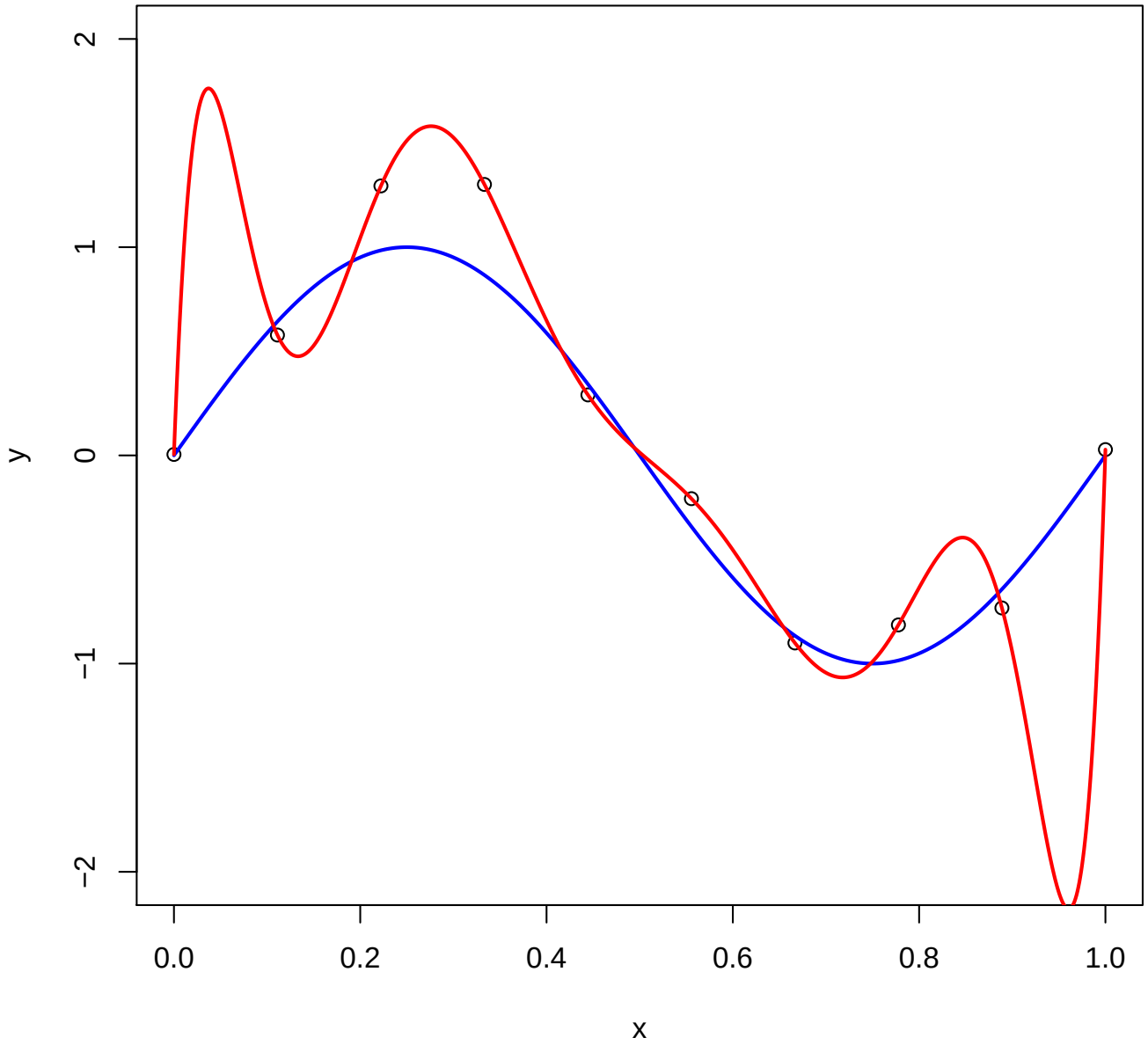
Average of degree 5 polynomials



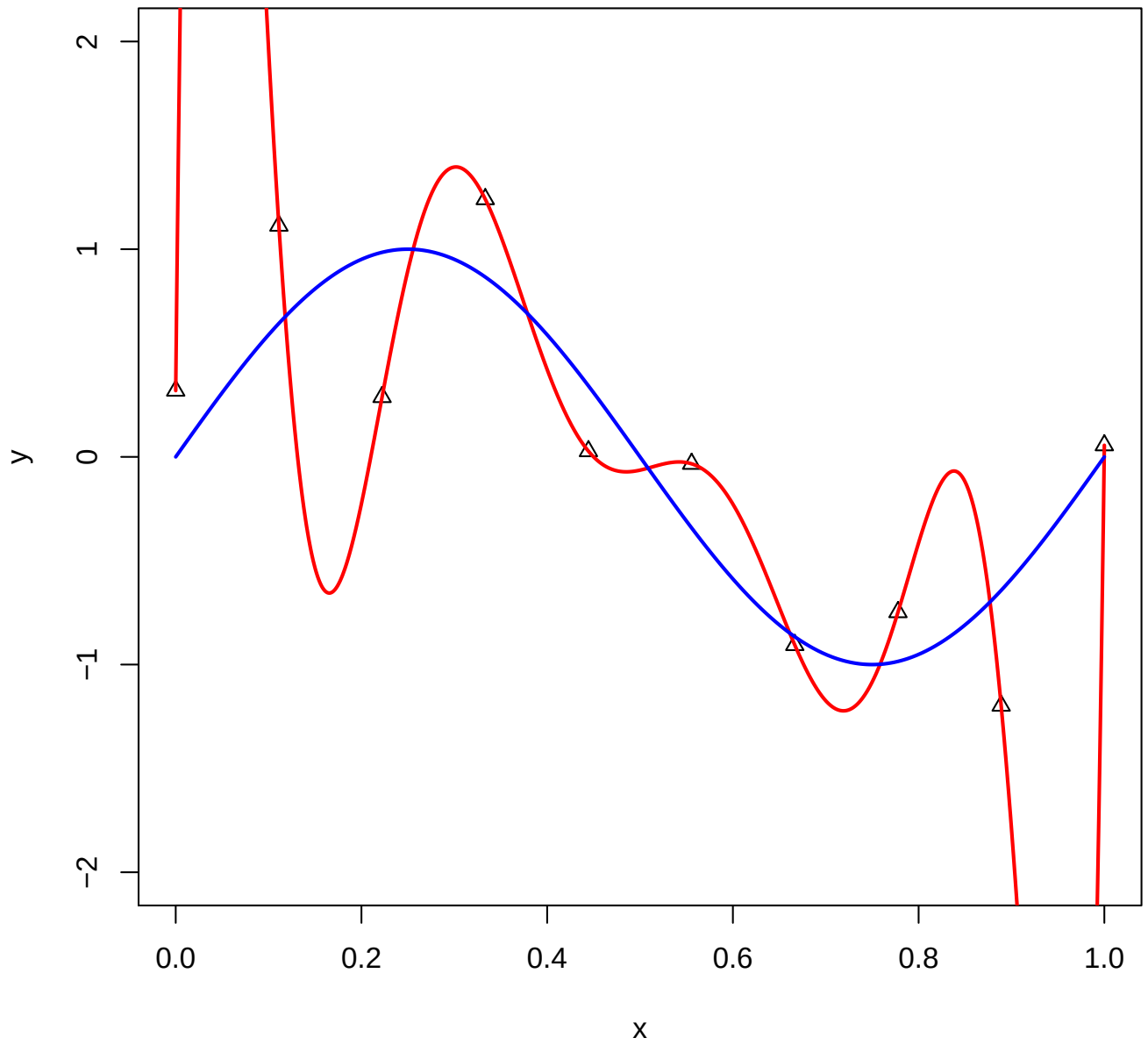


Average predictions of degree 5 polynomials estimated on 10,000 samples of size 10 (on the left), and $\sin(2\pi x)$ (on the right).
Can you see the difference?

