MLE Übung 2

Knn Classifier using KFold cross validation

Allgemein Ich habe für diese Übung den Advanced Path mit Zusatzaufgabe gewählt. Nach Absprache mit den Professoren ist es mir auch erlaubt worden die Übung in Haskell zu machen.

Nachdem die Datensätze in verschiedenen Formaten gespeichert sind habe ich sie zuerst zu einem Format geändert und dieses für die verwendung der Library vorausgesetzt. (CSV ohne quotation; ',' als seperator und die letzte column ist das label)

Für die Implementation des Knn ist ein KdTree verwendet worden. Dafür habe ich die Haskell Library `kdt` verwendet.

Programm verwendung

```
kfold [OPTIONS] [FILE]
  KFolf Knn classification

Common flags:
  -f --folds[=INT]
  -k[=INT]
  -? --help Display help message
```

Iris data

n=15		predicted setosa	predicted versicolor	predicted virginica	Total	Recall
actual	setosa	**5.0**	*0.0*	*0.0*	5.0	1
	versicolor	*0.0*	**4.9**	0.1	5.0	0.98
	virginica	*0.0*	0.5	**4.5**	5.0	0.9
	Total	5.0	5.4	4.6		
	Precesion	1.00	0.91	0.98	**Acc:**	**0.96**

Parameter: k=5 folds=10

Die Irisblumen Daten eignen sich hervorragend zum Klassifizieren mittels Knn-Algorithmus.

Wine data

Im folgenden abschnitt ist immer vom `winequality-red.csv` die Rede.

		predicted							
		3	4	5	6	7	8	Total	Recall
actual	3	**0.00**	0.09	0.45	0.36	*0.00*	*0.00*	0.90	0.00
	4	*0.00*	**0.09**	2.36	2.09	0.27	*0.00*	4.81	0.02
	5	0.09	0.45	**39.63**	20.09	1.54	0.09	61.89	0.64
	6	*0.00*	0.09	18.72	**31.63**	7.00	0.54	57.98	0.55
	7	*0.00*	0.09	2.54	8.81	**6.36**	0.27	18.07	0.35
	8	*0.00*	*0.00*	*0.00*	0.91	0.72	**0.00**	1.63	0.00
	Total	0.09	0.81	63.70	63.89	15.89	0.90		
	Precisior	0.00	0.11	0.62	0.50	0.40	0.00	**acc:**	**0.53**

Wie in der obigen Abbildung zu sehen ist liefert der Knn hier nur ein mässiges Ergebniss. Dies kann auch darauf zurückzuführen sein, dass es im Datensatz kaum Vertreter der äußeren Klassen gibt und fast alle Daten der Klasse `5` und `6` ensprechen.

Dies kann man deutlicher veranschaulichen, wenn man die Aufgabe leicht ändert und nur mehr Klassifizieren will ob der Wein schlecht(3,4), mittel(5,6) oder gut(7,8) ist.

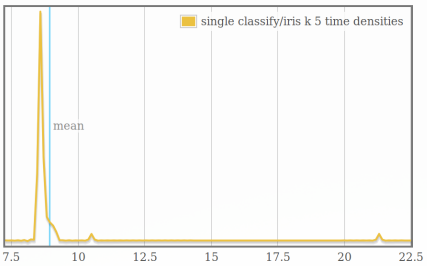
		predicted							
		3	4	5	6	7	8	Total	Recall
actual	3	**0.18**		5.26		0.27		5.71	0.03
	4								
	5	0.63		**110.07**		9.17		119.87	0.92
	6								
	7	0.09		12.26		**7.35**		19.70	0.37
	8								
	Total	0.90		127.59		16.79			
	Precision	0.20		0.86		0.44		**acc:**	**0.81**

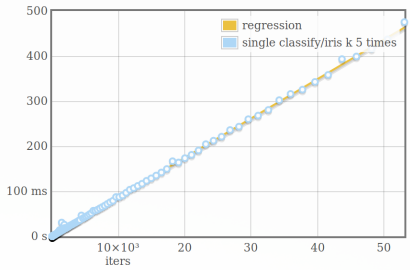
Hier wird eine Accuracy von über 80% erzeugt, obwohl nur die mittlere Klasse hohe Precision und Recall aufweist.

