Machine Learning
Blatt 4

Markus Vieth, David Klopp, Christian Stricker

19. Mai 2016

Nr.1

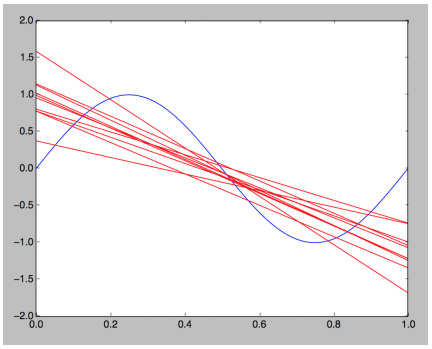

Abbildung 1: d=1

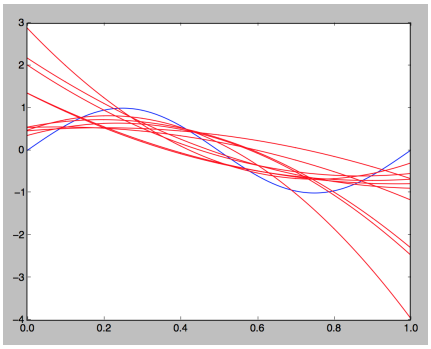

Abbildung 2: d=2

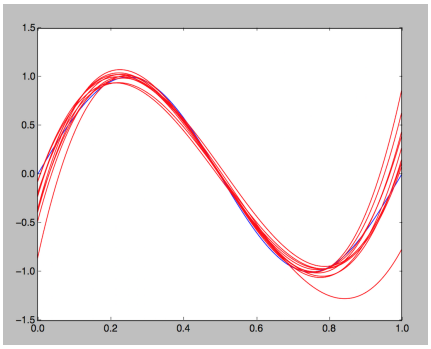

Abbildung 3: d=3

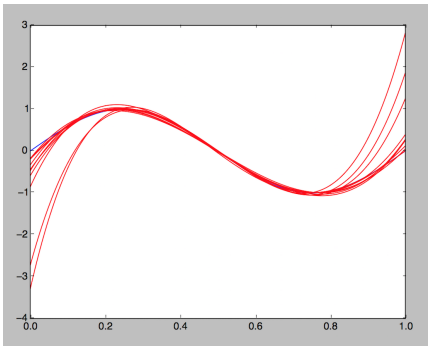

Abbildung 4: d=4

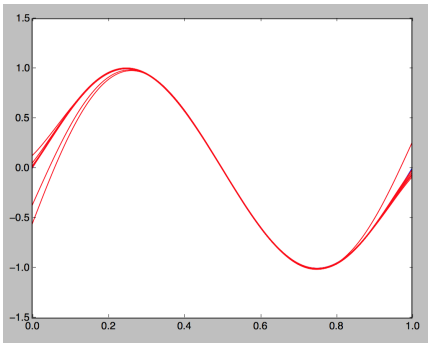

Abbildung 5: d=5

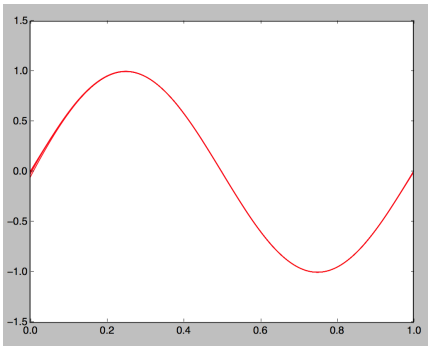

Abbildung 6: d=10

Die Grafiken verdeutlichen, dass der Bias zu Beginn sehr hoch ist, da der Sinus nicht durch eine Funktion ersten Gerades 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 Interval 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

