Classification Problems

One of the hot topics for investors recently is the Alibaba IPO, which has brought back the long existing puzzles about IPO. That is typical IPO return from offering price to the price when market starts trading is about 10%, while in long run (2-3 years) IPO shares seems underperformed. For some individual investors, they cannot get the offering price and probably care more about the short term returns (like in 6 month). Our first classification task is to answer a specific question: if investor bought IPO stock on first day, is it worth to hold the share till 6 month if say we are at 60 days after IPO. There could be many factors that determine the price after 6 month, and there are number of reasons make it especially hard to predict performance for IPO stock. First there is very limited public info available for IPO Company, secondly there is not much of historical price data for analysis, and lastly IPO usually attracts many speculators which further increase the uncertainty around IPO stock.

The data has 6 attributes as below

- 1_day(first day price change (%) over offering price)
- 30_day(30 day price change (%) over offering price)
- 60_day(60 day price change (%) over offering price)
- Month(month of IPO)
- Underwriter (underwriter group)
- Grade1(class : T—price is higher than first day closing , F—Price is lower than first day closing)

The data is extracted from two sources and compiled together. The IPO stock performance is from
NASDAQ(http://www.nasdaq.com/markets/ipos/performance.aspx?timeframe=6+months)
Data was extracted on 9/22, the underwriter info is from on same date
http://www.123jump.com/ipo/ipo_recent/4/1/0

The data include IPO from 2013/09 to 2014/03, total 130 instance, which is t separated to training set (104) and test set (26), we will refer the problem as “IPO”

The second classification problem is to determine if a person makes more than $50K a year based on census data. We like to see if any of the factors has or has no impact on how much a person earns, we think that attributes such as age, education years, marital status, hours working per week may have biggest impact, and we also like to know if something fixed when we born contribute, such as race, sex, native country. There must be some “noise” factor in the attributes, which can be a good test for overfitting for different algorithms we try. And t the dataset has more than trivial amount of instance and attributes, which makes it good candidate for our exercise.

The data is from UCI's mach8ine learning repository
http://archive.ics.uci.edu/ml/datasets/Adult
The data used for our purpose was resampled from the original dataset to a smaller size that contains only 1627 instance, then it is separated to training set(1302) and test set(325). We'll refer this problem as "Adult".

Methodology

We ran five different supervised learning algorithm on the dataset, including

- Decision tree with pruning (C4.5 implemented as J48 in Weka), information gain is used to split attributes
- Neural Network(with single layer and 3 nodes implemented as MultiLayerPerceptron in WeKa)
- Boosting version of same decision tree above (with AdaBoostM1 implemented in WeKa)
- Support Vector Machine(with three different Kernel functions implemented as wrapper class LibSVM in WeKa)
- K-nearest Neighbors (for K=2,5,10, implemented as IBK in WeKa)

as

Our training and test set are created by split one whole dataset, so the test set instances are total unknown to learning model. The training set is further resampled with replacement at 90%-10% to created training set of different size.

In the next section we will discuss the result for reach algorithm, considering performance metrics such as error rate for training and testing, and also in some cases, running time if it is noticeable. As baseline rate that we can compare to, we also reported the result with ZeroR in Weka (it essentially assign all instances to one class using majority rules)

Performance by Algorithms

1. Baseline error rate
   We used ZeroR in Weka get the base line error rate, it is just error rate when we assign all instance to one class.

problem\sample size	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	testing set
IPO	30%	50%	45.16%	43.90%	46.15%	45.16%	43.06%	44.58%	46.24%	44.23%	34.62%
Adult	18.46%	21.15%	24.36%	23.85%	23.66%	23.56%	23.71%	24.50%	23.83%	23.20%	20.92%

2. Decision Tree

First we tried J48 with default parameter settings, the performance for each problem is shown as figure below, we can see from both that a clear down trend as we increase the training set size until we reach best performance for both training and testing, then the error rates start increasing as result of overfitting. The jump at 60% size most likely be result of the particular characteristic of that training set.

