CS 7641 Machine Learning: Project-1

[Arti Chauhan: Feb-3-2018]

Abstract

This paper presents the experimental results and analysis of different supervised learning algorithms on two different datasets namely '*Phishing Websites*' and '*Abalone*'**.** Both datasets were taken from UCI repository. Performance of following learning algorithms is contrasted and compared in this report.

- *K-Nearest Neighbor, Decision Tree, Boosting, Support Vector Machine and Neural Networks.*

Dataset introduction and what makes them interesting?

PHISHING WEBSITES

This dataset entails features that have proved to be sound and effective in predicting phishing websites. It has 2456 instances and 30 discrete attributes. Goal is to predict if a website is malicious or not.

Why is it interesting? – In current ever-growing digital world, online security is of utmost importance. This issue gives us an opportunity to leverage ML techniques to define and refine features that reliably characterize phishing websites and build a robust phishing detection system that can protect online assets from malicious actors. Phishing websites usually have layout as legitimate websites and attackers get very creative, thus making this problem particularly challenging and interesting.

ABALONE

This is a slightly bigger dataset than Phishing, with 4177 instances and 9 attributes (mix of categorical, integer real values). The goal is to predict the age of abalone from physical measurements such as length, weight, height, gender etc.

Why is it interesting? - The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope, which is a tedious and time-consuming task. By using powers of ML techniques if we can predict the age of abalone, it will greatly help marine biologists in their research. Another interesting aspect of this dataset is that it's a multi classification task as opposed to Phishing dataset (which is a binary classification task). This will give me chance to use these learning algorithms beyond binary classification.

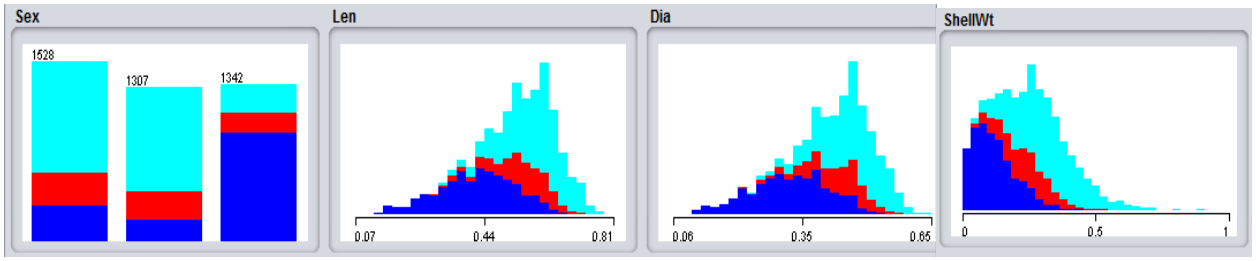
Understanding the data

Instead of blindly dumping the data as is into a learning model, it is critical to get some insights into it via EDA. Charts below help in understanding the distribution, variance, and feature correlation in each dataset.

ABALONE

Insights from charts below

1. From Ring count histogram (Fig-b) , it is evident that this is a not a balanced dataset.There are way less samples for range 1-7 and 12-29 when compared to 8-11. To combat this, I tried to bin data several different ways and tested it and finally settled on 3 classes - 0 [<8] , 1 [8-10] and 2 [>10], which makes dataset somewhat balanced.
2. I dummyfied categorical variable 'sex' otherwise it would have posed a problem for ANN model.However feature-importance graph(Fig-c) marks 'sex' as least discriminating feature.



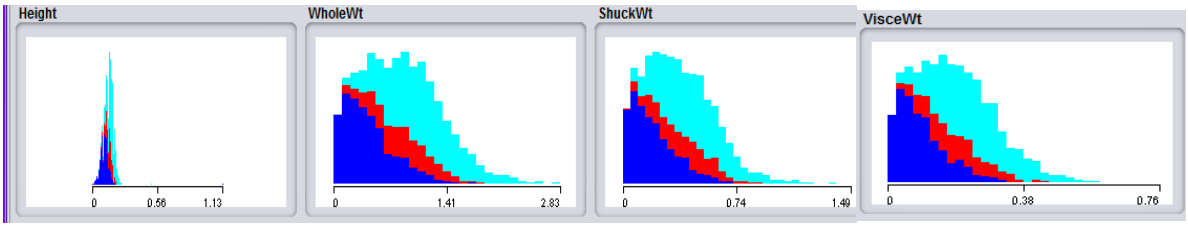


Fig-a

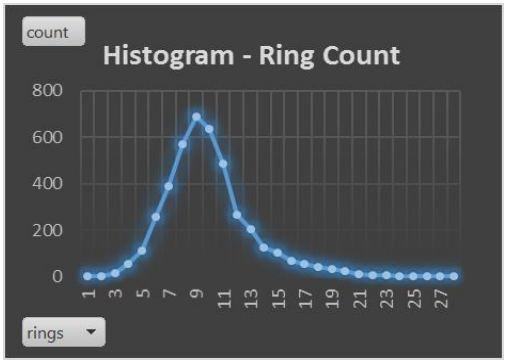


Fig-b

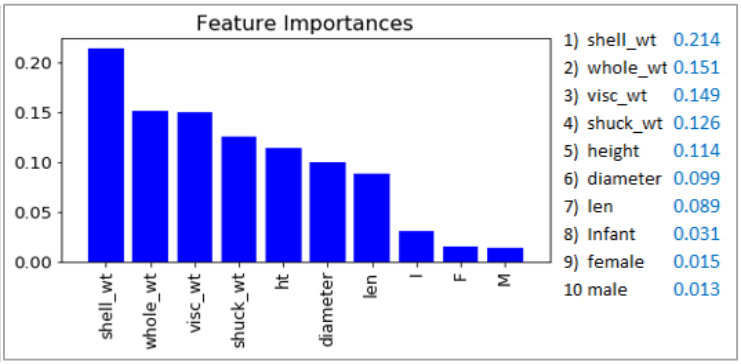


Fig-c

PHISHING WEBSITES

1. This is a quite balanced dataset ie both classes are proportionally represented.(Fig-d)
2. Most discriminating features are SSL state, url of anchor and site's populatrity (web_traffic).
3. Though there are 30 features in all , top-5 features explain 75.7% of the data and top-10, 90.5%.

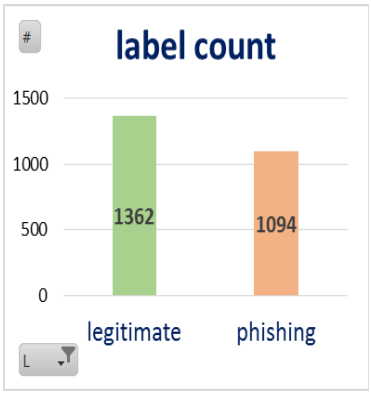


Figure-d

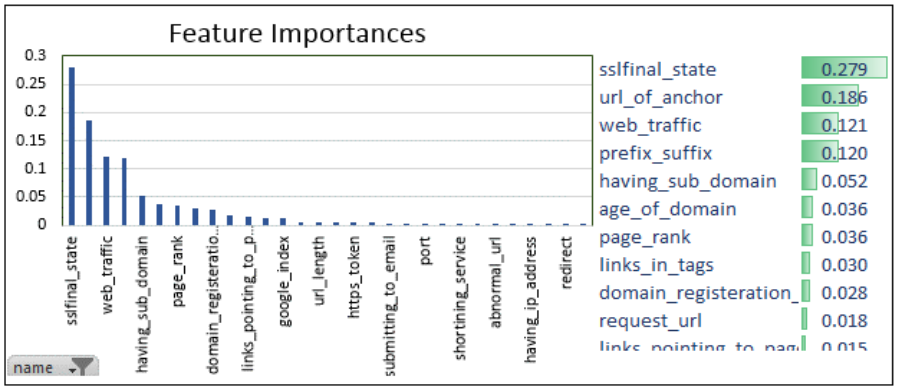


Figure-e

Please note a) I evaulated Feature-importance for sake of my understanding but I didn't remove any features for this assignment, as was advised on piazza. b) Feature importance was computed using Random Forest classifier (with 300 estimators).

