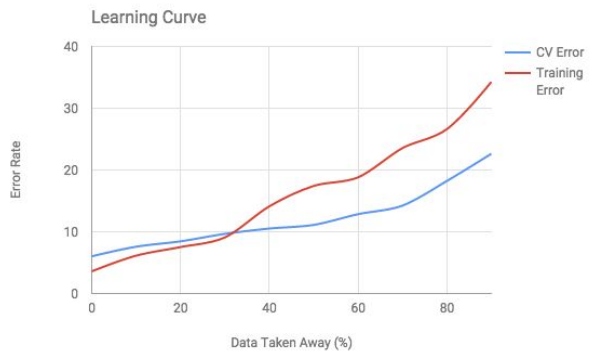
Car Data set

The cars dataset has 1728 instances and 7 attributes. Based on the first 6 attributes (buying price , maintenance cost , number of doors, number of persons that can fit, luggage space, and safety standard) cars are classified into 4 categories(unacceptable, acceptable, good and very good). Attributes values and more information about the dataset can be found in the dataset source file.

Recently, I had to go through a car buying experience and I came across TrueCar. TrueCar classifies used and new cars into different categories based on the price you are going to pay for the car and whether it is a good deal or not. It uses different metrics like the retail MSRP, the average selling price, the price paid by people near you and other attributes. I thought it would be interesting as well to classify if a certain car is good for someone or not based on the attributes it has before even going to the market to buy one. A family might be confused about buying a Honda Crv or a Nissan Murano. Getting the features for the respective car and seeing if that is good or not for that family size and need would be an interesting problem to solve. This cars dataset can be a starting point for working on a classification problem like that. Furthermore this classification problem can be extended to classifying other commodities like house and others things that you want to own.

Decision Tree : J48

In weka the confidence factor (CF) of basic J48 algorithm was changed from 0.1 to 1 in increments of 0.1 to find the CF for the best accuracy. Default value of 2 number of objects per leaf was used. Increasing the CF decreases the level of pruning, hence lesser the CF, the more complex the model is. As shown in the model complexity graph the error rate stabilizes to around 6% at confidence factor 0.5. Both training and cross validation(CV) error rates follow a similar trend, so there does not seem to be any overfitting. The model with CF around 0.2 - 0.3 tends to overfit a bit, even though the training error stabilizes CV error increases.



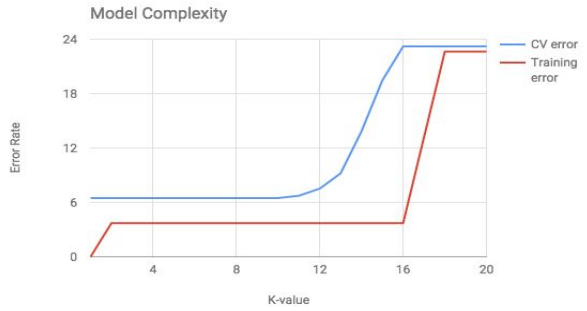
Keeping the CF at 0.5, the min number of objects on leaf was changed from 2 to 20 (2, 5, 10, 15, 20) where the CV error rate varied from 6.67% to 18.7%. Decreasing the size of tree further by increasing the items per leaf increased the error rate. Since the dataset is small with less number of attributes too much pruning and decreasing tree size does not seem to improve model accuracy as models tends to bias towards certain attributes.

CF of 0.5 , with minimum of 2 objects per leaf was then used to model the classifier supplying varying percentage of data from dataset to figure out if we had optimum amount of data to train the model. As seen from learning curve, the CV error rate goes down as we increase the data size, so we might need more data to figure out where that error rate saturates.

With mediocre pruning (CF 0.5), we might want to keep at least 2 items for leaf to have an optimum model (having a good accuracy). The other way around would be to have a smaller tree(min 10 objects per leaf) with a somewhat lesser accuracy (around 85%).