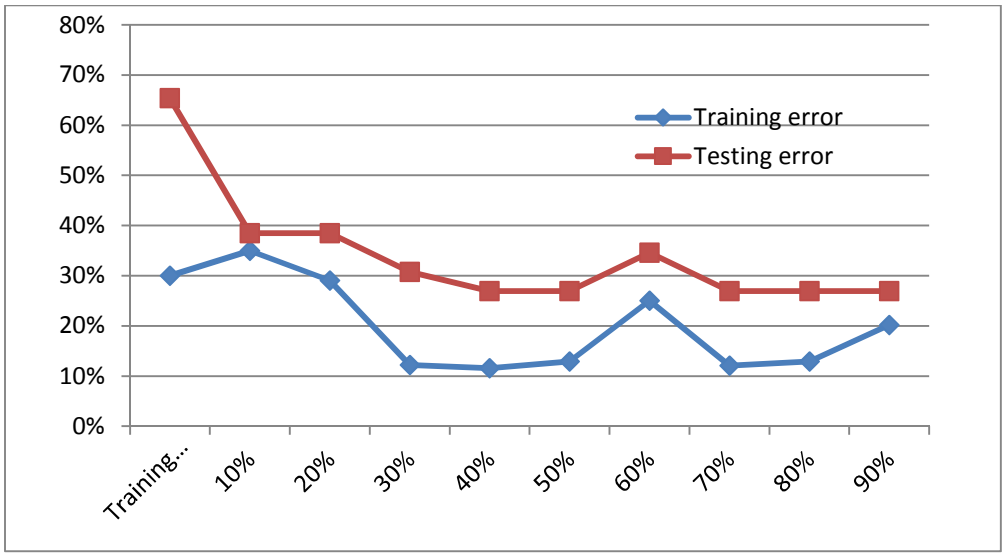

Figure 1. For IPO Problem--J48 in WEKA with default parameter

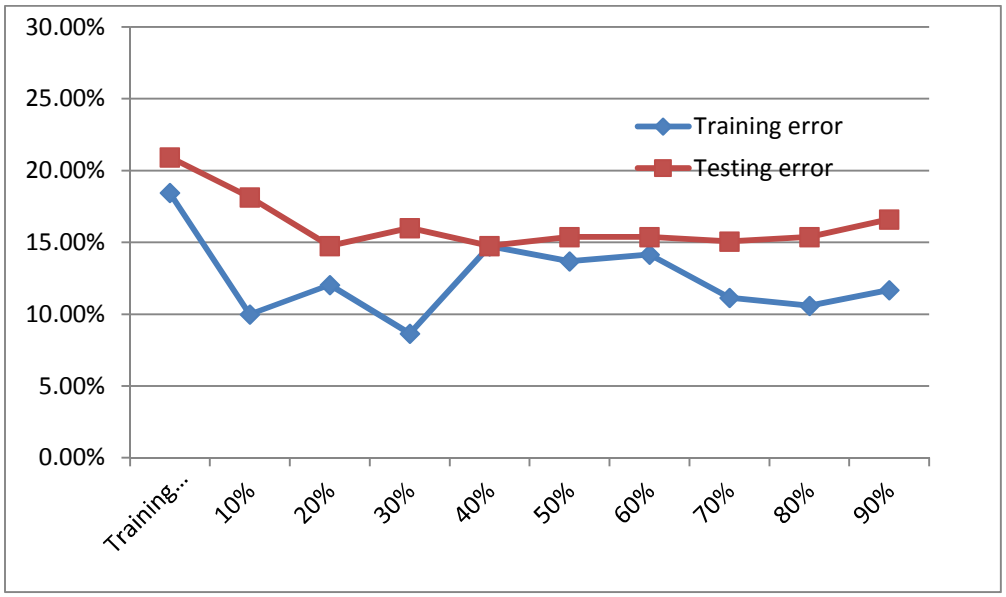

Figure 2. For Adult Problem--J48 in WEKA with default parameter

Next, we explore two different parameter settings, one is unpruned J48, the other prune more aggressively (decrease confidence factor to 0.2 from 0.25). The result is show below . As we can see for IPO problem, the tree size is limited b/c of the very few

attribute in this dataset, this could be the main driving force that we could not see further improvement on error rate. while for "Adult" problem, the aggressive pruning generally produces much simple model yet the performance on testing and training is quite stable and good.

IPO	Unpruned			Default(c 0.25)			Aggressive(c=0.2)		
Size	training error	testing error	Tree Size	training error	testing error	Tree Size	training error	testing error	Tree Size
100%	18.27%	26.92%	24	20.19%	26.92%	18	32.69%	30.77%	3
90%	10.75%	26.92%	28	12.90%	26.92%	22	12.90%	26.92%	22
80%	10.84%	26.92%	28	12.05%	26.92%	24	12.05%	26.92%	24
70%	8.33%	30.77%	33	25%	34.62%	7	25%	34.62%	7
60%	12.90%	26.92%	20	12.90%	26.92%	20	12.90%	26.92%	20
50%	11.54%	26.92%	18	11.54%	26.92%	18	30.77%	38.46%	3
40%	12.20%	30.77%	16	12.20%	30.77%	16	34.15%	38.46%	3
30%	19.35%	30.77%	12	29.03%	38.46%	3	29.03%	38.46%	3
20%	25%	30.77%	12	35%	38.46%	3	35%	38.46%	3
10%	20%	53.85%	10	30%	65.38%	1	30%	65.38%	1