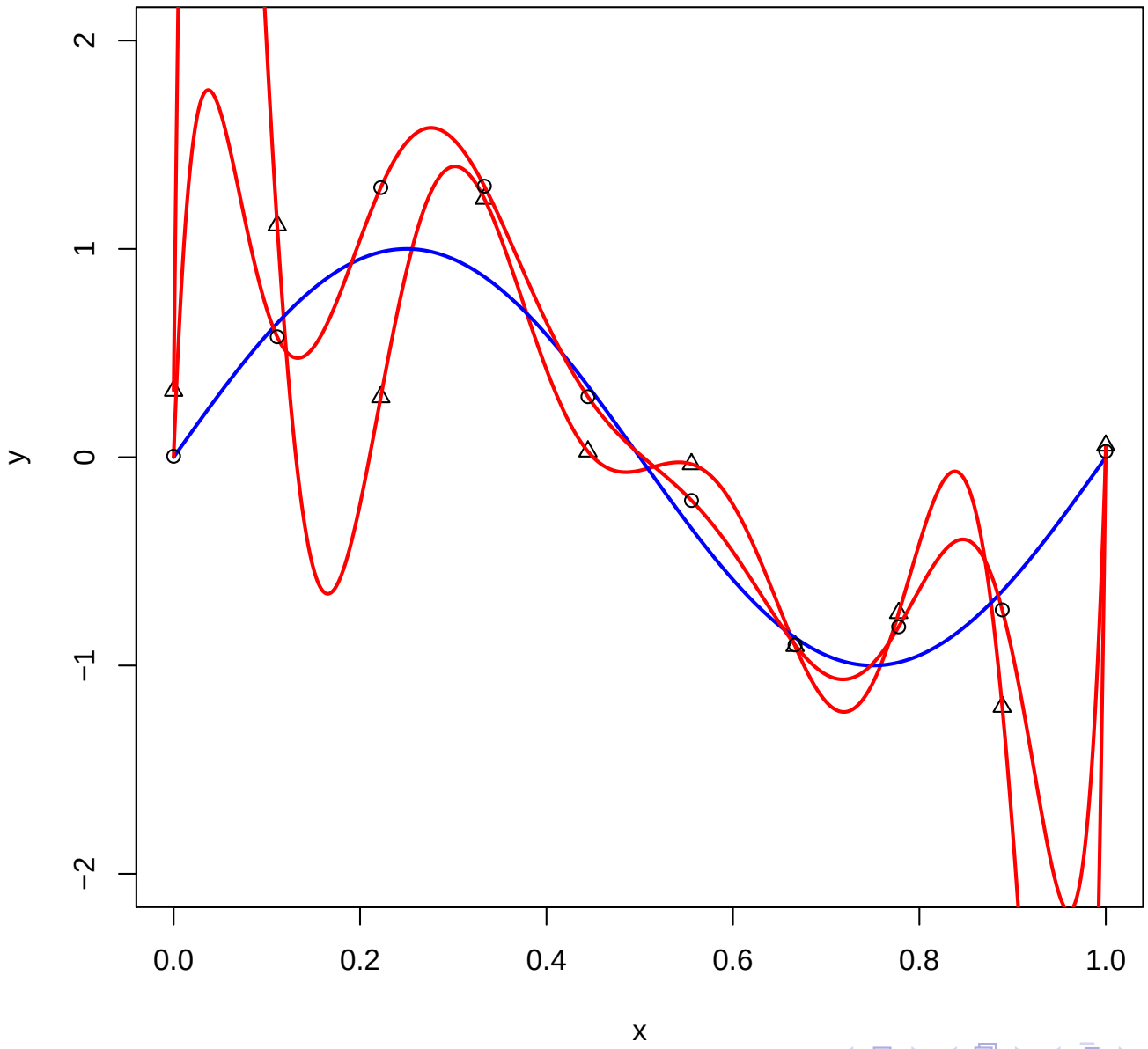
Polynomial of Degree 9 on Sample 1



Polynomial of Degree 9 on Sample 2



A High Variance Predictor!