Methodology

- Analysis was done using Scikit-learn (python 2.7). Each dataset was split into training (70%) and test set (30%). Given the relatively small size of these datasets, I decided to use cross validation (CV). More importantly, CV helps to get more accurate estimation of model's predictive power on unseen data and generalizability.

- Data was scaled to avoid feature with high value dominating other features when computing distance in KNN. It can make learning process faster in case of neural net (NN).

- **Training set selection** – Data was shuffled and stratified before performing train-test split to ensure that we have proportional representation of each class in train, validation and test set.

- **Model selection** - Hyperparameters for each learning algorithm were tuned using training set (via 5-fold cross validation) and analyzing model complexity via validation curves (ie training and Cross-Validation score as a function of different values of hyperparameters). Model with best CV-score is selected and its performance is evaluated against unseen test-set. Test set is not touched at all during training phase.

- **Performance metric** – For each model, classification accuracy is used as primary evaluator. In addition, Precision/Recall/F1-measure/Roc-Auc is evaluated as secondary measure to understand what type of errors (Type I /II) model is making and where classification threshold can be drawn. However, analysis below primarily talks about accuracy because F1-measure and classification accuracy was very similar for both datasets.

- **Learning curve** uses Cross validation with 30 iterations to get smoother mean test and train score curves, each time with 20% data randomly selected as validation set. Solid line shows the mean score and shaded region shows standard deviation (std_dev) of scores.

In following sections, graphs for Model complexity, Learning curve and Time complexity is presented for each algorithm, accompanied by brief description of which and why certain hyperparameters were selected and analysis of afore-mentioned graphs.

1. K-Nearest Neighbors (KNN)

Range of K [1-200] was explored with two different weight settings 'uniform' and 'distance'. This exercise was repeated for Manhattan, Euclidean and Minkowski metric for both datasets.

- A very small K (neighbor#) leads to overfitting (high variance) and very large K leads to underfitting (high bias). As K increases so does test time.
- 'Weight' metric decides if all neighbors should be treated equally ('uniform') or one should be preferred more over the other based on 'distance' (some notion of similarity).

Below I am showing crucial metrics but complete results can be found in *output/tune/knn_ *.xlsx*.

1.1 PHISHING WEBSITES

1. Model complexity: Validation curves in Fig1-1a below shows train and CV score as a function of K (number of neighbors) for 'Manhattan' distance for two different weight metrics –'uniform' and 'distance'. Data shows that there isn't much difference in model's predictive power when distance metric is changed from Euclidean to Manhattan or Minkowski. However, different trends are achieved when 'uniform' vs. 'distance' based weight is used. Hence I evaluated best model for each weight metric to have a better understanding.
   - *Model-1 : Best K when weight = **distance** => {'n_neighbors': 6, 'metric': 'manhattan', 'weights': 'distance'}*
   - *Model-2 : Best K when weight = **uniform** => {'n_neighbors': 1, 'metric': 'manhattan', 'weights': 'uniform'}*
2. Learning curve: Fig1.1b &d show learning curve for above models. As number of training examples increase, training accuracy degrades however test accuracy improves, indicating reduction in overfitting. I prefer model-1 over model-2 because it gave better test-scores when training set size is small and had less std_dev in test-scores. Moreover, in model-2 (K=1), decision boundary would be too jagged to generalize.

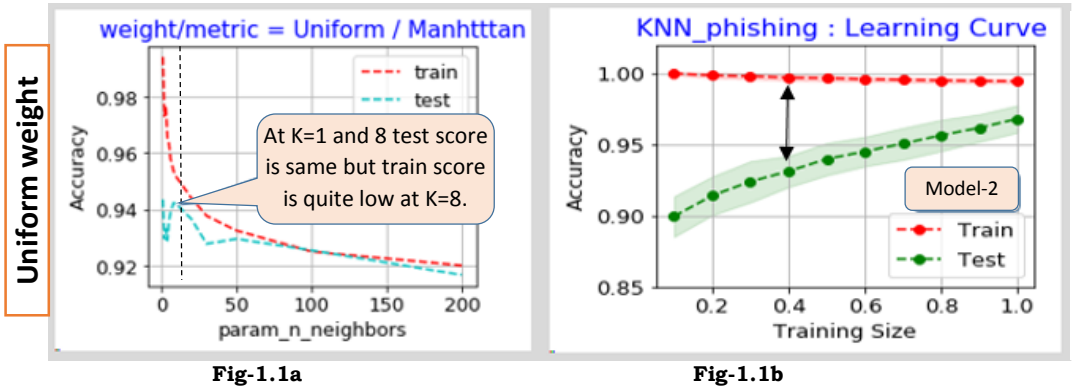


Fig-1.1a **Fig-1.1b**

For 'uniform' weight, both train and CV accuracy go down as K is increased beyond 8

Best results are achieved at K=1.

Distance based wt

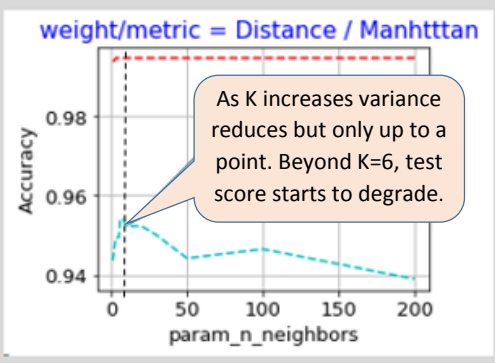


Fig-1.1c

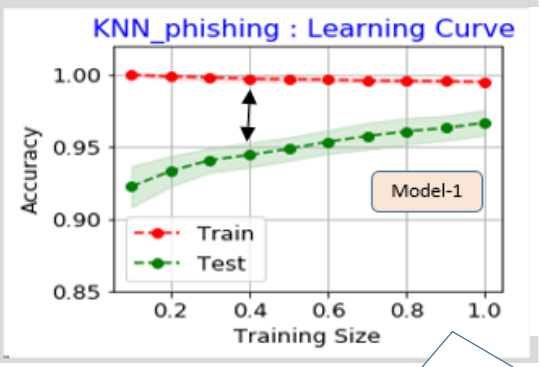
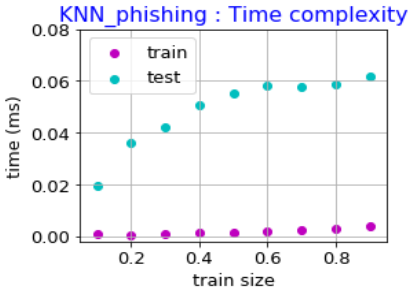


Fig-1.1d

For 'distance' based weight, train accuracy stays relatively same for all Ks whereas CV accuracy starts to degrade beyond K=6.