Table 1 For IPO problem – Performance for three setting of J48

Adult	Unpruned			Default(c 0.25)			Aggressive(c=0.2)		
Size	training error	testing error	Tree Size	training error	testing error	Tree Size	training error	testing error	Tree Size
100%	5.61%	19.08%	585	11.67%	16.62%	63	13.75%	14.46%	18
90%	5.72%	19.38%	451	10.59%	15.38%	87	11.44%	15.69%	58
80%	5.76%	17.54%	407	11.14%	15.08%	58	11.14%	15.08%	58
70%	5.60%	18.15%	452	14.16%	15.38%	20	14.60%	15.38%	12
60%	5.12%	17.54%	381	13.70%	15.38%	20	14.21%	15.38%	12
50%	5.22%	18.46%	431	14.75%	14.77%	11	14.75%	14.77%	11
40%	6.54%	16.62%	224	8.65%	16%	100	10.77%	17.23%	42
30%	4.62%	17.54%	198	12.05%	14.77%	56	12.05%	14.77%	56
20%	4.23%	21.54%	240	10%	18.15%	83	12.31%	16%	58
10%	3.08%	20.92%	54	18.46%	20.92%	1	18%	20.92%	1

Table 2 For IPO problem – Performance for three setting of J48

Next we try cross validation to see if further improvement can be achieved, the 10 fold cross validation performed on default setting, which gets 15.9% training error rate 16.1% testing error rate for "Adult" problem, and 20.2% training error rate and 26.9 % testing error rate for "IPO" problem. We can see for "Adult " problem, it possible reduces the overfitting(as we see the higher training error rate) while still producing same level of performance, but for "IPO", there is no further improvement as overfitting is not the culprit here.

3. Boosting with decision tree

Boosting can work with decision tree to further improve the performance and we can afford to use much aggressive pruning technique. We used the AdaBoostM1 (with iteration step=10) implemented in WeKa applied to the same decision tree(j48) as we discussed earlier. For problem "Adult", aggressive pruning or not, the training error is always zero, so we only show the testing error on figure 3. As we expected, the aggressive pruning does not decrease the performance, comparing to without booting though, we did not see much improvement. it might suggest some lower bound as decision tree can achieve.

Now we turn to "IPO" problem, as the performance for training error is shown on figure 4. and testing error on figure 5. We can see boosting performed well with training dataset as the training error quickly decreases to near zero, meaning the model fits the training set pretty well,

and similar treats can be seen testing error rate on figure 5. So we can afford of aggressive pruning.

Figure 3. For Adult Problem—Boosting J48 in WEKA default pruning (c=0.25) vs aggressive pruning(c=0.1)