KNN

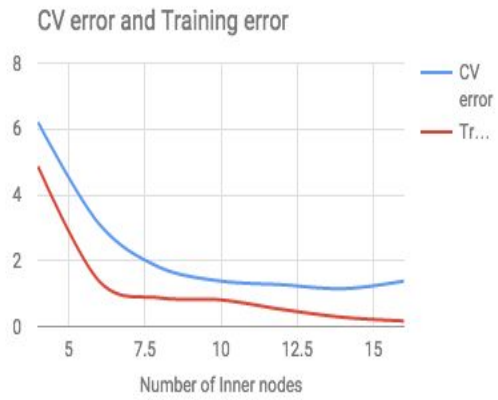
Modeling the kNN classifier with k = 1 was a overfit as the training data gave a 100 percent accuracy and CV error was around 6 percent. This might be because the data might have very less noise. Modeling with k = 2, gave a more realistic error rates on both training and CV runs.
The model seems to overfit a bit when k ranges from 11 to 16 and then underfit for larger k values.

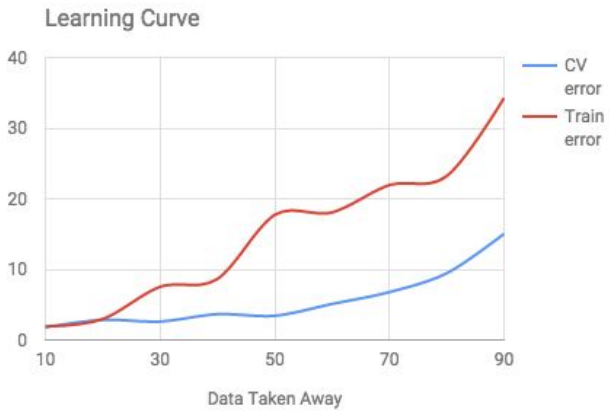


The model might have some bias, as for certain range of k as error rates follow similar pattern.
It might be because the attributes are limited and are not as diverse, like number of persons and number of doors can be inferred to have some dependency. Adding more features to the dataset like mileage can provide a different insight. Even though K was increasing, either error rates were not changing. It might be because the instances might have been separated into small related groups in hyperplane.

Picking k = 5 for the model, learning curve was plotted taking 10 percentage of data away from the dataset for 9 iterations. As we increase the data size, the error rates keep on decreasing and they merge at around where 75% of the data set is used, but that point does not correspond to the best error rate for either curves. As more data is used, training error keeps on decreasing more while CV error tries to stabilize. If this is the case, adding more training data will not help matters as the model might tend to overfit.

Neural Network





Having a single network layer provided very good accuracy on trial weka run. It was not optimal for this small size dataset to have multiple layer of nodes while a single layer was providing a good accuracy. Keeping one hidden layer, with 13 (attributes + class values divided by 2) nodes per layer, and the default parameters for learning rate 0.3 and momentum 0.2, I explored what epoch time would yield the best accuracy. As the number of epochs was increased there was increase in accuracy to some extent, but there was increase in model complexity too.

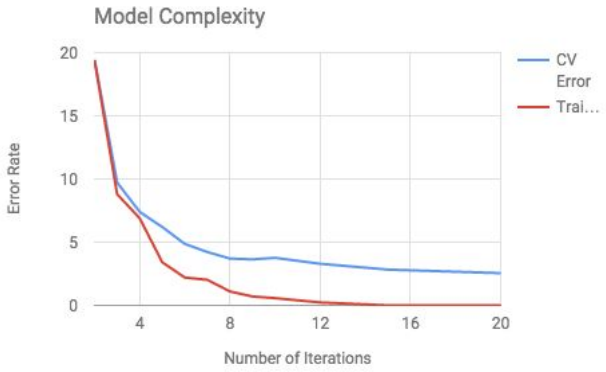
I decided to stick with 25 epochs and then worked on changing the number of nodes in the layer. 25 provided the best bargain for accuracy and complexity. As the number of nodes grew the accuracy increased as well. Based on the graph we can see the error rates saturate when the inner nodes are in range of 10-14.