Reducible and Irreducible Error

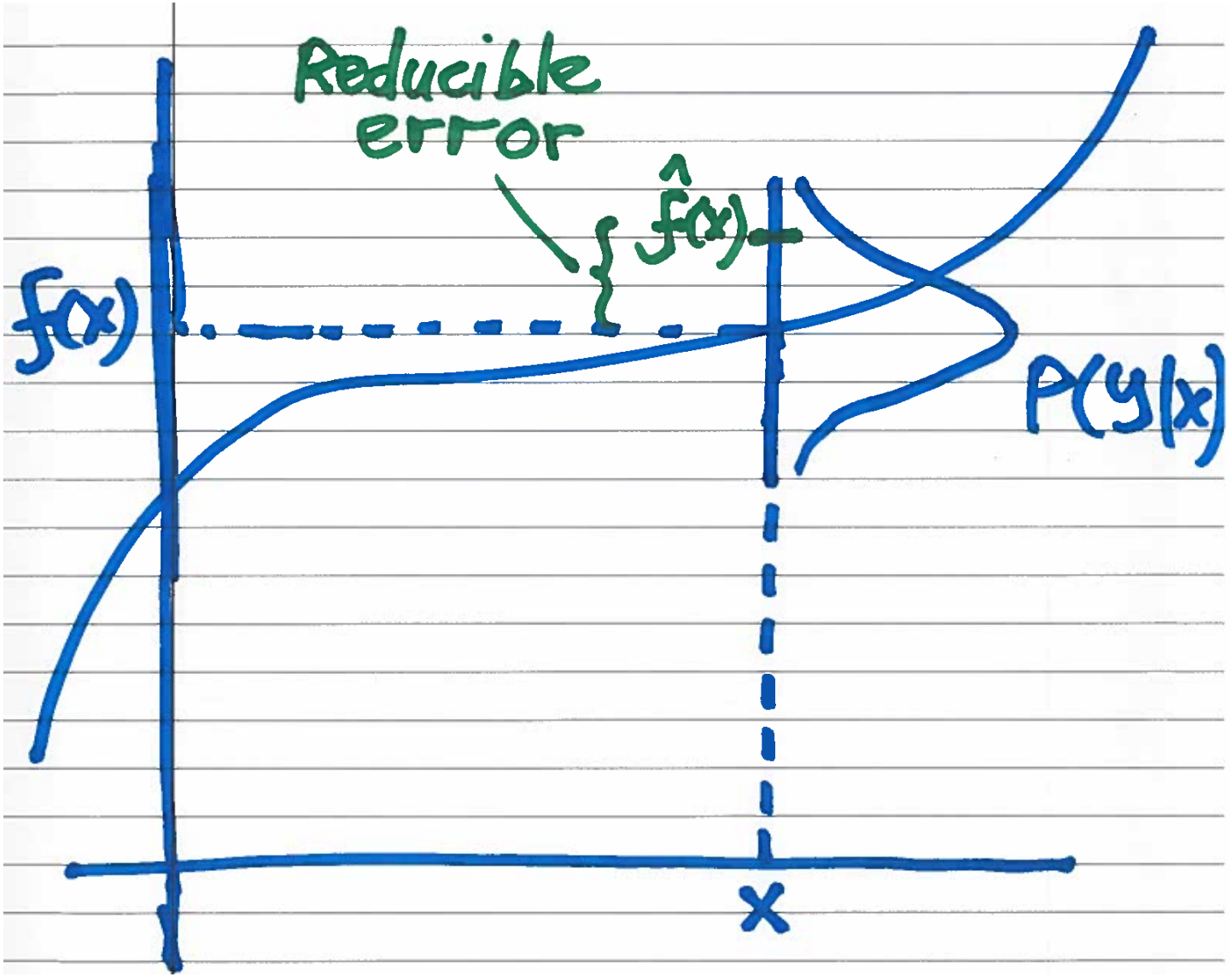
Consider a model created by an algorithm on a single data set, sampled from some population. What is the mean squared prediction error of this model on the population? For simplicity, we restrict our analysis to some fixed point x.

$$\mathbb{E}_{P(Y \mid x)}\left[(Y - \hat{f}(x))^2\right] = (f(x) - \hat{f}(x))^2 \qquad \text{(Reducible Error)}$$
$$+ \mathbb{E}_{P(Y \mid x)}\left[(Y - f(x))^2\right], \qquad \text{(Irreducible Error)}$$

where $f(x) \equiv \mathbb{E}[Y \mid x]$ is the true (population) regression function, and the expectation is taken with respect to $P(Y \mid x)$. $\hat{f}(x)$ is the model prediction at x.

The irreducible error is the error of the best possible prediction (which is $f(x) \equiv \mathbb{E}[Y \mid x]$), and is equal to the variance of Y around the regression line at the point x.

Reducible and Irreducible Error



Bias-Variance Decomposition of Estimation Error

Now let's focus on reducible error.