Gap between train and score shrink as sample size increases but doesn't completely close. More samples or informative features can help reduce this gap.

Time Complexity: As KNN is a lazy learner, train-time is low but test-time is high.Test-time increases as sample size grows [O(n) - assuming '*quickselect*' algorithm is used to compute the k^{th} smallest distance.]



1.2 ABALONE

1. **Model complexity** – Changing metric from Euclidian to Manhattan or Minkowski didn't change results much. But weight metric had impact on scores.
   - Signs of overfitting to train data are seen when distance-based weight is used (Fig-1.2a) where train scores are very high and there's a large gap between train and CV scores. However, both models gave similar test scores. So I selected weight=uniform to find best classifier.
   - Fig-1.2c shows accuracy scores as function of K for 'uniform' weight. Though train accuracy decreases between K range 1-40, test score improves, reducing overfitting, achieving best scores at K=40.For K > 40 , both scores degrade showing model becomes too simplistic beyond that point to learn anything significantly meaningful.
2. **Learning curve:** Fig1.2d show train and CV scores for best classifier [K=40, weight=uniform, 'metric': 'euclidean'] as a function of training-set size. As number of training examples increase both scores continue to improve, indicating more data will help the model to learn. However, std_dev in test-scores (green shaded region) doesn't reduce much as training-set size increases, which was bit concerning.
3. **Time Complexity:** Fig-1.2e shows time to train and test as function of training set size. Since KNN is a lazy learner train-time is less than that of test-time. As expected, test time increases as sample size increases.

Model_1 = {'n_neighbors': 30, 'metric': 'euclidean', 'weights': ***'distance'***}

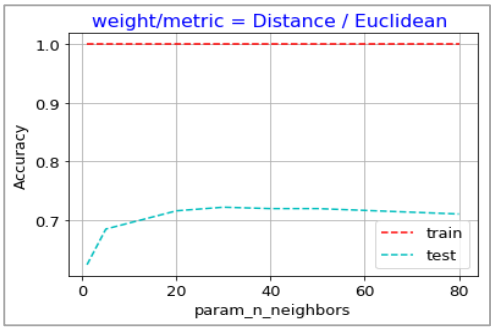


Fig-1.2a

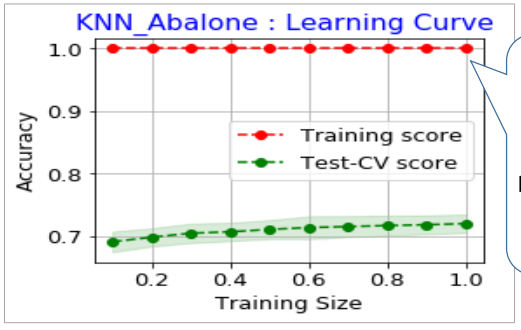


Fig-1.2b

This model overfits train data. Large gap seen between train and test scores

*Model_2= {'n_neighbors': 40, 'metric': 'euclidean', 'weights': **'uniform'**}*

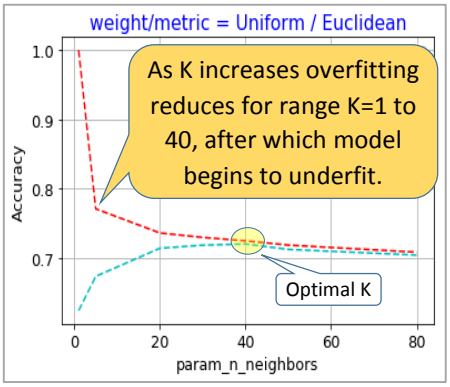


Fig-1.2c

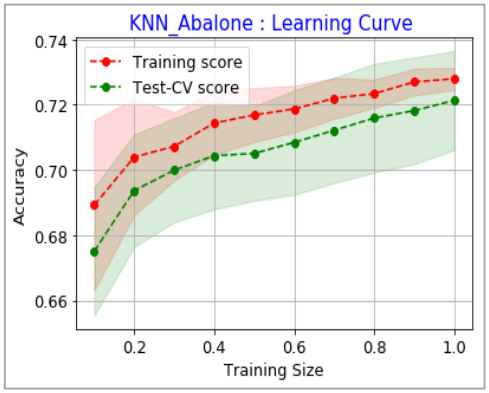


Fig-1.2d

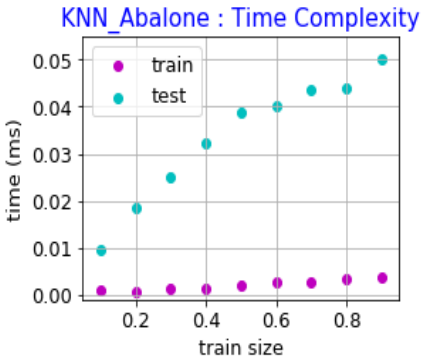


Fig-1.2e

2. Decision Tree (DT)

DT learners can create over-complex trees that do not generalize the data well. This is can be overcome by pruning. Though Sklearn doesn't support post pruning, DT can be pre-pruned by limiting max_depth or min_samples required at leaf. For both datasets, I explored a range of tree depth [1-100] for 2 different splitting criteria ('Best' & 'Random'). This exercise was repeated for both impurity measure 'Gini' and 'Entropy'.

2.1 PHISHING WEBSITES

1. Fig-2.1a shows model's performance as tree-size grows for different impurity measures and splitting criteria.
   - *Using 'Entropy' as impurity measure gives a slightly better performance than 'Gini'.*
   - *Using 'best' split gave slightly better results than 'random' split.*
   - *As tree size grows both train and CV scores improve but only up to a point [depth=10], after which model is not able learn anything meaningful even though complexity (depth) increases. Applying Occam's razor principle, I chose depth =10 which gives least complex model with highest accuracy.*
2. Fig2.1c gives learning curve for the best classifier. As training size increases, train accuracy decreases and test accuracy improves, indicating model is generalizing to new data. Though both lines don't completely merge but they are trending in right direction, indicating that this gap can further reduce with more training examples and/or with better features.
3. Fig2.1d gives time to train and test as function of training set size. Unlike KNN, here train time is higher than that of test as it's an eager learner. As expected training time increases with training set size. Test time stays relatively flat.

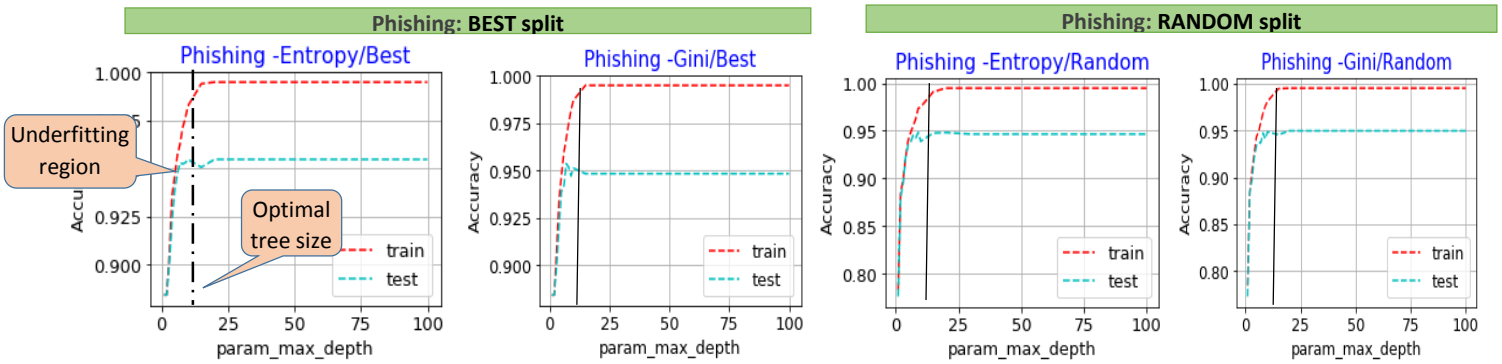


Fig-2.1a

Best classifier

Best Performance for different param criteria

mean test score	mean train score	criterion	max depth	splitter
0.954625	0.982402	entropy	10	best
0.951134	0.986911	gini	10	best
0.948226	0.994764	entropy	20	random
0.949389	0.994764	gini	20	random