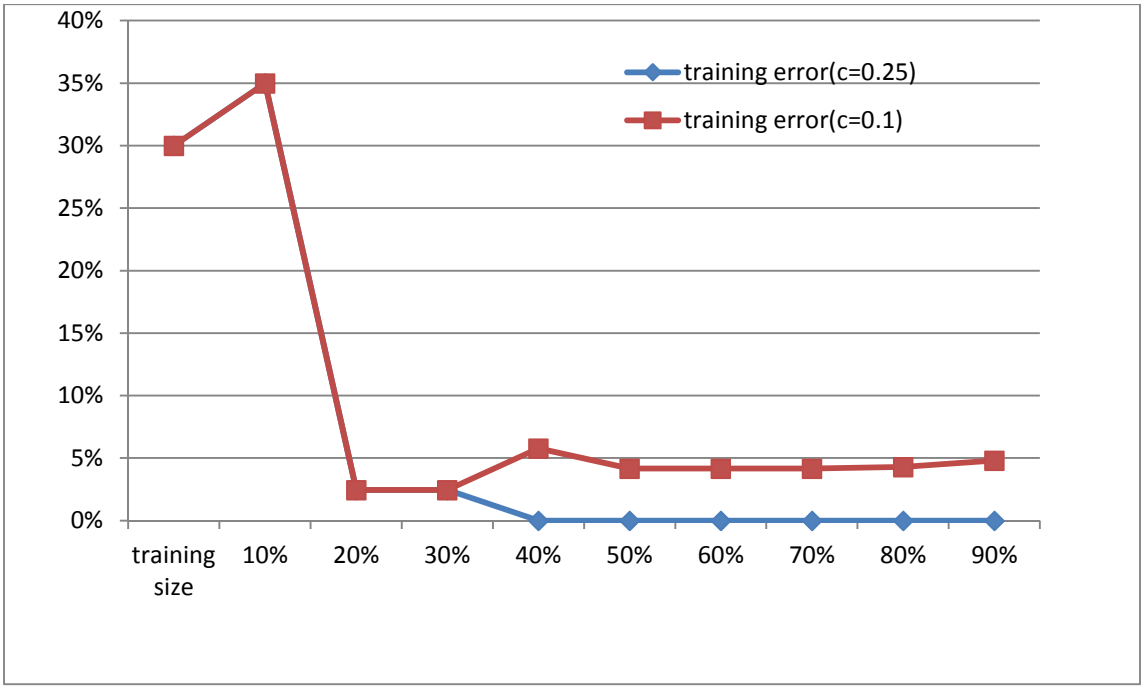

Figure 4. For IPO Problem—Boosting J48 in WEKA default pruning (c=0.25) vs aggressive pruning(c=0.1)

Figure 5. For IPO Problem—Boosting J48 in WEKA default pruning (c=0.25) vs aggressive pruning(c=0.1)

When comparing the performance to decision tree without boosting, we can see generally for these two problem, booting seemed make the performance worse regarding to testing error rate. We would suspect that it is not boosting itself, rather the decision tree that should bear much of the responsibility.

4. Neural Network

A simple one layer 3 node neural network implemented as MultiLayerPerceptron in WeKa. The performance has been shown on figure 6 and 7. One interesting thing we noticed right away is that ANN gets really small training error for small size training set, suggesting it fit the small sample really well in expense good generalization as we can see from the higher testing error rate. As training set size increase, it tends to make more room for generalization with increased training error rate and decreased testing error rate. This is in big contrast to what we have seen so far from decision tree.

