CS 7641: Supervised Learning Assignment

Ymakram3

Introduction

In this assignment we are analyzing a set of supervised machine learning algorithms on two different datasets to gain an intuition on how different algorithms perform. We are using Python, sickit-learn, and matplotlib libraries to perform the experiments. The experiments are performed in a set of Jupyter notebooks, a notebook per algorithm. The algorithms are Decision Trees, Neural Networks, Boosting, Support Vector Machines, and k-Nearest Neighbors.

We are using two datasets to evaluate the different algorithms. The first dataset is a tic-tac-toe end game dataset, with a board state as features, and win or loose as prediction. The second dataset is MNIST digits dataset, with a set of handwritten digits pixels as features, and numbers as classes.

Datasets

Tic-tac-toe End Game Dataset

https://www.openml.org/d/50

The dataset has 958 instances representing all valid end game configuration of tic tac toe games assuming x played first. Each instance has 9 features representing one tic-tac-toe square. Each feature can be one of {x, o, b} values. The value x means player x has taken the square, value o means player o has taken the square, and value b means a blank square.

The target label represents win or loose configuration with two possible classes {positive, negative}. The value positive represents a win for x, and value negative represent a loss for x. The class distribution is 65.3% positive and 34.7% negative. This is a slight imbalance that need to be addressed either with sampling, or some precision-based scoring.

This dataset was selected for the following reasons:

- Simplicity: The relatively small number of features and instances allows faster iterations on algorithms and experiment with API, and algorithm configuration, while not being trivial for meaningful results.
- Categorical: The features have string labels, which requires preprocessing for different algorithms to work.
- Complete: The dataset represents the full problem space of the game.
- Relatively small number of instances: It is interesting to compare how different algorithms perform on relatively small number of instances.

MNIST Database of Handwritten Digits

http://yann.lecun.com/exdb/mnist/

The MNIST database has a training set of 60,000 examples of handwritten digits, and a test set of 10,000 examples. The database has been preprocessed and normalized to 28x28 images with various grey levels generated by anti-aliasing. The number of features is 784 corresponding to pixel values in 28x28 images. The target classes are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} representing single digit numbers.

Class	Number of Instances
0	6903
1	7877
2	6990

3	7141
4	6824
5	6313
6	6876
7	7293
8	6825
9	6958
Total	70000

The dataset was selected for the flowing reasons:

- High Dimensionality: The dataset has 784 features, which is interesting to evaluate how algorithms are performing on high dimensional set.
- Relatively large number of instances: Interesting to compare the performance of different algorithms compared to the small number of instances in the first dataset.
- Relatively large of number of classes: Interesting to compare the performance of classifying 10 different classes compared to the two classes in the first dataset.

Decision Trees

Tic-tac-toe Dataset

The default DecisionTreeClassifier of sickit-learn generated a tree with maximum depth of 12 with no pruning. Performing a grid search across the two algorithms “gini”, and “entropy” where entropy is the information gain showed that “entropy” is performing slightly better on this dataset. It also showed that pruning decreases the performance.

Figure 1 shows the error rate against number of samples. The training loss is zero suggesting overfitting, while the validation score is increasing suggesting no convergence. The learning curve suggests a good validation accuracy at depth of 9, but when tried the test set performed a little bit worse with accuracy of 83.5% vs. 85.4 with no pruning.

Figure 1Learning Curve (no pruning)

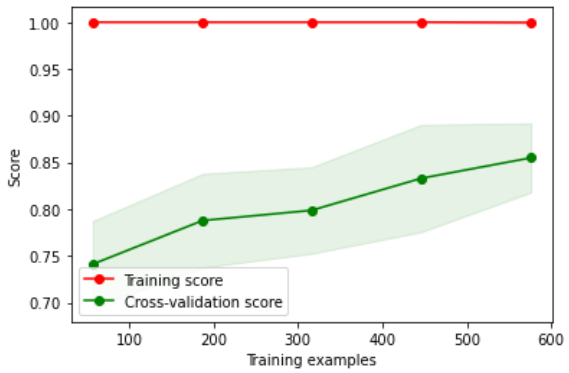


Figure 2 Validation curve (max_depth)