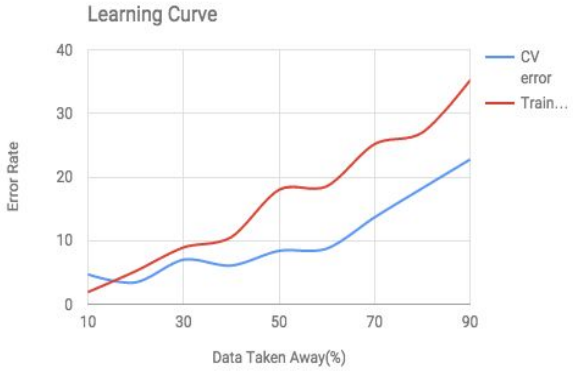
For smaller number of inner nodes and less number of epochs error rates are higher as those models would under-fit, as can be seen from the graphs.

Setting number of epochs to 25 and number of inner nodes to 10, learning curve was made taking away 10 % of data in increments. When all the data was used, the error rates converge to a minimum, so we can conclude the model was almost a perfect fit for the data and there was not much noise either, or the noise was handled well with cross validation.
All the runs on weka for different iterations were under 1 minute for this dataset, so I thought it was not worth to discuss the running time. This is because the dataset has about 1800 instances and only 6 attributes.

AdaBoost

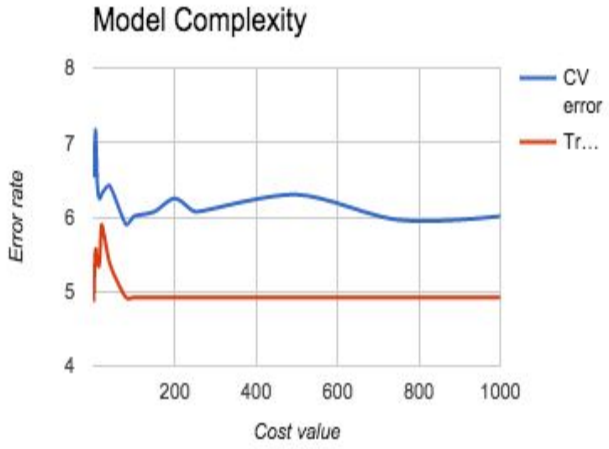
J48 tree with CF 0.1 and minimum number of items per leaf to be 15 giving accuracy of around 80% is taken a base learner.Then we change the number of iterations parameter in weka to get error rates. As we increase the number of iterations, both CV and training error rates continue to decrease. When number of iterations increases to 4, training error decreases more than the CV error. With 12 number of iterations the error rates stabilize. Error rates for training set drops to almost zero. The model under fits for less number of iterations as seen on the left side of the complexity curve. This leads to a higher error for both the training set and the CV set. On the far right side of the plot, we have a very high number of iterations, which over-fits the data. This can be seen in the fact that the training error is very very low, while the CV error is higher.

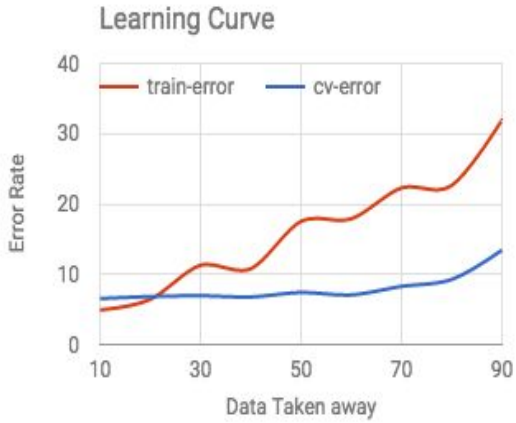




Taking number of iterations to be 8; the point of overfitting, we change the data size, taking 10 % of data away from data set in iterations to get the error rates for a learning curve. As we increase data size the error rates continue to decrease and they finally intersect at when 85% of data is used. Error rate is very low at that point as well. As the data amount is increased there seems to be overfitting as the curves seem to diverge the other way.

All the runs on weka for different iterations were under 1 second for this dataset, so I thought it was not worth to discuss the running time for this dataset. This is because the dataset has about 1800 instances and only 6 attributes.
SVM
On weka, C-SVC SVM type was chosen with default values for gamma and others. Linear and polynomial kernel types were chosen. Polynomial kernel was assigned degree 2.