Fig-2.1b

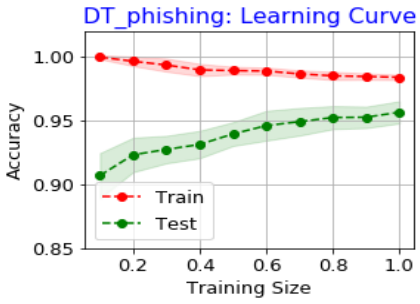


Fig-2.1c

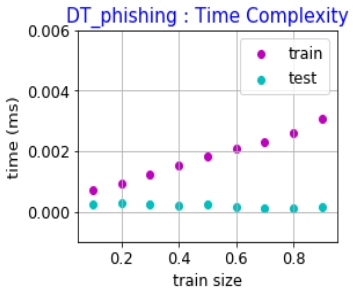


Fig-2.1d

2.2 ABALONE

Exhaustive search was performed to find a classifier that gives highest accuracy with smallest tree depth (least complex model). Fig2.2a shows few validation curves.As depth increases, model becomes more complex, leading to reduced bias but increased variance, ie model learns training data rather than its characteristic and loses its 'generalizibilty' and thus performs really bad on test-data (blue dotted line).

Fig2.2c shows the learning curve. As training set-size inreases, training accuracy decreases but test-accuracy increases and gap between the two shrinks significantly, giving reasonable accuracy.This makes me believe that model doesn't suffer from very high bias and/or variance. Time to train was higher than time to test, as expected.

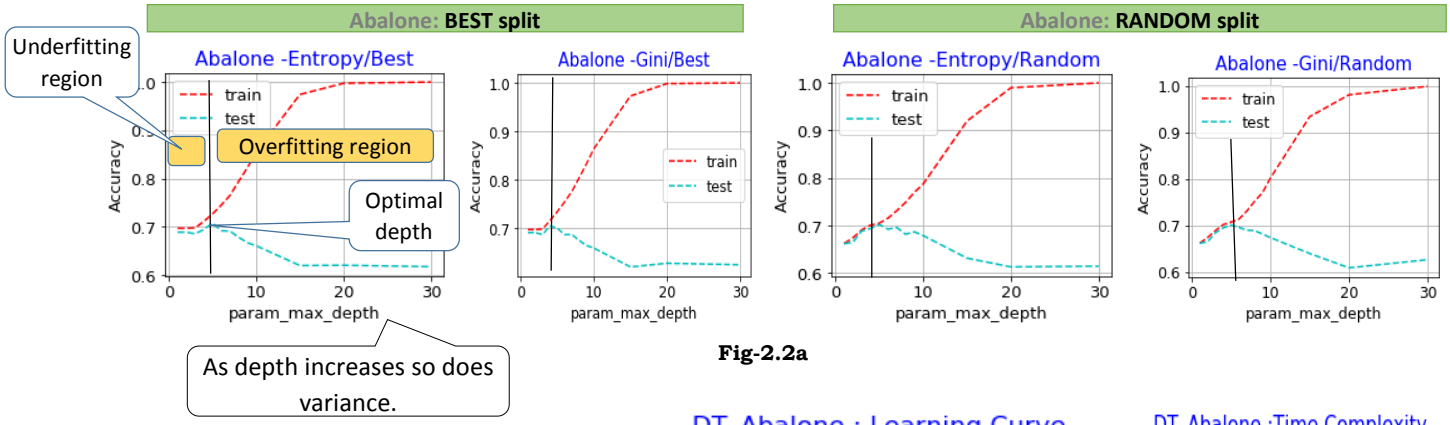


Fig-2.2a

mean test score	mean train score	criterion	max depth	min samples split	splitter
0.703045	0.704927	entropy	5	10	random
0.700992	0.707321	gini	5	2	random
0.707834	0.726736	entropy	5	2	best
0.705098	0.715873	gini	4	2	best

Fig-2.2b

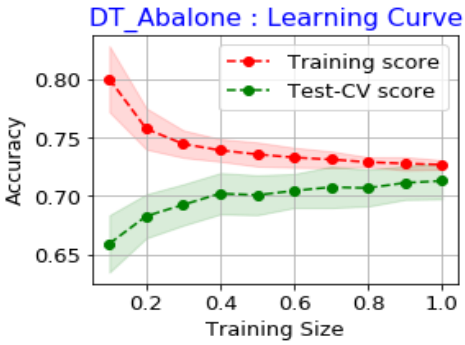


Fig-2.2c

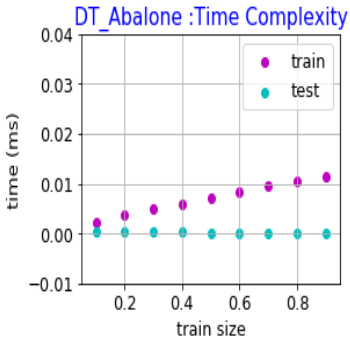


Fig-2.2d

3. Boosting

Adaboost with decision tree as base estimator (weak learner) was used for this exercise. Number of base estimators control the complexity of boosting model. Boosting is known to decrease bias as well as variance, however it is sensitive to outliers and behavior (over/underfitting) of base estimators.

Search for optimal model was conducted by tuning number of base estimators [1-1000], learning rate [0.1-2.0], max_depth [1-30] & splitting criteria [random/best] for both datasets. Below I am showing only top results, however complete results can be found in '*output/tune/Adaboost_*.xlsx*'

3.1 PHISHING WEBSITES

mean time		mean score		base estimator (DT) param			Adaboost param	
train	test	test	train	criterion	max depth	splitter	learning rate	estimator#
1.9218	0.0696	0.9604	0.9948	entropy	10	random	0.1	600
0.2962	0.0116	0.9599	0.9866	entropy	5	random	0.01	100
0.1588	0.0062	0.9593	0.9933	entropy	7	random	0.01	50
0.6046	0.0230	0.9587	0.9914	entropy	5	random	0.01	200
0.0306	0.0014	0.9581	0.9772	entropy	7	random	0.01	10
0.7646	0.0234	0.9581	0.9948	entropy	7	best	0.1	200
0.3376	0.0118	0.9575	0.9859	entropy	5	best	0.01	100
0.6786	0.0230	0.9575	0.9917	entropy	5	best	0.01	200

Model-1

Model-2

1. Table on left shows some top performing classifiers for Phishing dataset along with their hyper parameters.
   Here there are several winners, so I examined learning curve for bunch of them.
   Focusing on top-2, Model-2 gives comparable accuracy & F1-score to Model-1 but in 1/6thof the time.