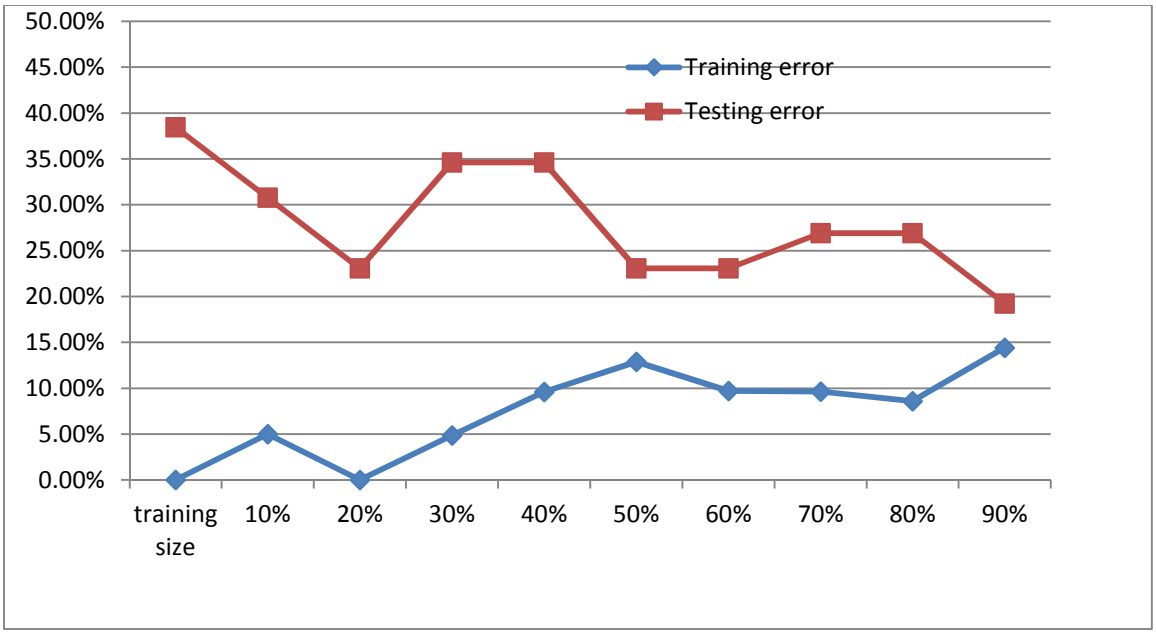

Figure 6. For IPO Problem—ANN with single layer and 3 node

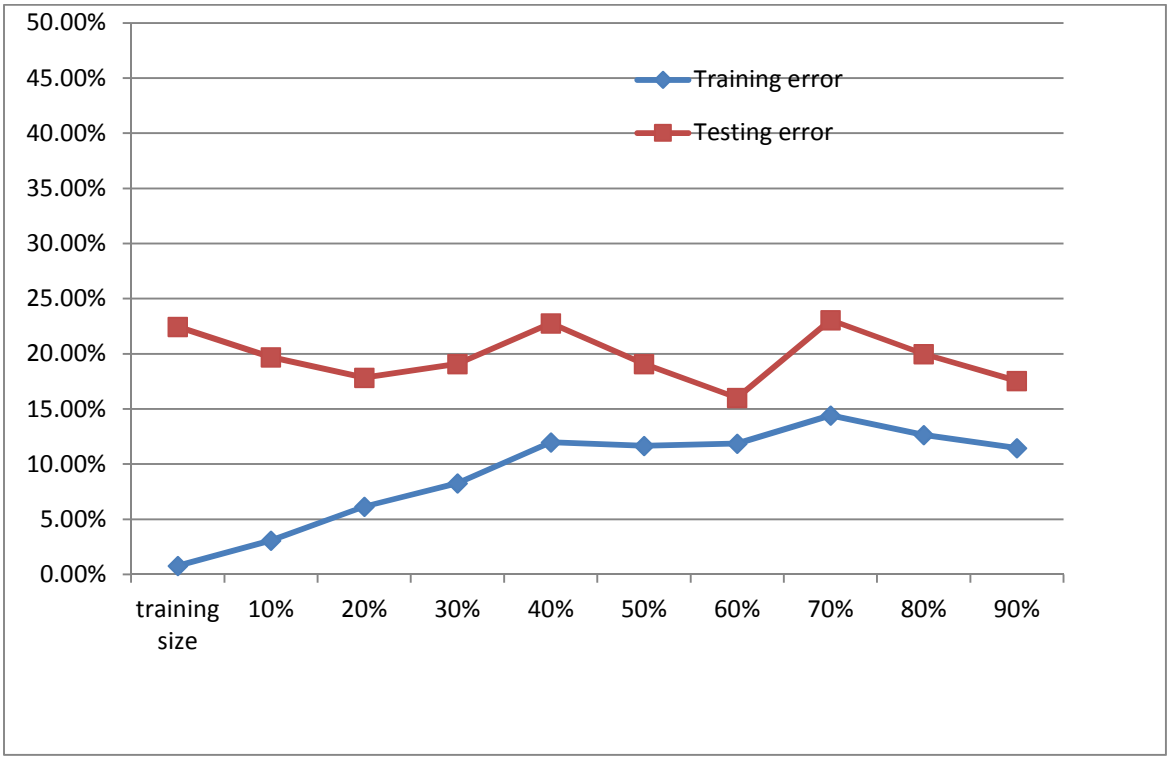

Figure 7. For Adult Problem—ANN with single layer and 3 node

5. SVM

We used wrapper class in Weka called LibSVN to test on SVM algorithm with three different kernel function. One is radial basis function (exp(-gamma*|u-v|^2), second one is sigmoid (tanh(gamma*u'*v+coef0) and the last one is linear(u'*v).we showed the performance as function of training dataset size (percentage) in Figure8 and Figure 9 .the further right data point is the result for cross validation performed on the 100% training set with 10 fold. For the result of other two kernel function, please see table 3. We can see for the SVM with linear kernel function, the general trend is that as training set size increases, the testing error rate decreases. Cross validation definitely helps in the "Adult" problem and probably as well in IPO problem, since the testing error rate does not go up much, a good indication of generalization. However, the run time is exceptionally high comparing with SVM with other two kernel function s (which almost return model instantly) for SVM with linear kernel function as show in figure 10 using "Adult" problem as example.

From table 3 we can see that it is very import to carefully select the kernel function. We want it has the best feature to help separate the "regions" for better model, at same time we should not forget the time involved for computation.