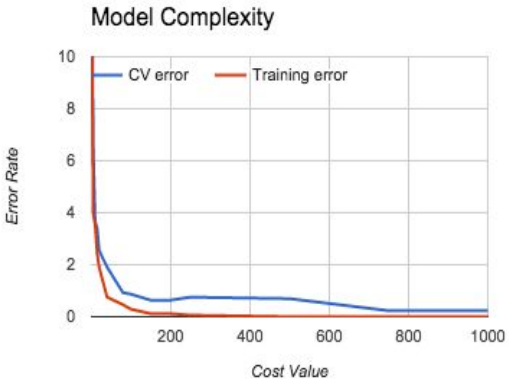


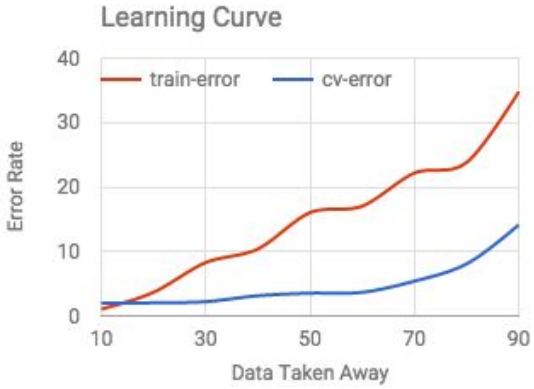


Linear:
Cost value was then changed from values from 1 to 1000 to calculate CV and training errors. For smaller values of cost from 1 to 10, the model seemed to be under fit as the error rates were higher. Error values gradually decreased for cost values from 40 to 100. After cost value increased from 100 to higher, training error rate saturated while the CV error increased a bit and came back down. The model does not seem to be affected by noise that much as both the curves follow similar trend. The data seem to be uniformly distributed as well as the increasing the penalty factor after certain point does not matter much. The model seem to be pretty accurate and not so complex at cost value 15 and cost value 80.

Taking cost value as 80, learning curve was drawn varying the size of the data on the training set. As more data is used training error decreases rapidly while CV error decreases gently. The error rates intersect at about when 80% of data is used. Adding more data does not seem to increase the CV accuracy, but might cause overfitting. This is a small dataset so using less data will clearly under fit the model.

Polynomial:
The polynomial kernel model seems to be more accurate than linear one. The model seems to underfit while cost values are i range of 0-5, yielding higher error rates and overfit a bit when cost value reaches 200. As can be seen from the graph, error rates are the minimal for range 750 to 1000 of the cost values. Using 40 as the cost value (a less complex model) learning curve was drawn. The error rates converge at where 90 percent of data is used and CV error seem to be minimum at that point as well. Adding more data does not seem to help with the CV error rate. This model seems to be a very good classifier.





Conclusion

Algorithm	Best Error Rate
Decision Tree	5.84%
kNN	6.48%
Neural Network	1.15%
Boosting	2.55%
SVM	4.91%

Decision tree gave a pretty good error rate, better than kNN, which was surprising. It might have been that the attributes were more closely related to each other or had similar impact on the outcome. Boosting and neural network provided the best error rates as expected. Having a larger number of instances to model might help improve the decision tree accuracy while having more features might help with the kNN accuracy.

Mashable Data

Description :
The dataset contains information about 39797 mashable articles, with 61 attributes, the last attribute being the number of times the article was shared. The articles were classified as popular and unpopular based on the number of shares. Articles shared more than 1400 were classified popular and others as unpopular. Two non predictive attributes url and time were taken out before modeling. Some familiar attributes were number of videos, number of keywords, global rating of positivity words and others. All the attributes were numeric and the class attribute was nominal. There were 18490 unpopular and 21154 popular articles. More description of the dataset can be found on the data source files.