3. Learning curve (Fig-3.1a & b) : Model-1 has slightly less std_dev in test-scores compared to model-2.Thats probably because it uses 6x more estimators (thus effective in reducing bias/variance) but on flipside it uses 6x more time to train. Based on business objective, one model can be more favorable than other, balancing tradeoff between time and accuracy.
   - Note that model-1 (with more learners) tends to overfit training data compared to Model-2.
   - Both models show some variance, which should reduce as more examples are fed.
4. Training the model took bulk of the time, testing was fast (1.9 vs .07sec).Since Adaboost here is an ensemble of trees, it takes more time than plain Decision tree. Training time increases as number of estimators increase.

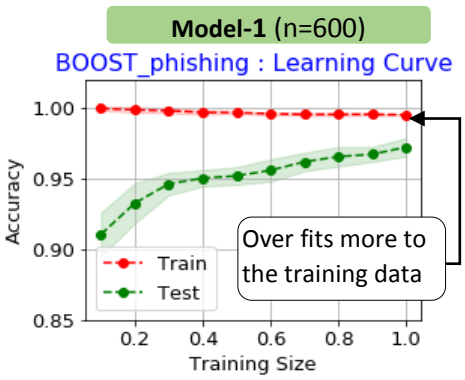


Fig-3.1a

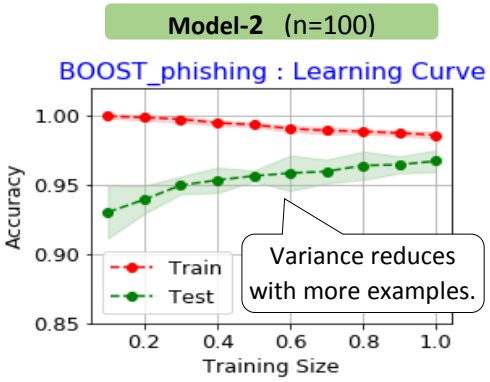


Fig-3.1b

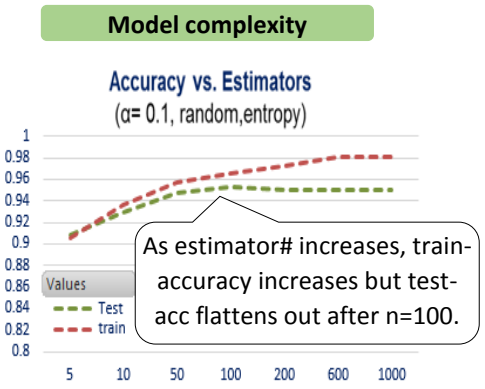


Fig-3.1c

3.2 ABALONE

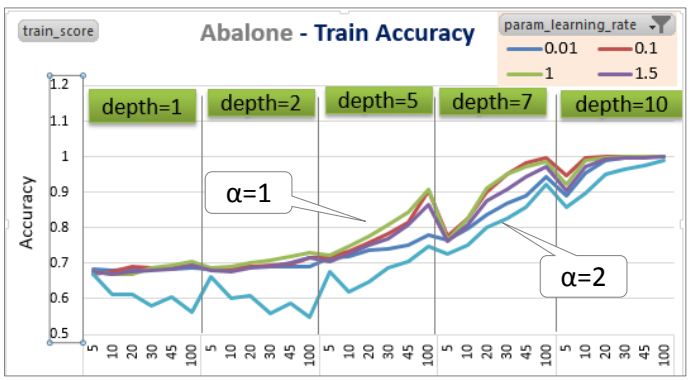


Fig-3.2a

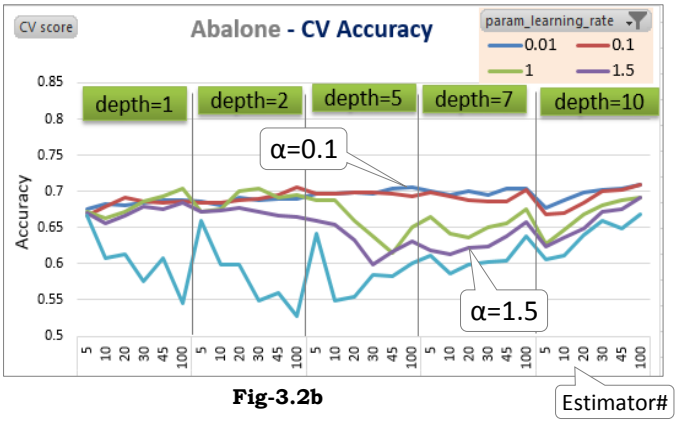


Fig-3.2b

1. Model Complexity: Fig-3.2a & b show train & CV scores as function on number of estimators (x-axis) for different learning rates. This information is presented for different tree depths. It can be seen from the graphs that

- There's a tradeoff between learning rate (α) and number of estimators required. For α=2 (larger learning steps) more estimators are required to achieve similar accuracy as with lower α values.
- As max_depth increases, train accuracy reaches 100% but without any substantial gain in CV accuracy. As depth is increased DT tries to fit training data better, leading to overfitting.

2. Learning curve: Fig3.2c shows learning curve for best model. As sample-size increases, overfitting (variance) reduces - training accuracy declines but test-accuracy improves, shrinking the gap between the two. These results are better than plain vanilla Decision tree. Just for kicks, I ran this model with 1000 estimators, but it didn't improve the scores significantly.
3. Boosting took significantly more time to train and test when compared to other learning algorithms - time increases as model complexity (deep trees and/or large estimators) increases.

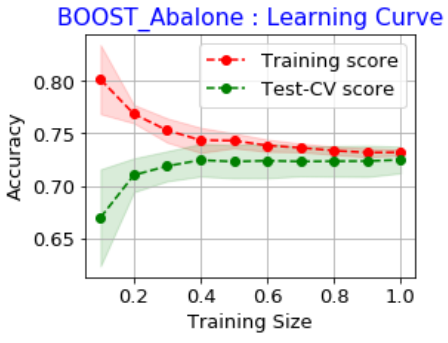


Fig-3.2c

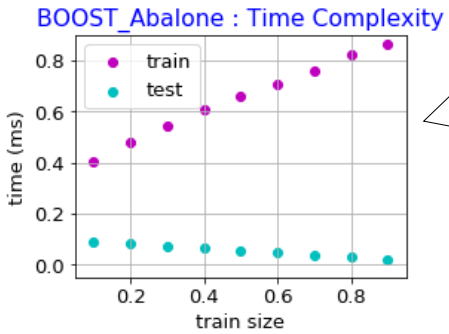


Fig-3.2d

best_model = {
estimators : 200,
criterion : 'entropy',
max_depth: 1,
learning_rate : 0.5,
splitter : 'best'}

4. Support Vector Machine (SVM)

Grid search was performed for a range of C (penalty parameter) for Linear and RBF kernel, for both datasets. Additionally, Gamma (kernel coefficient) was tuned in case of RBF kernel. Goal was to find optimal C & gamma.

- **C** => cost of misclassification. Too small C can lead to higher bias (underfit). Vice versa for large C.
- **Gamma**=> inverse of radius of influence of support vectors. With small gamma, two points are considered similar even if they are far apart. Very small gamma may lead to underfitting. Vice versa for large gamma.
- **Kernel:** RBF kernel is more complex and takes more time to train but can fit data that is not linearly separable.