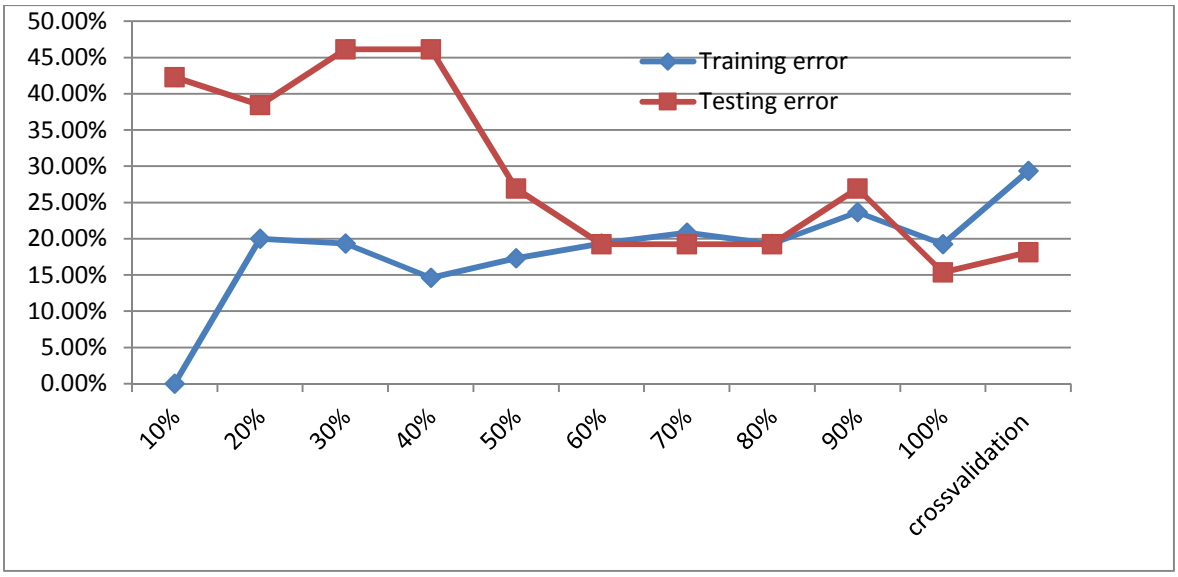

Figure 8. For IPO Problem—SVM with Linear Kernel Function

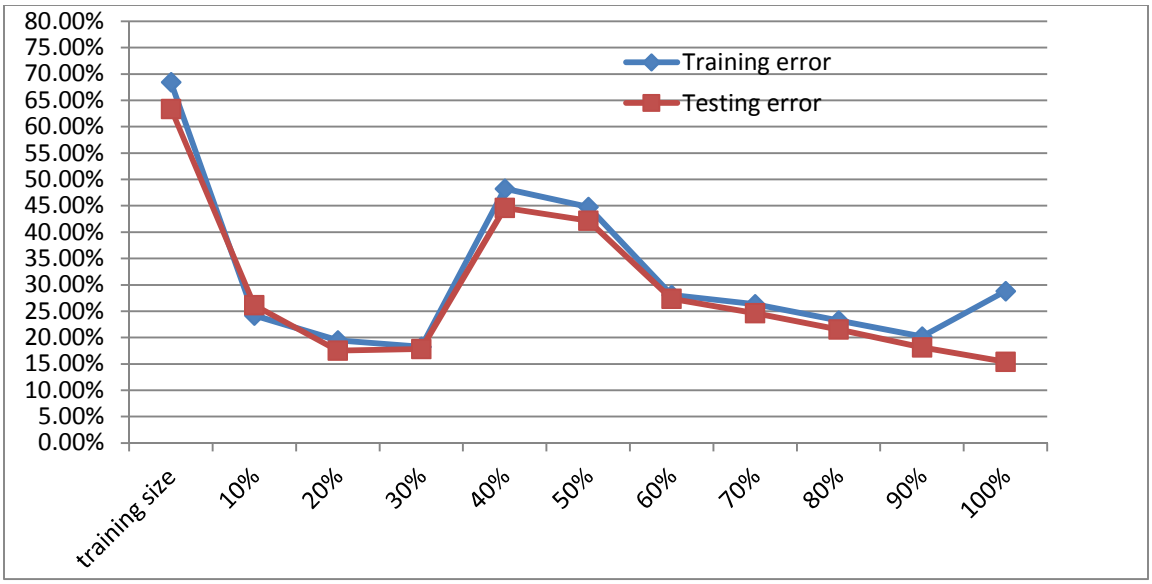

Figure 9. For Adult Problem—SVM with Linear Kernel Function

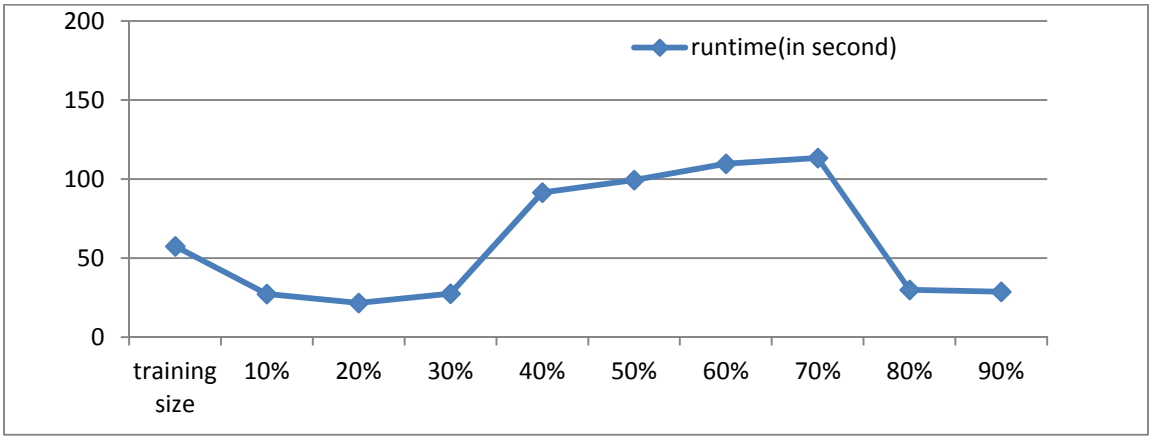

Figure 10. For Adult Problem—runtime for SVM with Linear Kernel Function

	Adult				IPO			
	radial basic function		sigmoid		radial basic function		sigmoid	
	training error	testing error	training error	testing error	training error	testing error	training error	testing error
cross validation	23.12%	20.62%	23.20%	20.92%	45.19%	34.62%	46.15%	34.62%
100%	0%	20.62%	23.20%	20.92%	0.00%	34.62%	36.54%	34.62%
90%	0.17%	20.62%	23.83%	20.92%	0.00%	34.62%	37.63%	38.46%
80%	0.19%	20.92%	24.50%	20.92%	0.00%	34.62%	38.55%	38.46%
70%	0.11%	20.92%	23.71%	20.92%	0.00%	34.62%	40.28%	38.46%
60%	0.13%	20.92%	23.56%	20.92%	0.00%	34.62%	48.39%	42.31%
50%	0.00%	20.92%	23.66%	20.92%	0.00%	34.62%	44.23%	34.62%
40%	0.00%	20.92%	23.85%	20.92%	0.00%	34.62%	46.34%	34.62%
30%	0.00%	20.92%	24.36%	20.92%	0.00%	34.62%	29.03%	30.77%
20%	0.00%	20.92%	21.15%	20.92%	0.00%	65.38%	50%	30.77%
10%	0.00%	20.92%	18.46%	20.92%	0.00%	65.38%	30%	61.54%

Table 3 – Performance for SVM with other kernel functions

6. K-nearest Neighbor

For KNN we used IBK implemented in WeKa, the results for K=2 are shown in figure 11 and 12. The outcome shows quite different error rate curves for the two problems, for Adult problem, the curve is almost flat, that could imply that we are facing a homogeneous instance, increase the sample size does not let eh algorithm perform better or worse. And the fact that testing error is consistently could be result of some characteristic difference between training and test sets, which obvious not shown in IPO case.