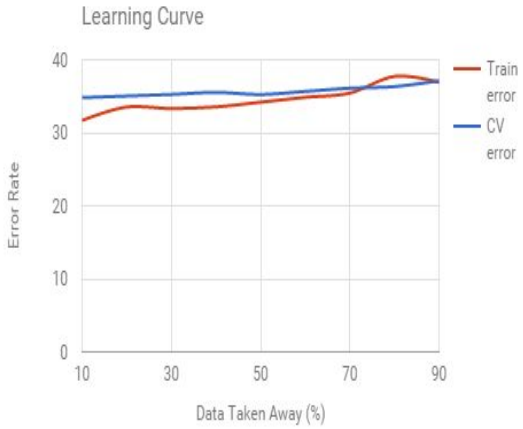
There are millions of people who read mashable articles. For such a huge information platform, the decision to publish an article or not can be a daunting task. A series of unpopular articles can turn an user away which can mean people not coming back for more articles in future. Furthermore, in such a huge user base it is a very hard task to present a tailored content for each user manually. Machine learning models can atleast do a better job at predicting if an article will be popular or not based on the history of other articles and the interest of the particular type of user. Google news does a decent job of tailoring content towards our preferences, but it not up to par yet. Other news outlet still have some work to do before they present truly tailored content. There is also this market where content providers are providing content and news outlets like google news and flipboard are trying to provide a customized set of content based on an individual's preference. I thought working on this mashable dataset would provide a base for me to work on classifying contents to tailor to individual people.

Decision Tree

Since this was a large dataset, I decided using a higher number of objects per leaf with value of 20 and then kept changing the confidence factor from 0.05 to 0.35, pruning the tree less as I increased the CF. As can be seen in the graph the model gave the best accuracy for CFof 0.1 at around 65% accuracy. Increasing the confidence factor more that 0.1 seems to overfit the model to some extent as can be seen from diverging plots.

Holding the confidence factor at 0.1, I tried changing the number of items at leafs to see if that could help with the accuracy. As can be seen in the graph, less number of items per leaf was not helping with getting better accuracy but

the model was overfitting in those cases. Hence having larger number of items per leaf (around 25 - 30) would definitely help because we will have a more friendly tree.



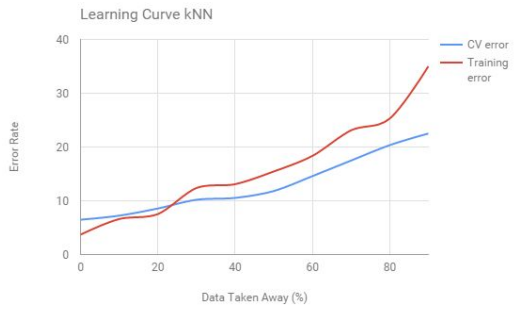
With confidence factor of 0.1 and minimum number of items per leaf to be 10, there was not much difference on accuracy results, when different size dataset were used for training, the CV error rate hovered around 36-37 % on each case. Similar results for accuracy were obtained on similar size datasets with confidence factor (0.25) and keeping number of items per leaf at 20. So trying to change the tree size either by pruning or by having more items on the leaf was inconsequential. A highly pruned tree with a higher confidence factor to about 0.7 and more number of items like 500 on each leaf was not much consequential either.

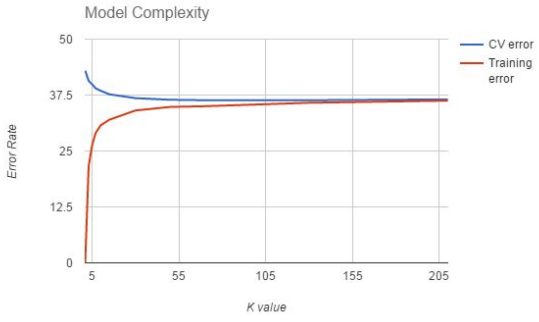
The model seems to have bias for confidence factor 0.1 and number of objects per leaf 30. We can attain nearly the same error rate with a smaller training sample as well as the learning curves meet at where 25% of data is used. Modeling with large training set is computationally expensive, reducing the training sample size can lead to very large improvements in speed even though it will not improve classification much. Adding more features to the data set can help improve high bias model.

On the other hand model with high confidence factor and lesser number of items per node overfits, showing the model has high variance. Doing feature selection or increasing the number of training samples can reduce effect of overfitting of that model.

KNN

Using k = 1, was a complete overfit as can been seen from the complexity curve. Increasing the k value further upto 130 models the data better as the train error increases and CV error decreases. However using a high k value to model just for the sake of small increase in accuracy would not be ideal because of the curse of dimensionality brought by rising k value, which would cause very high training times.