4.1 PHISHING WEBSITES

Table below shows some top performing classifiers along with their hyper parameters. RBF performed better than linear kernel, suggesting that data isn't linearly separable.

mean test	mean train	C	gamma	kernel
0.955788	**0.987929**	**10**	**0.1**	**rbf**
0.955788	0.991274	20	0.1	rbf
0.955207	0.990692	15	0.1	rbf
0.951716	0.975422	2	0.1	rbf
0.945899	0.968442	1	0.1	rbf
0.94299	0.962915	20	0.01	rbf
0.941827	0.954916	10	0.01	rbf
0.940663	0.945171	0.1	0.1	rbf
0.9395	0.946044	2	0.01	rbf
0.9395	0.959569	15	0.01	rbf
0.937755	**0.944008**	**0.1**		**linear**
0.937173	0.945608	0.5		linear
0.937173	0.94619	1		linear

- Fig-4.1a and 4.1b show learning curve for linear and rbf kernel. Rbf model is able to achieve 95.6% test-accuracy whereas linear model achieves 93.7%. Linear model underfits the data.
- For RBF, there are two of winners – both yielded very similar learning curve. I picked the top model {RBF , C=10, ϒ = 0.1}.
- Learning curve (Fig4.1b) shows that gap shrinks between train and test accuarcy as number of samples increase, indicating model is able to learn from new data & generalizing
- Fig4.1c shows train and test time. As train size increases, train time becomes noticeably large.

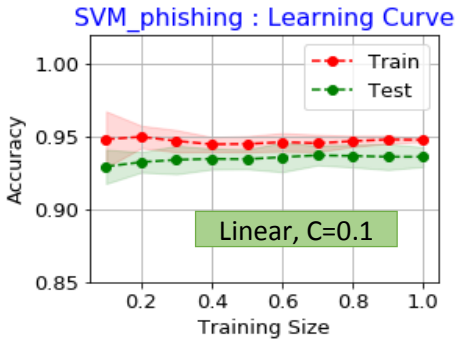


Fig-4.1a

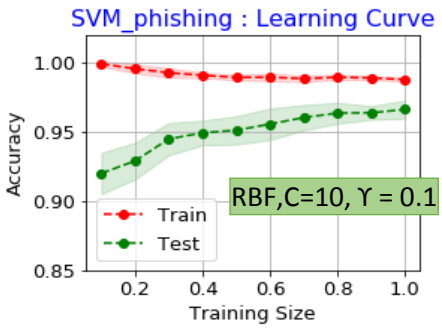


Fig-4.1b

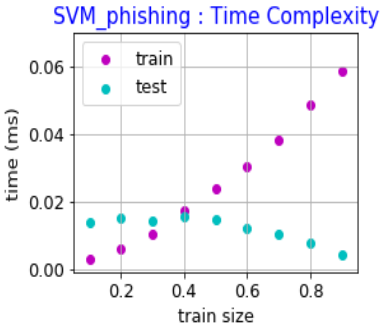
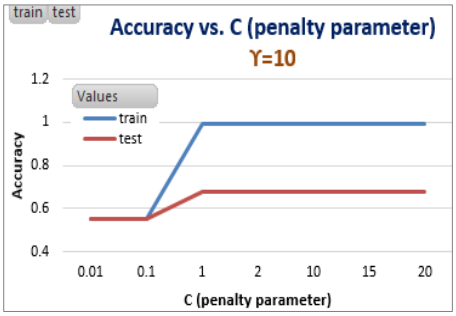
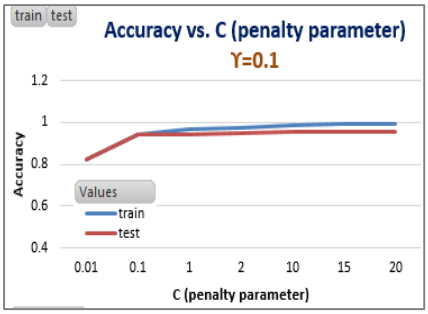


Fig-4.1c





Charts on left show train and CV accuracy as function of C (penalty term) for two different gamma values.

With smaller gamma, CV accuracy (red line) improves significantly.

4.2 ABALONE

I tried a varying range of C (penalty term) & gamma (kernel coefficient) combinations but due to lack of space, I am showing validation and learning curve only for best linear and RBF kernel model. However, all the results and hyperparameters range explored can be found in output/tune/SVM_abalone.xlsx.

- Fig-4.2a and Fig-4.2d show train and CV score as function of C for linear and RBF kernel. It is interesting to note that with Linear kernel, gap between train and CV score doesn't change as C increases, whereas with RBF kernel train-score improves as C increases because model is penalized more for misclassification and hence forced to fit training data better and eventually overfits. To combat this, value of C at the point of divergence was chosen (C=2).
- Fig-4.2b and Fig-4.2e gives learning curve for both models. With more samples both train and test scores improve. There's high std_dev in test scores even at larger train-set, which makes me nervous.
- Fig-4.2c and Fig-4.2f gives test/train time. Model with linear kernel took less time to train when compared to RBF model. This is because RBF kernel is more complex and hence takes more time. Test-times were quite close though.