Performance

Um die Performance zu testen habe ich die Haskell Microbenchmark Library `criterion` verwendet. Aufgrund der hohen Lazyness von Haskell Programmen kann es sehr schwer sein Benchmarks zu schreiben, da es nicht nur reicht etwas auszurechnen, da der Compiler aggressiv Funktionen und Datenstrukur-teile die nicht verwendet werden gar nicht berechnet. `Cirterion` vereinfacht dies indem es sich darum kümmert Datenstrukturen vertiefend zu evaluieren und außerdem einen statistische Auswertung liefert. ##### Single Classify Benchmark

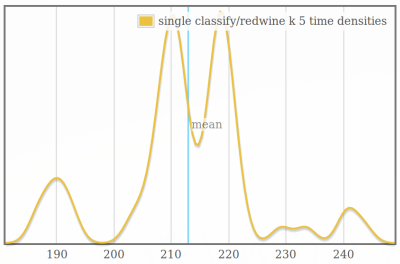
single classify/iris k 5

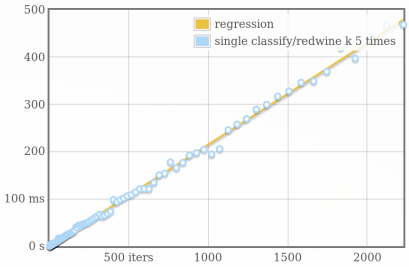




	lower bound	estimate	upper bound
OLS regression	8.65 µs	8.72 µs	8.79 µs
R² goodness-of-fit	0.999	0.999	1.000
Mean execution time	8.69 µs	8.94 µs	9.83 µs
Standard deviation	149 ns	1.68 µs	3.59 µs

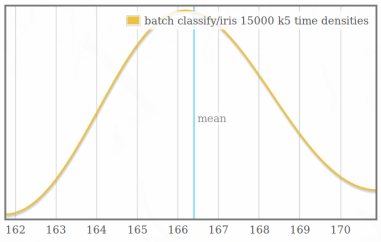
single classify/redwine k 5

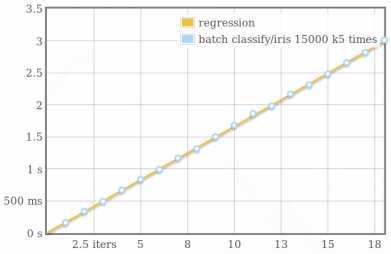




	lower bound	estimate	upper bound
OLS regression	212 µs	215 µs	218 µs
R² goodness-of-fit	0.996	0.998	0.999
Mean execution time	210 µs	213 µs	216 µs
Standard deviation	9.78 µs	12.0 µs	15.1 µs

batch classify/iris 15000 k5



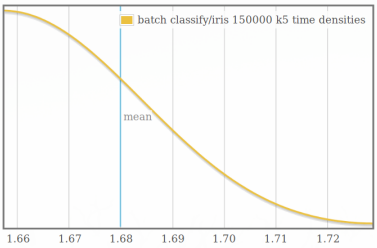


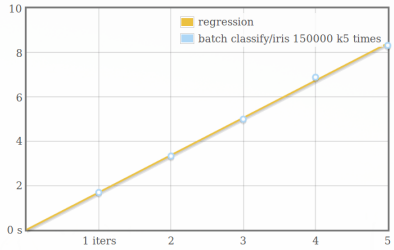
	lower bound	estimate	upper bound
OLS regression	165 ms	**166 ms**	167 ms
R² goodness-of-fit	0.999	**1.000**	1.000
Mean execution time	166 ms	**166 ms**	167 ms
Standard deviation	1.38 ms	**1.88 ms**	2.64 ms

Outlying measurements have slight (5.2%) effect on estimated standard deviation.