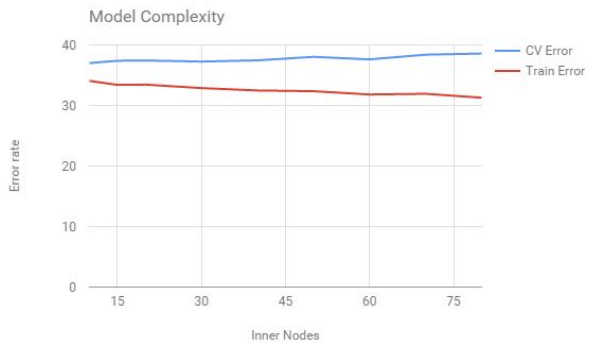


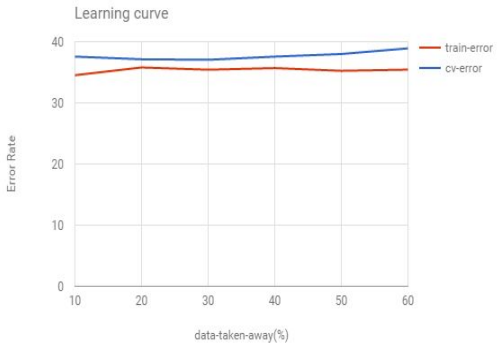


The model with k = 70 was then used to plot the learning curve, taking data away in increments of 10%. Training error and CV error do not seem to change much as more data was used. The CV error rate seems to be minimum when about 50% of data was used. We do not need this huge data set to design a proper kNN model. The model does not overfit much in that case either.

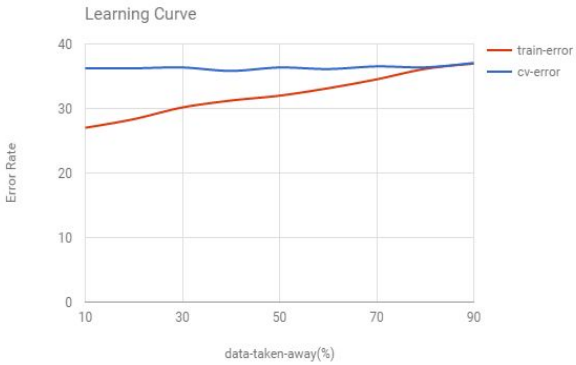
Neural Network

A model with a single hidden layer with 30 nodes was run as we changed the number of epochs to generate a complexity curve. The model overfits for smaller values of cost parameter as can be seen on the left side of the complexity curve. CV error decreases rapidly as we increase the number of epochs up from 25 to 200. Increasing the epochs further causes minor improvements on the CV error rate. But the thing to consider is that increasing the number of epochs causes big increase in the training time. The training time for 500 epochs with 30 inner nodes was around 900 seconds, and it kept on increasing upto 2200 seconds for 1000 epochs.





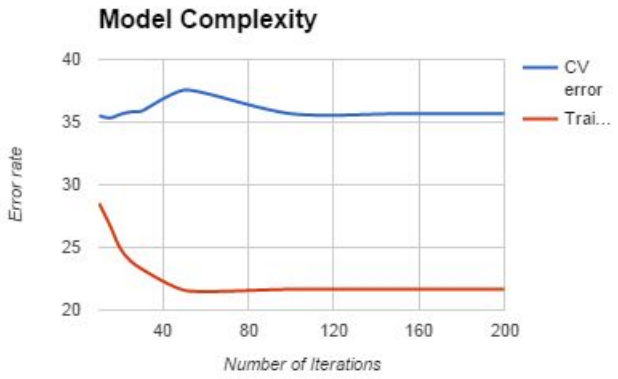
Taking 500 as the number of epochs, the number of nodes in hidden layer was changed to see if there would be a more accurate model with higher number of nodes. However the model seems to overfit as we increase the number of hidden nodes; CV error continued to increase, while the training error decreased. Having 30 nodes in the layer provided the best CV error rate.