{ Linear, C=3 }

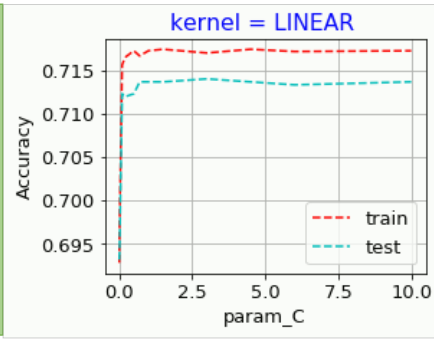


Fig-4.2a

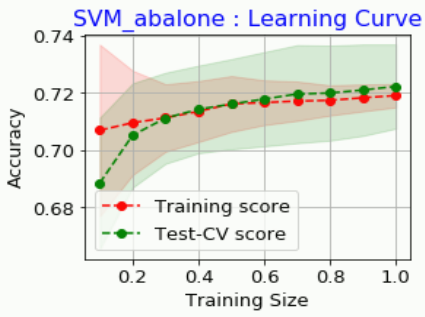


Fig-4.2b

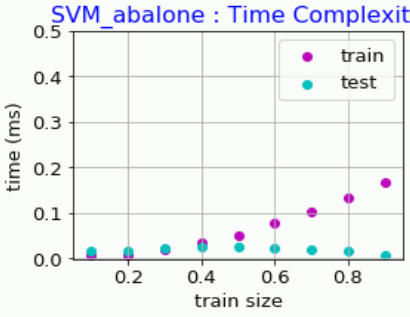


Fig-4.2c

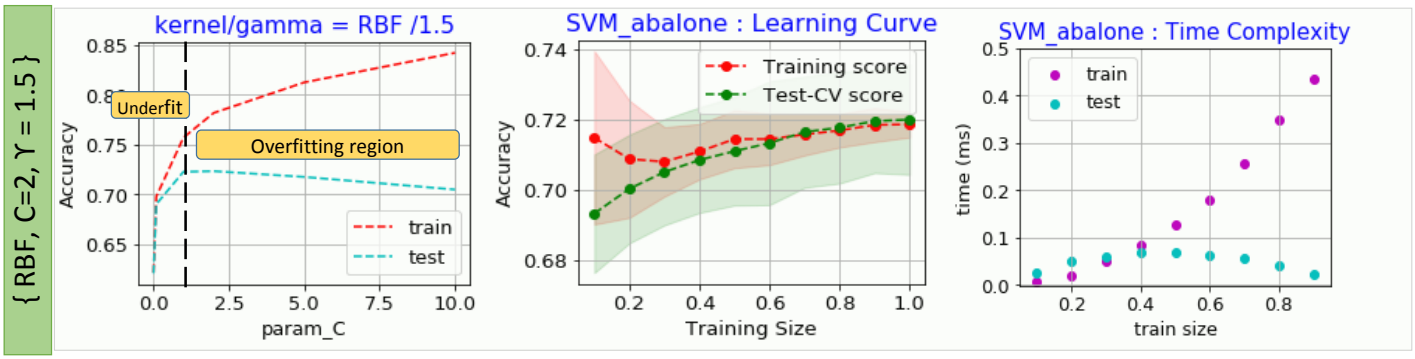


Fig-4.2d **Fig-4.2e** **Fig-4.2f**

5. Neural Networks (NN)

This experiment took me the most time as there were many hyperparameters to tune. In the end, I decided to focus on following parameters, which I think are critical in controlling model's complexity and achieve a right balance of bias vs. variance.

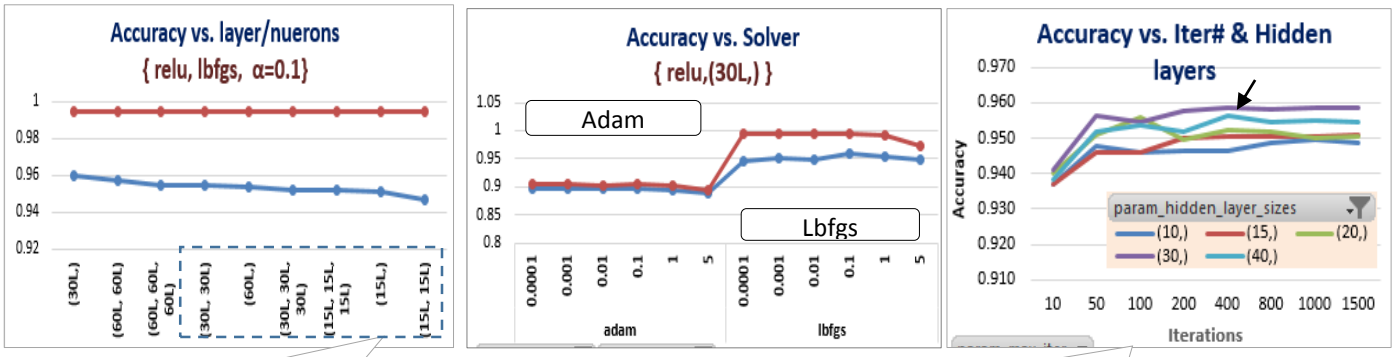
- Activation function – Since ReLu doesn't face 'vanishing-gradient' problem as with 'sigmoid' and 'tanh' function, it was my first choice. I chose to run experiment with sigmoid as well to see what difference it makes for my datasets.
- Solver – L-BFGS and Adam (optimized SGD) were explored.
- Number of hidden layers and neurons – Both these parameters influence model's complexity and help in combatting over/under fitting. I explored a range of neurons (guided by number of attributes in each dataset) and used this range to build single and multilayer (2, 3) models.
- Alpha - L2 regularization (keeps the weights small) to combat overfitting. Range explored [0.0001 – 1.0]
- Max_iterations – I used adaptive iteration, where training would terminate early if improvement in validation score is negligible (< 0.001) from one epoch to another, maxing out at 2000 iterations.

5.1 PHISHING WEBSITES

Model Complexity - Fig-5.1a shows train (red) and CV-accuracy (blue)

a) as number of hidden layers and neurons are varied.
b) as regularization parameter (alpha) is changed for different solvers – Lbfgs & Adam.
c) as a function of iterations for different layer/neurons.

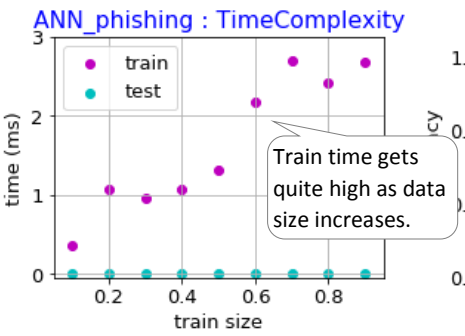
- Results with 1-layer/30-neurons were very similar to 2-layer/60-neurons. After evaluating learning curve for each, I chose former model as added complexity in 2nd model wasn't buying anything. Overly complex models (3 hidden layers) didn't generalize well and had poor CV scores.
- With high values of alpha (>1), scores began to degrade.
- L-BFGS solver gave better performance than 'Adam'. *(LBFGS performs better on small datasets)*
- Results with ReLu were slightly better than that with 'Sigmoid' activation.

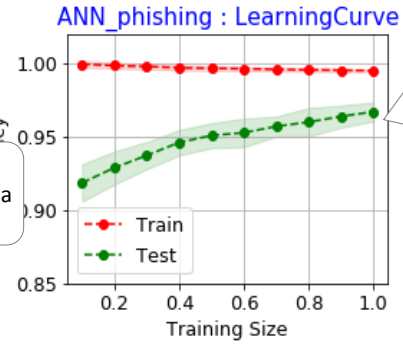


As number of layers/neurons increases model complexity increases, leading to more variance & lower test scores.

Fig-5.1a

Model with 30neuron/single layer gave best result, maxing out at 400 iterations.





best_model => { relu, lbfgs, (30,),α=0.1 }

Gap between train and test scores is higher than I would have liked. Model shows signs of variance – adding more samples and/or feature reduction may be helpful. Note this dataset has 30 features and only 2456 samples.

5.2 ABALONE

Here again L-BFS solver gave better performance compared to 'Adam'. Abalone is a small dataset and L_BFGS is known to perform better for small datasets. As can be seen from table below there were several models that gave quite close results. Following Occam's razor, I picked simpler model (M1- single layer) but I wanted to test both models (M1 & M2) on unseen data to be sure -*comparison shown below*

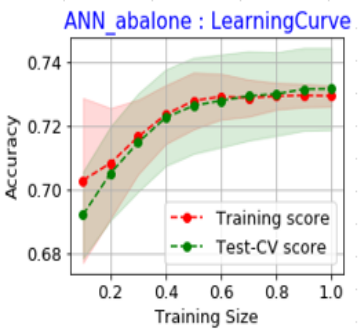
Model-1

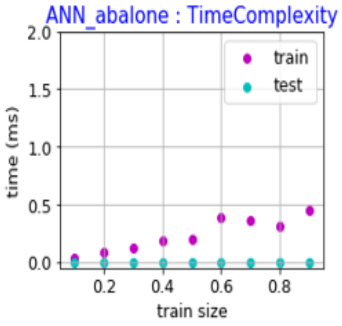
Model-2

mean fit time	mean score tim	mean test scor	mean train scor	activation	alpha	hidden layer size	solver
0.55760	0.00080	0.72973	0.73050	logistic	1	(5L, 5L)	lbfgs
0.17700	0.00000	0.72665	0.72896	logistic	1	(5L,)	lbfgs
0.51360	0.00020	0.72665	0.72819	logistic	2	(5L, 5L, 5L)	lbfgs
0.87320	0.00100	0.72665	0.73324	logistic	1	(10L, 10L)	lbfgs
1.34020	0.00040	0.72665	0.73366	logistic	0.001	(5L,)	lbfgs
0.91740	0.00060	0.72631	0.72776	logistic	2	0L, 10L, 10	lbfgs
1.01160	0.00080	0.72631	0.72802	logistic	2	(20L, 20L)	lbfgs
1.45000	0.00040	0.72631	0.73460	logistic	1	(20L, 20L)	lbfgs
0.56980	0.00040	0.72597	0.73153	logistic	1	(20L,)	lbfgs
0.73120	0.00000	0.72597	0.73597	logistic	0.01	(5L,)	lbfgs