Batch Benchmarks

batch classify/iris 150000 k5

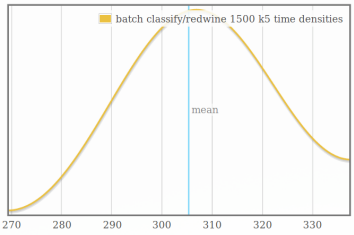


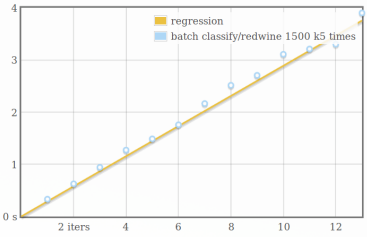


	lower bound	estimate	upper bound
OLS regression	1.63 s	**1.68 s**	1.78 s
R² goodness-of-fit	0.992	**0.999**	1.000
Mean execution time	1.67 s	**1.68 s**	1.71 s
Standard deviation	875 µs	**25.3 ms**	32.5 ms

Outlying measurements have moderate (16.0%) effect on estimated standard deviation.

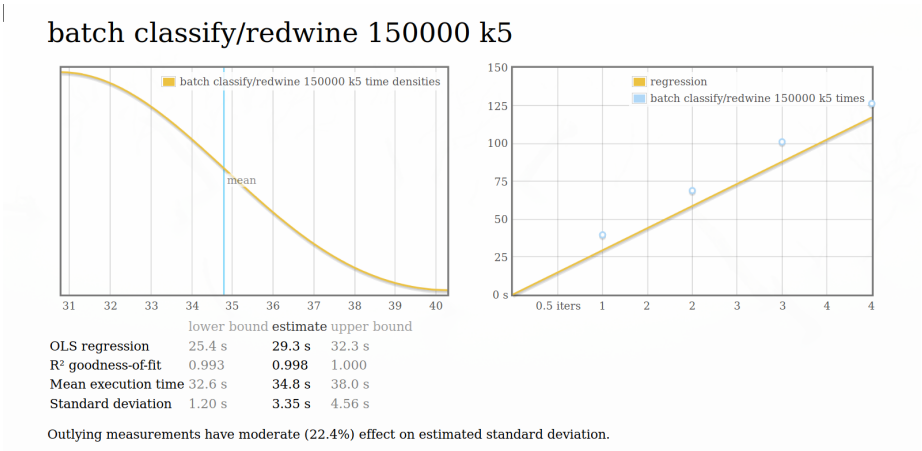
batch classify/redwine 1500 k5





	lower bound	estimate	upper bound
OLS regression	270 ms	**289 ms**	305 ms
R² goodness-of-fit	0.979	**0.993**	0.999
Mean execution time	298 ms	**305 ms**	312 ms
Standard deviation	9.96 ms	**14.4 ms**	21.2 ms

Outlying measurements have moderate (14.1%) effect on estimated standard deviation.



Weitere benchmark Ergebnisse:

benchmarking single classify/iris k 1 time 1.802 µs (1.756 s .. 1.846 s) 0.995 R² (0.992 R² .. 0.997 R²) mean 1.823 s (1.777 s .. 1.910 s) std dev 195.1 ns (126.4 ns .. 310.5 ns)

benchmarking single classify/iris k 5 time 8.710 s (8.650 s .. 8.806 s) 0.998 R² (0.995 R² .. 1.000 R²) mean 8.859 s (8.733 s .. 9.203 s) std dev 614.9 ns (251.7 ns .. 1.250 s)

benchmarking single classify/iris k 10 time 11.99 s (11.89 s .. 12.11 s)S 0.999 R² (0.998 R² .. 1.000 R²) mean 11.97 s (11.88 s .. 12.14 s) std dev 391.1 ns (274.0 ns .. 573.4 ns)

benchmarking single classify/redwine k 1 time 3.456 s (3.386 s .. 3.528 s) 0.997 R² (0.995 R² .. 0.999 R²) mean 3.401 s (3.350 s .. 3.472 s) std dev 202.8 ns (143.3 ns .. 263.9 ns)

benchmarking single classify/redwine k 5 time 210.6 s (202.1 s .. 217.1 s) 0.990 R² (0.984 R² .. 0.996 R²) mean 213.3 s (207.4 s .. 218.2 s) std dev 17.22 s (13.93 s .. 21.35 s)

benchmarking single classify/redwine k 10 time 267.8 s (254.8 s .. 279.8 s) 0.991 R² (0.986 R² .. 0.997 R²) mean 264.2 s (258.2 s .. 269.7 s) std dev 19.47 s (16.86 s .. 22.51 s)