

# Machine Learning

## Blatt 4

Markus Vieth, David Klopp, Christian Stricker

17. Mai 2016



## Nr.1

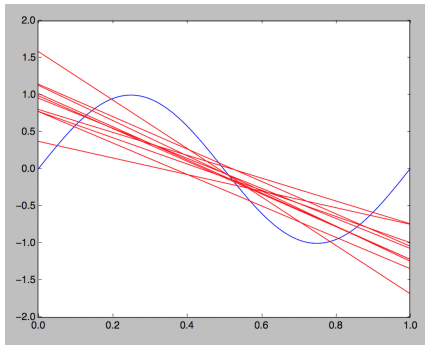


Abbildung 1: d=1

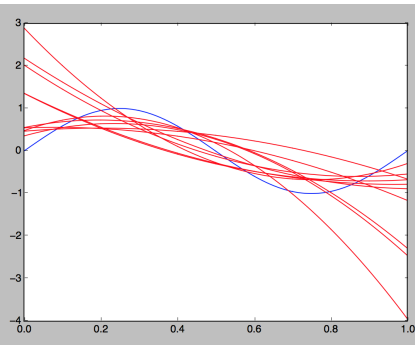


Abbildung 2: d=2

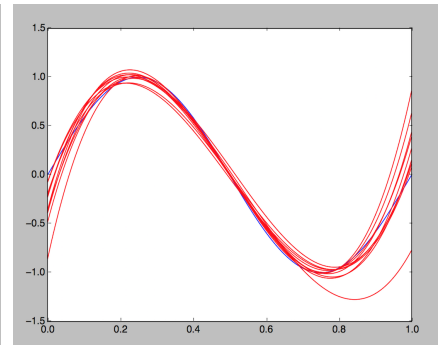


Abbildung 3: d=3

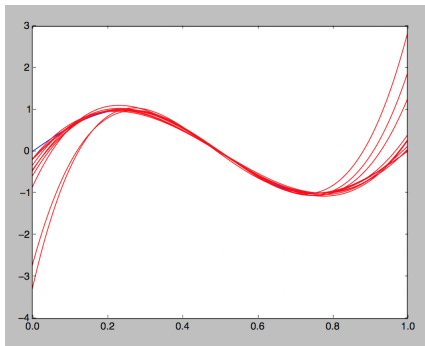


Abbildung 4: d=4

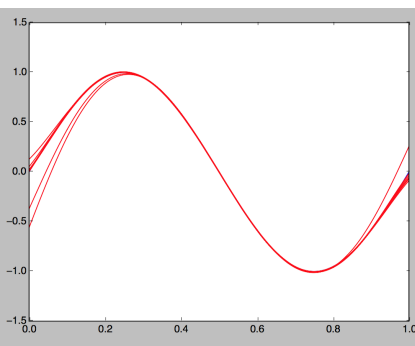


Abbildung 5: d=5

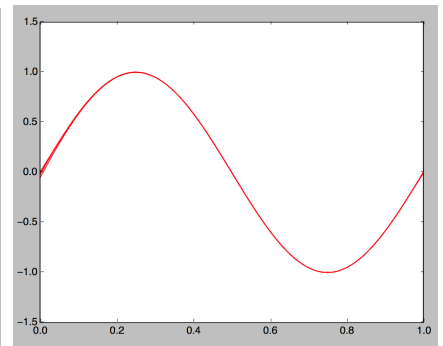


Abbildung 6: d=10

Die Grafiken verdeutlichen, dass der Bias zu Beginn sehr hoch ist, da der Sinus nicht durch eine Funktion ersten Grades akkurat angenähert werden kann. Je höher man nun den Wert  $d$ , also den Grad des Polynoms zum annähern wählt, desto präziser kann die Sinus-Funktion approximiert werden und desto kleiner wird der Bias. Die Varianz lässt sich in den angegebenen Abbildungen nur schwer erkennen. In Abbildung 6 im Intervall 0 bis 0.2, lässt sich eine leichte Streuung erkennen, die durch das Rauschen der Trainingsdaten verursacht worden sein könnte. Diese leichte Streuung kann allerdings auch durch den immer noch zu geringen Grad  $d$  des Polynoms entstanden sein.

## Nr.2

## Nr.3

```

1 // erzeugen eines Remove Filters, der die Attribute an den angegebenen Indices entfernt
2 Remove rm = new Remove();
3 rm.setAttributeIndices("1,4,6,last");
4 // erzeugen eines neuen untrainierten J48 Classifiers
5 // unpruned := versuche nicht den Baum zu vereinfachen => größerer Baum
6 J48 j48 = new J48();
7 j48.setUnpruned(true);
8 // erzeugen eines FilteredClassifier, der auf die Daten zuerst den angegebenen Filter anwendet, bevor
   der Algorithmus (J48) trainiert wird
9 FilteredClassifier fc = new FilteredClassifier();
10 fc.setFilter(rm);
11 fc.setClassifier(j48);

```

```

12 // erzeugen einer neuen CVParameterSelection, die Parameter auf Basis von cross-validation auswählt
13 CVParameterSelection cvp = new CVParameterSelection();
14 cvp.setClassifier(fc);
15 // C := confidence-Parameter 0.1 bis 0.5 mit einer Schrittweite von 5
16 String[] params = new String[]{"C", "0.1", "0.5", "5"};
17 cvp.setCVParams(params);
18 // trainiere den Classifier und schicke die Trainingsdaten zuerst durch den Filter
19 cvp.buildClassifier(dataset);
20 // treffe eine Vorhersage über die Klassenverteilung für die gegebenen Instanzen
21 // die Testdaten gelangen hierbei nicht durch den Filter
22 cvp.distributionForInstance(instance);

```

## Nr.4

## Nr.5

$$\begin{aligned}
 accuracy &= \frac{TP + TN}{TP + FP + FN + TN} \\
 &= \frac{TP}{P + N} + \frac{TN}{P + N} \\
 &= \frac{P \cdot TP}{P \cdot (P + N)} + \frac{N \cdot TN}{N \cdot (P + N)} \\
 &= \frac{TP}{P} \cdot \frac{P}{P + N} + \frac{TN}{N} \cdot \frac{N}{P + N} \\
 &= \frac{TP}{P} \cdot \frac{P}{P + N} + \frac{TN}{N} \cdot \left(1 - \frac{P}{P + N}\right)
 \end{aligned}$$

Mit  $A = \frac{P}{P+N}$  folgt:

$$accuracy = sensitivity \cdot A + specificity \cdot (1 - A)$$