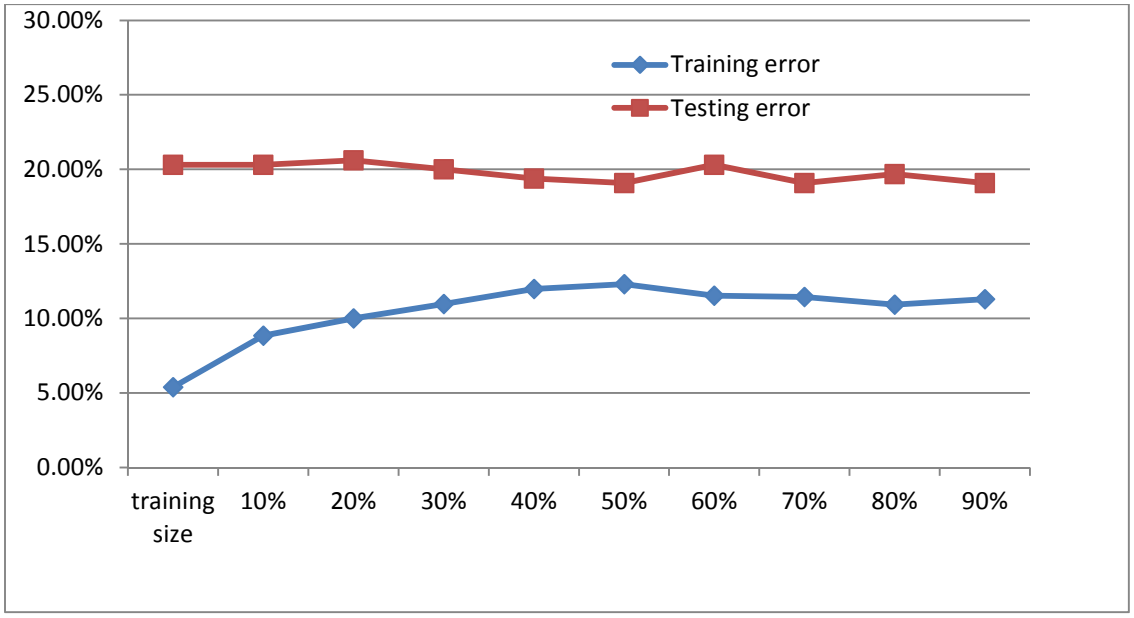

Figure 11. For Adult Problem—KNN with k=2

Looking at table 4, we still see the similar pattern.

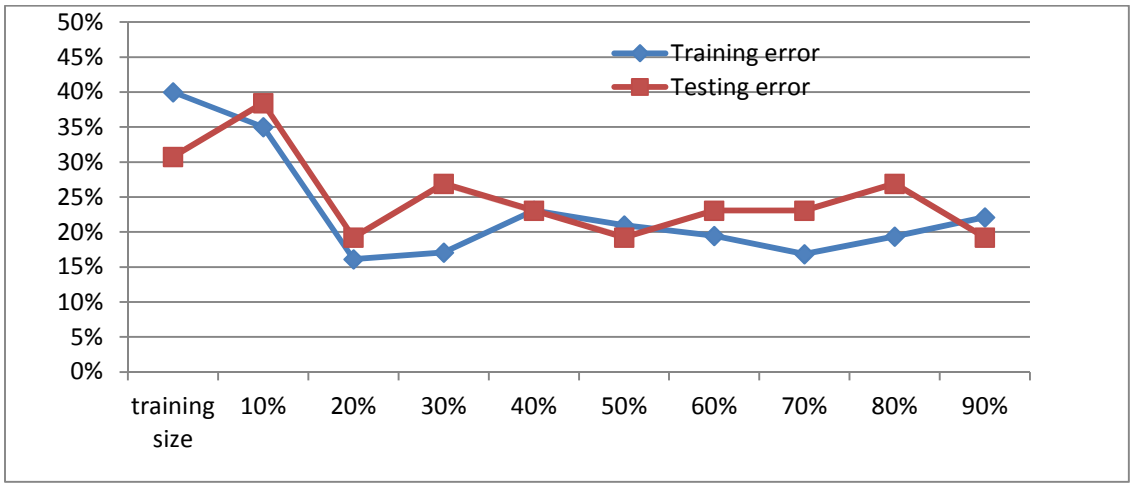

Figure 12. For IPO Problem—KNN with k=2

	IPO				Adult			
	K=5		K=10		K=5		K=10	
training size	training error	testing error	training error	testing error	training error	testing er	training error	testing error
100%	27.88%	23.08%	30.77%	30.77%	12.83%	18.15%	14.52%	16.92%
90%	23.66%	30.77%	29.03%	26.92%	12.55%	17.85%	14.43%	15.69%
80%	21.69%	23.08%	28.92%	19.23%	12.30%	18.15%	14.89%	16%
70%	22.22%	23.08%	29.17%	23.08%	12.29%	16.62%	14.16%	16%
60%	22.58%	30.77%	24.19%	19.23%	13.19%	17.54%	15.36%	15.08%
50%	21.15%	30.77%	26.92%	19.23%	11.52%	17.54%	15.05%	16.62%
40%	21.95%	26.92%	39.02%	19.23%	11.15%	16.62%	13.85%	14.46%
30%	19.35%	34.62%	35.48%	26.92%	11.79%	19.38%	14.87%	16.62%
20%	45%	30.77%	55%	30.77%	9.62%	16.31%	11.92%	17.23%
10%	40%	61.54%	30%	65.38%	9.23%	17.54%	13.85%	18.15%

Table 4 – Performance for KNN for different K

Conclusion

What is the best algorithm out there is always worth debate, even “best” itsself is debatable, it really depends on what your ultimate goal is, and most times you need balance between overfitting and underfitting, cross validation helps under certain circumstance, but not always, and most importantly, how much you know about your data, it is much better to understand the data and come up with simple but good enough model than to throw a complex model to something you know little.