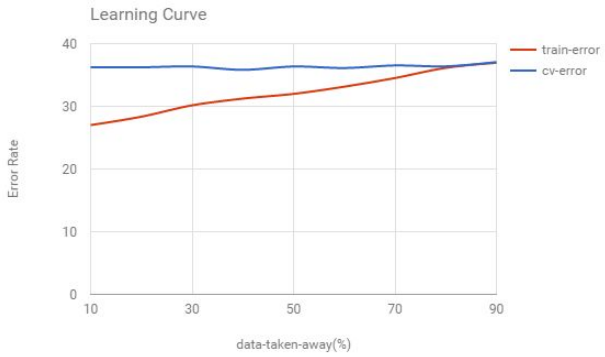


Taking 250 epochs and 30 nodes in one hidden layer as the parameters, learning curve was drawn taking away 10% of data from dataset in increments. While we were using all the amount of data the model seemed to overfit a bit as can be seen on the left part of the chart. At around 80% of data used the model seems to give a better CV error and overfitting is minimal as well.

Boosting

In weka, a J48 decision tree with CF of 0.1 and 200 minimum number of objects per leaf, with accuracy around 60 percent was chosen as a base learner for the AdaBoost function. Number of iterations was kept on changing from 10 to around 400 as the error rates were calculated. As we can see from the chart, the model overfits for smaller number of iterations. After about 90 iterations the model does not produce any changes in the error rates. The minimum CV error rate is 35.33% for 15 number of iterations. It must be because the tree was highly pruned and that many unique base learners could not be created. Boosting does not seem to provide better accuracy than the best of J48 model. J48 with 0.1 CF and 40 number of items per leaf provided 65% accuracy. It might be because of the data that we were testing on. Making a new base learner which is not as pruned and can create higher number of base learners, can help improve the accuracy, but again, creating and iterating with many learners is time intensive process.



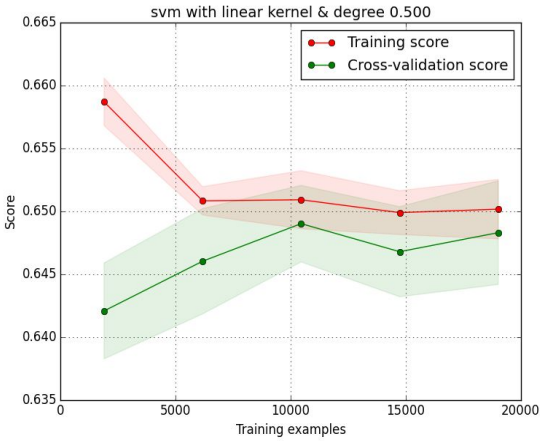


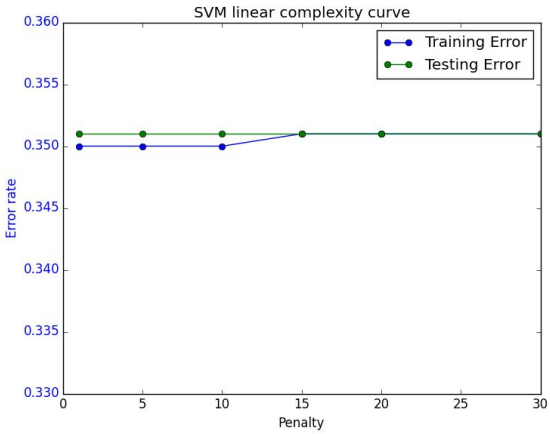
Using 20 number of iterations, learning curve was made taking away 10% of data from data set on each iteration. The model seems to overfit when all the data is used and it still overfits until we use around 30 percent of data, where the curves meet and the training error saturates. So if we decide to use smaller number of iterations and a highly pruned tree we do not need the entire data set to train the boosting model.

SVM

SkLearn was used to train the SVM model. C-SVC type with default parameters was used .(*degree=3*, *gamma=0.0*, *coef0=0.0*, *shrinking=True*, *probability=False*,*tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *random_state=None)*

For the linear kernel, the penalty attribute was changed to create a model complexity curve. The testing error rate went down negligibly as penalty increased from 0.025 to 0.1 and then it saturated at 35% as the penalty factor increased.