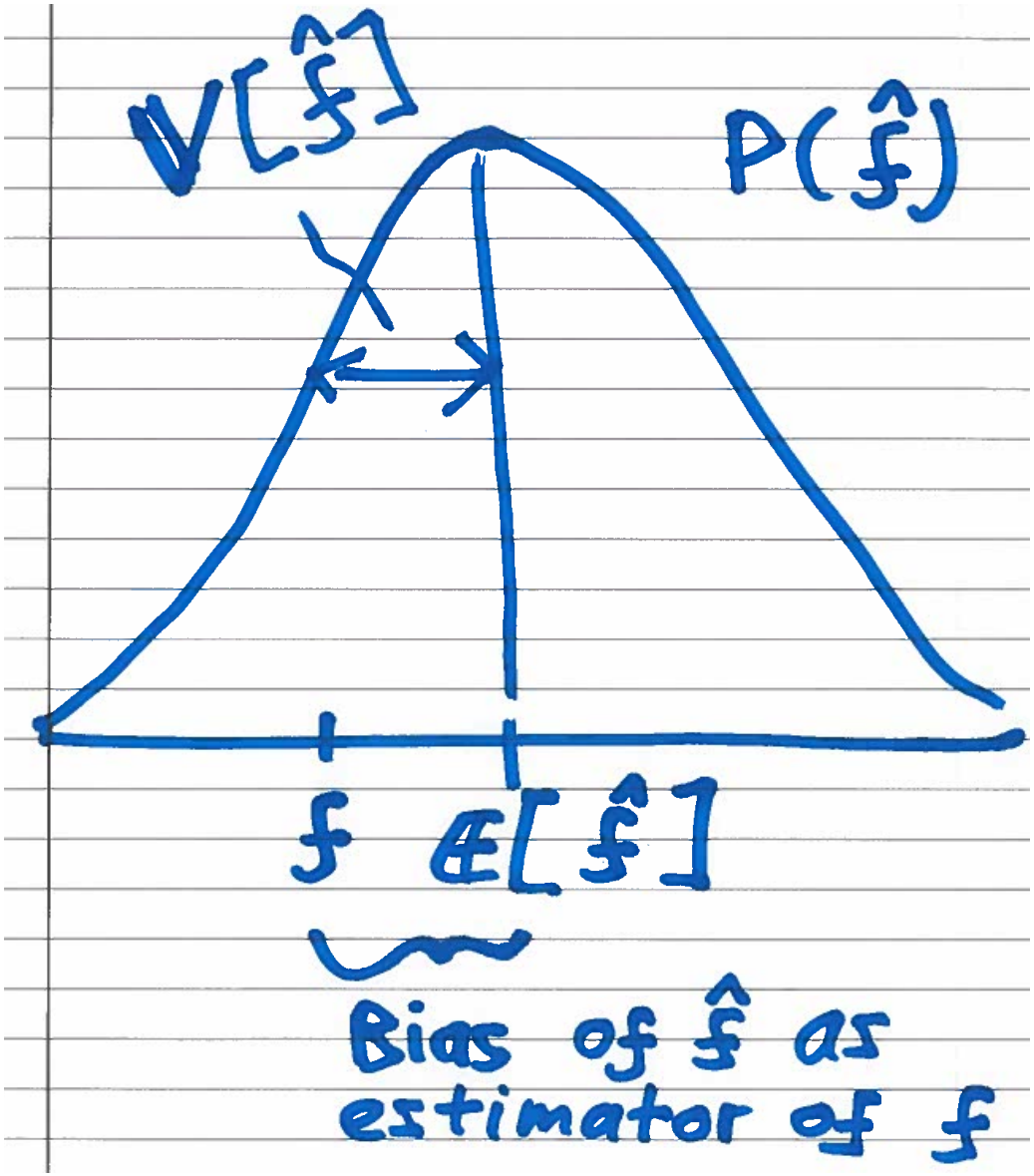
Consider the collection of models created by an algorithm on different data sets, sampled from the same population. We use each sample to estimate $f(x)$. What is the mean squared estimation error of these models?

$$\mathbb{E}_D\left[(f(x) - \hat{f}(x))^2\right] = (f(x) - \mathbb{E}_D[\hat{f}(x)])^2 \qquad \text{(Squared Bias)}$$
$$+ \mathbb{E}_D\left[(\hat{f}(x) - \mathbb{E}_D[\hat{f}(x)])^2\right] \qquad \text{(Variance)}$$

where expectation is taken with respect to repeated sampling from the same population.

$$\text{Mean Squared Error} = \text{Squared Bias} + \text{Variance}$$

Bias and Variance of $\hat{f}$ as estimator of f



Bias-Variance Decomposition

Some observations:

- Simple (inflexible) models tend to have high bias and low variance.
- Complex (flexible) models tend to have low bias and high variance.
- As sample size increases, variance goes down, but bias doesn't.
- Hence, we can afford to fit more complex models
  if the data set is large.
- In practice, we have to find the right trade-off between bias and variance in order to get small prediction error.

Reducing Variance by Bagging

- Classification trees are high variance classifiers.
- Variance can be reduced by averaging.
- Average how?
- Bootstrapping!