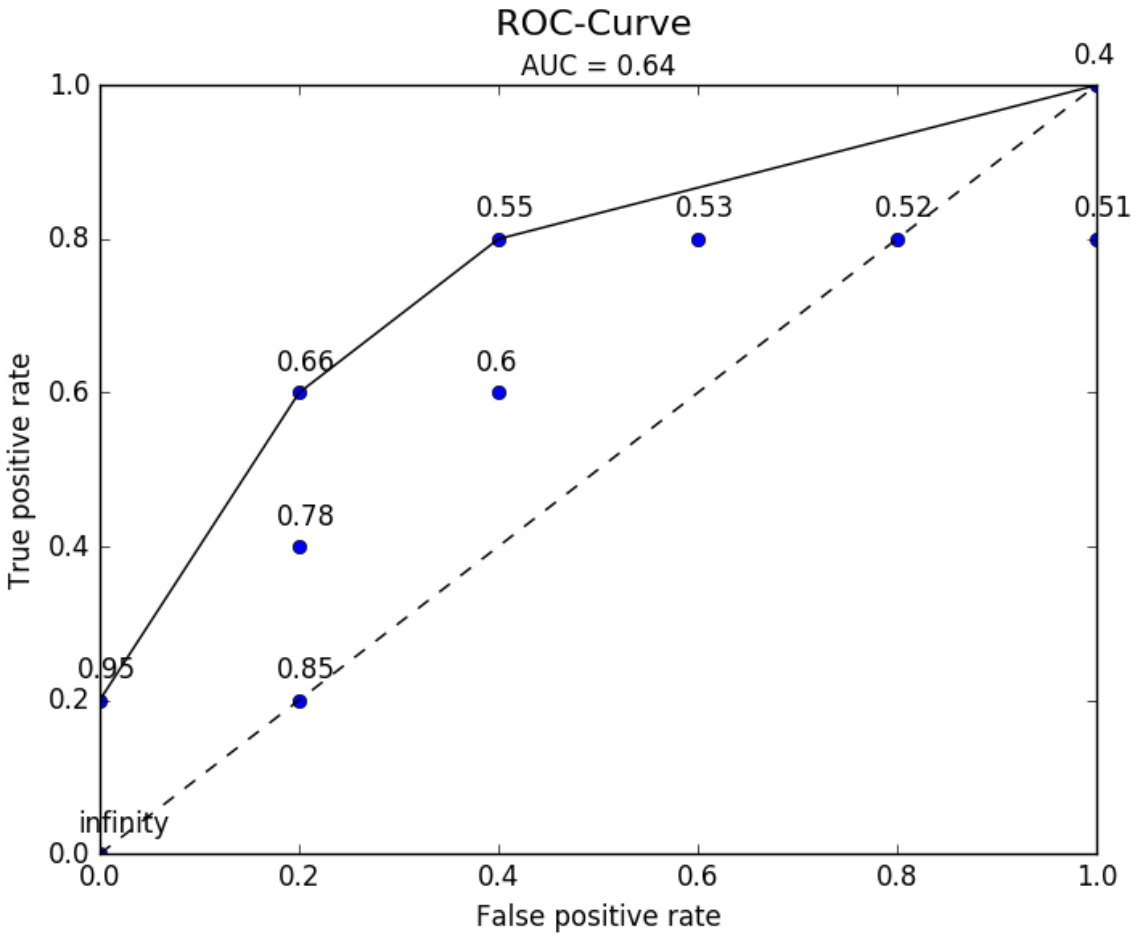

Abbildung 3: ROC-Curve with convex hull

Anhand des Graphen ist zu sehen, dass der Classifier besser ist, als einfaches, gleichverteiltes raten (gestrichelte Linie). Ebenfalls geht hervor, dass der Classifier eher richtig als falsch klassifiziert, da die Kurve fast konkav ist. Eine AUC von wie hier $0,64$ ist nicht besonders gut, da dies bedeutet, dass das die Wahrscheinlichkeit für ein zufälliges positives Beispiel aus dem Datensatz einen kleineren Score zu haben als für ein zufälliges negatives Beispiel bei über 35% liegt. Hier wäre ein höherer Wert wünschenswert. Im Allgemeinen handelt es sich hier um einen schlechten Classifier (vergleiche `http://mlwiki.org/index.php/ROC_Analysis`).

Nr.3

```
1  // erzeugen eines Remove Filters, der die Attribute an den angegebenen Indicies entfernt
2  Remove rm = new Remove();
3  rm.setAttributeIndices("1,4,6,last");
4  // erzuegen eines neuen untrainierten J48 Classifier
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 angegeben 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

Quellcode zu den Aufgaben aufgrund des Umfangs digital.

a)

Es wurden folgende Datensätze gewählt:

- splice.arff
- letter.arff
- kr-vs-kp.arff

b)

Die Vergleiche brachten folgende Ergebnisse (Bootstrap.635-Validierung):
File res/letter.arff
0 = a = RandomForest
1 = b = J48 as DecisionTree
letzte McNema Tabelle

	a falsch	a richtig
b falsch	244	64
b richtig	798	6288

avg. McNema Stat. over 10
577.539575643173

File res/kr-vs-kp.arff
0 = a = RandomForest
1 = b = J48 as DecisionTree
last McNema Table

	a falsch	a richtig
b falsch	2	6
b richtig	3	1128

avg. McNema Stat. over 10
2.1657428274340043

File res/splice.arff
0 = a = RandomForest
1 = b = J48 as DecisionTree
last McNema Table

	a falsch	a richtig
b falsch	32	66
b richtig	57	1016

avg. McNema Stat. over 10
2.354155198842717

c)

Wie in den Vergleichen deutlich wird, lässt sich keine Allgemeine Aussage treffen. In 1 von 3 Fällen, war der RandomForest deutlich besser, während er in den anderen beiden Datensätzen sogar (unwesentlich) schlechter war als der DecisionTree.

d)

Bei dem durchgeführten Experiment wurde klar, dass RandomForest nicht immer besser ist als ein DecisionTree. Die auffälligsten Unterschieden zwischen den Datensätzen 2 und 3 zu 1 ist, dass 1 im Vergleich viele Klassen besitzt. Auch die Anzahl der vorhandenen Instanzen ist um ein vielfaches Größer, was mich zu der Annahme verleitet, dass RandomForest auf einem begrenzten Datensatz seine Stärken zeigt, jedoch ein DecisionTree bei ausreichend vielen Trainingsdaten besser klassifizieren kann, weil dann zu ausreichend vielen Attributen eine Abhängigkeit erstellt werden kann (unter der Voraussetzung, dass die vorhandenen Attribute einen kausalen Zusammenhang zur Klasse haben).

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:

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