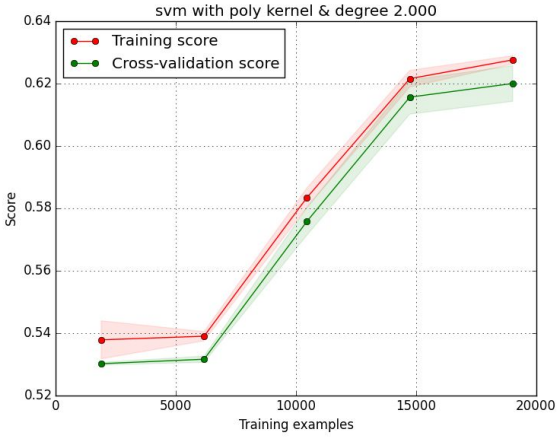


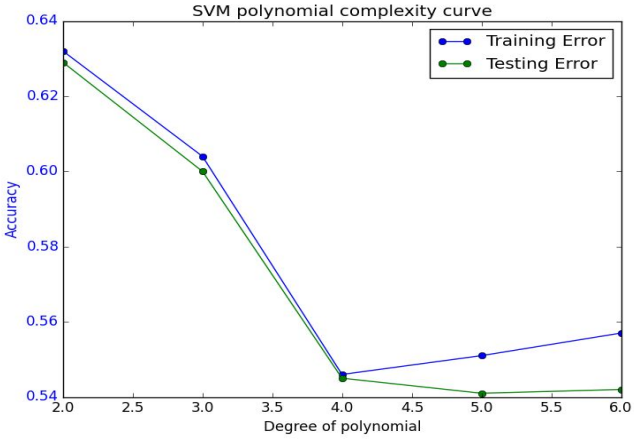


Taking the penalty factor at 0.5, learning curve was drawn. About 50% of data was separated for CV.As the number of sample increaed the CV accuarcy increased upto when 10,000 smaples were used, after which the rates somewhat saturate. The model seems to overfit for smaller data size as can be seen on the left side of the learning curve.

For the polynomial kernel, the degree of polynomial was changed to check the error rates. As we increased the degree the error rates kept on increasing. The model with degree 2 provided the best testing error rate at about 37%. The model does not seem to overfit at the point either. At degree 4 and higher, training and testing error curves diverge as can be seen on the right side of the graph, the model overfits at those cases.

Using degree 2 for the polynomial kernel, learning curve was drawn, increasing the data size in increments. About 50% of data was separated for CV. As the data size increased, accuracy on both the training and CV curve increased. The model with very small amount of sample data does not provide good accuracy as expected. However the model provides a pretty good accuracy when 50% of data is used, which is similar to other algorithms we analysed before. It does not overfit in that case either.





Conclusion

Algorithm	Best Error Rate
Decision Tree	35.8%

kNN	35.35%
Neural Network	32.3%
Boosting	35.33%
SVM	35.1 %

Accuracy of Decision tree was on par with other algorithms. It must be because we were doing binary classification and the major instances of data were linearly separable. Neural network provided the best accuracy at 68% but it was the most time consuming algorithm to run along with boosting. Boosting performance was disappointing, it might have to do with the base learner I chose or it might be because of the data itself. While I spend a lot of time changing the attributes for the base learner, I was not much successful. Overall most of the algorithms did not require the entire dataset to train to get the best CV results.
It must be the way the instances are classified as popular and unpopular based on number of shares being less than 1400 or not, was in first place not a proper way to categorise the data. The shares had high standard deviation of 11,627. 1400 was just a random number to divide such a diverse data. There were many instances around the borderline of 1400 which could either be classified as popular or not.Furthermore having more attributes like what type of reader was reading the article, how many people read the entire article, average time spend on the article, is the article related to current trend can help models predict better.

Reference

Caruana, Rich, and Alexandru Niculescu-Mizil. "Data mining in metric space: an empirical analysis of supervised learning performance criteria." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004.

Hearn, Alison. "Structuring feeling: Web 2.0, online ranking and rating, and the digital ‘reputation’economy." ephemera 10.3/4 (2010): 421-438.

Tatar, Alexandru, et al. "From popularity prediction to ranking online news."Social Network Analysis and Mining 4.1 (2014): 1-12.