Reducing Variance by Bagging

Bagging is short for Bootstrap Aggregating.

Training:

1. Draw a sample *with replacement* from the training set.
   The sample should be of the same size as the training set.
2. Grow a tree on this bootstrap sample (pruning not necessary).
3. Repeat these steps M times to create M different trees.

Prediction:

1. Predict on a test sample using each of the M trees in turn.
2. Take the majority vote of the M predictions as the final prediction.

Reducing Variance by Bagging

For regression problems, generate M bootstrap samples, and combine the predictions by averaging:

$$\hat{f}_{\mathrm{BAG}}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x),$$

where $\hat{f}_m(x)$ is the prediction of the model trained on the $m-th$ bootstrap sample.

Notice that we never actually average *models*, we average their *predictions*.

Reducing Variance by Bagging

The true regression function is $f(x)$, so the output of each model can be written as the true value plus an error term in the form:

$$\hat{f}_m(x) = f(x) + e_m(x)$$
$$e_m(x) = \hat{f}_m(x) - f(x)$$

The expected squared error of the $m-th$ model then becomes

$$\mathbb{E}_{P(x)}\left[(\hat{f}_m(x) - f(x))^2\right] = \mathbb{E}_{P(x)}\left[e_m(x)^2\right],$$

where the expectation is taken with respect to the distribution of x.

Note: the models $\hat{f}_m$ are fixed now!

Reducing Variance by Bagging

The average error made by the models acting individually is therefore:

$$E_{\mathrm{AV}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{P(x)} \left[e_m(x)^2 \right]$$

Similarly, the expected error of the committee is given by

$$E_{\mathrm{BAG}} = \mathbb{E}_{P(x)} \left[\left(\frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x) - f(x) \right)^2 \right]$$

$$= \mathbb{E}_{P(x)} \left[\left(\frac{1}{M} \sum_{m=1}^{M} e_m(x) \right)^2 \right] = \frac{1}{M^2} \mathbb{E}_{P(x)} \left[\left(\sum_{m=1}^{M} e_m(x) \right)^2 \right]$$

Reducing Variance by Bagging

If we assume that the errors have zero mean and are uncorrelated, so that:

$$\mathbb{E}_{P(x)}\left[e_m(x)e_n(x)\right] = 0, \text{ for all } m \neq n,$$

then we obtain

$$E_{\mathrm{BAG}} = \frac{1}{M^2}\sum_{m=1}^{M}\mathbb{E}_{P(x)}\left[e_m(x)^2\right] = \frac{1}{M}E_{\mathrm{AV}}$$

This is a sensational reduction!

In practice, the errors of individual models tend to be positively correlated, and the reduction in overall error tends to be much smaller than suggested by this formula.

Random Forests

- Random forests can be regarded as an attempt to "de-correlate" the predictions of the individual trees, so

$$\mathbb{E}_{P(x)}\left[e_m(x)e_n(x)\right]$$

  is closer to zero.
- Each time we have to determine the best split in a node, we first randomly select a subset of the features.
- The size of this subset is a hyper-parameter of the random forest algorithm.
- We then determine the best split on this random subset of features, and perform that split.
- Otherwise, the procedure is identical to that described for bagging.

Boosting

- Bagging: fit a separate decision tree to each bootstrap sample, and combine all of the trees in order to create a single predictive model.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

Boosting Regression Trees

Algorithm 1 Boosting$((x_i, y_i)_{i=1}^N, B, \lambda, d)$

$\hat{f}(x) \leftarrow 0$ {Initialize model}
for i in $1, \ldots, N$ **do**
  $r_i \leftarrow y_i$ {Initialize residuals}
end for
for b in $1, \ldots, B$ **do**
  $\hat{f}_b(x) \leftarrow$ growRegressionTree$((x_i, r_i)_{i=1}^N, d)$
  $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$ {Update model}
  for i in $1, \ldots, N$ **do**
    $r_i \leftarrow r_i - \lambda \hat{f}_b(x_i)$ {Update residuals}
  end for
end for
return $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$ {Return final model}

Idea behind this procedure

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- We fit a decision tree to the residuals (errors) of the current model, and then add this new decision tree to the fitted function in order to decrease the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d (tree depth) in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Tuning parameters for boosting

- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Variable Importance Measure

- For bagged/RF *regression trees*, we record the total amount that the *RSS* is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF *classification trees*, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.