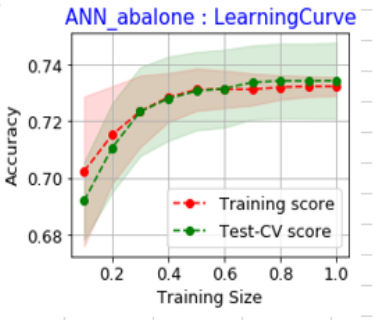
Learning curve below shows that model learns quickly as training-set size is increased from 10 to 60%. Beyond that, improvement in scores is not as rapid. My educated guess is that features in this dataset are not truly reflective of underlying concept. In addition to having more samples to learn from, we might have to do some feature engineering to improve the scores (reduce bias).

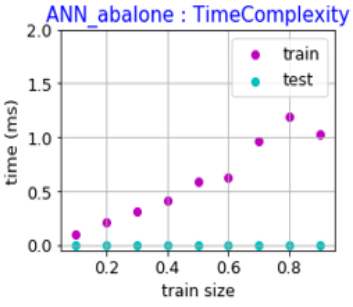
Model-1 (1Layer)





Model-2 (2Layers)





Clf_name	Train_Acc	Test_Acc	F1-score	Recall	Precision
Model-1	0.72802	0.73365	0.67891	0.73365	0.66925
Model-2	0.73178	0.73285	0.67922	0.73285	0.66328

Model-1gave very close results to Model-2 and takes much less time.

6 Conclusion

Below I have summarized key metrics such as classification accuracy, F1 measure, ROC-AUC for different classifiers for easy comparison. Models selected for each learning algorithm performed considerably better than the baseline (zero-R model). However, Phishing dataset achieves higher accuracy than Abalone. Reason for this could be

a) Phishing dataset has much more 'informative' examples and hence learner is able to learn underlying concept, whereas Abalone dataset either has '*irreducible*' noise or features captured in this dataset are not very informative in their current form, i.e. some feature engineering is required.

b) It is also possible that the way I bin the labels for Abalone is not optimal, though I tried several different bin-sizes but scores were always in low to mid 70s range.

Result summary

PHISHING WEBSITES

Rank		Accuracy		F1	Recall	Precision	AUC	time	
		train	test					train	test
2	KNN	0.9942	0.9783	0.9783	0.9783	0.9784	0.9771	0.00	0.05
3	SVM	0.9872	0.9756	0.9756	0.9756	0.9757	0.9759	0.04	0.01
5	DT	0.9820	0.9661	0.9661	0.9661	0.9661	0.9652	0.00	0.00
4	BOOST	0.9849	0.9742	0.9742	0.9742	0.9744	0.9747	0.34	0.02
1	ANN	0.9942	0.9810	0.9810	0.9810	0.9811	0.9814	2.80	0.00
	zero-R		0.5569	0.398	0.557	0.31	0.5		

<--- *baseline*

CV accuracy vs. training-set size

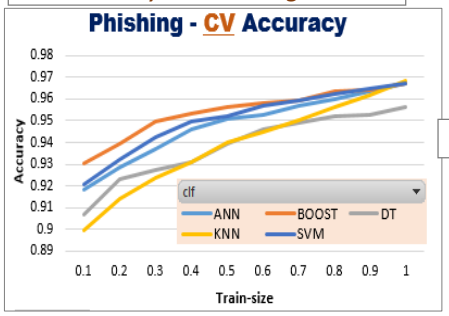


Fig-6a

ABALONE

ank		Accuracy		F1	Recall	Precision	time	
		train	test				train	test
3	KNN	0.7249	0.7305	0.6783	0.7305	0.6955	0.003	0.036
3	SVM	0.7605	0.7305	0.6754	0.7305	0.6684	0.298	0.063
5	DT	0.7263	0.7185	0.6591	0.7185	0.6354	0.009	0.000
1	BOOST	0.7294	0.7329	0.6823	0.7329	0.6952	0.762	0.037
2	ANN	0.7280	0.7327	0.6789	0.7337	0.6693	1.076	0.000
	zero-R		49.82	0.331	0.498	0.248		

<--- *baseline*

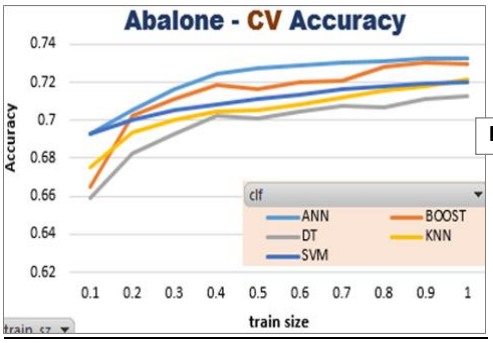


Fig-6b

From table above, we see 'No Free Lunch' theorem in action – '*no one model works best for every problem*'. For Phishing dataset, ANN gave the best accuracy/F1/AUC while for Abalone, Adaboost was the winner. However, both learners have high training time cost. So there's a tradeoff between predictive power and time complexity.

- For both datasets Adaboost gave better results than plain Decision Tree. This was in line with my expectation as Adaboost can reduce bias/variance by averaging over many weak learners.
- KNN was in top 3, which was bit of surprise for me. Since KNN doesn't employ regularization (and both datasets have redundant features) I wasn't expecting it to perform at par with SVM or Boosting. Moreover, in absence of domain knowledge I used standard distance function and yet results were surprisingly good.

Learning curve comparison:

- For Phishing dataset (Fig-6a): Boosting gives superior performance even when training-set size is small. But at 90% samples, all models give very similar performance except for Decision tree.
- For Abalone dataset (Fig-6b): ANN gives give good results even for small training-set size but Boosting catches up quickly as samples-size grows and gives best performance on unseen data.
- All models (for both datasets) show that accuracy improves with more samples. This tells me that models don't suffer from high bias (underfitting), otherwise adding more samples wouldn't have had any impact.

Note – due to lack of space Roc-Curves and learning-curve comparison for training-set is provided in summary_roc.xlsx

Reflection

1. Given that F1 score is very close to classification accuracy, it indicates to me that best model selection based on classification accuracy was a reasonable choice but in hindsight, I would try model selection for both metrics. This will be useful in case of unbalanced datasets.
2. I learned how crucial cross-validation is, especially when datasets are small like mine. In all cases, CV-scores (used in tuning phase) were a good estimation of error observed on unseen data.
3. Evaluating Complexity curves taught me how to strike a balance between variance and bias and apply Occam's razor principle. It also gave a taste of real world data. Eg: for many classifiers, learning curve was way different than what I had seen in theory and hence made me think critically and relate to dataset characteristic.
4. Learning curves gave a second chance to determine if selected model suffers from over/under fitting and if /how much adding more training example can help the model.