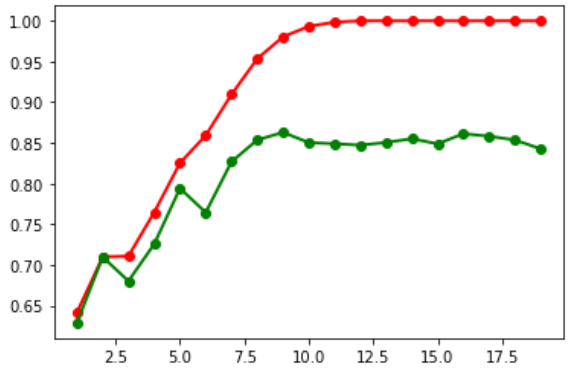
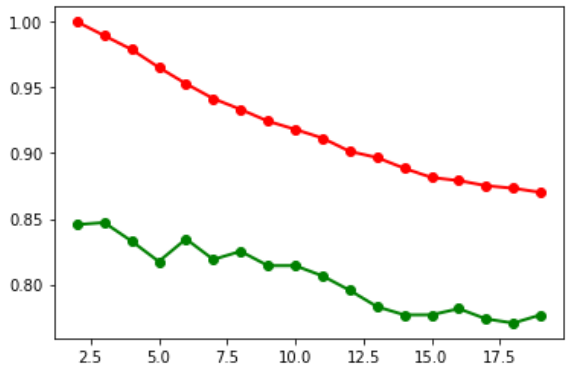


Figure 3 Validation curve (min_samples_split)



MNIST Dataset

The default sickit-learn classifier yielded a tree with depth of 44. With a randomized hyper parameter search RandomizedSearchCV on criterion {gini, entropy} and a max_depth between {1, 44} recommended a tree with depth 22.

The validation curve shows convergence at depth of 12. The tuned learner with entropy

criterion and depth of 22 yielded 87.5% prediction accuracy.

Figure 4 Learning curve (max_depth=22) pruning

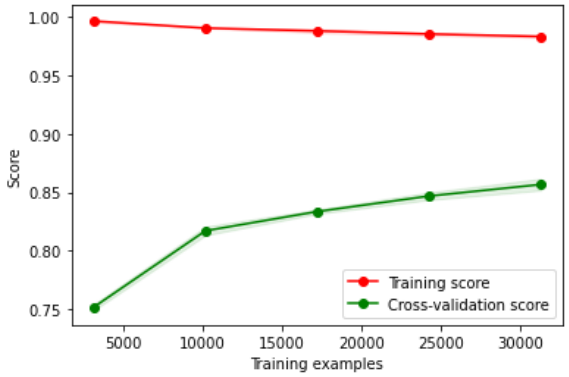


Figure 5 Validation curve (max depth)

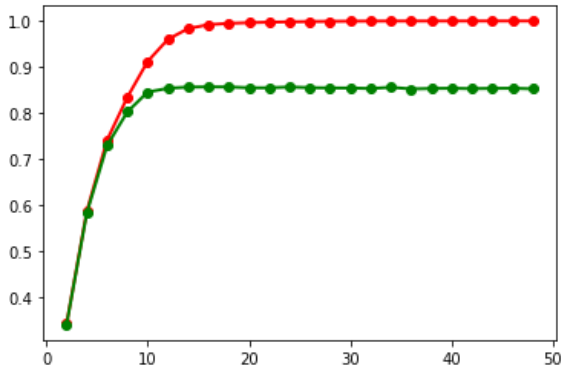
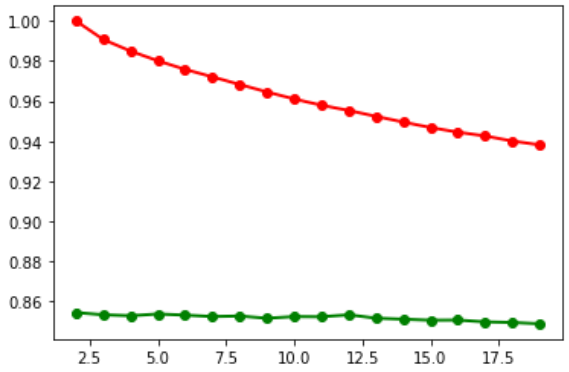


Figure 6 Validation curve (min samples split)



Analysis

The decision tree classifier is performing poorly on the tic-tac-toe dataset. This was surprising since the data seems to fit a decision tree, but the classifier was overfitting and generalizing poorly. Pruning did not help improve the prediction accuracy. The classification report shown below that the precision and recall on the 0 class is worse than the positive class.

	precision	recall	f1-score	support
0	0.74	0.76	0.75	102
1	0.89	0.87	0.88	214
accuracy			0.84	316
macro avg	0.81	0.82	0.81	316
weighted avg	0.84	0.84	0.84	316

On MNIST dataset, the decision tree classifier seems to overfit as well. Even with pruning the overfitting is starting at lower tree depths and the accuracy seems to converge at 87.5% prediction accuracy. The precision and recall showing better performance on some digits like 0, 1 and poor performance on digits like 5, 9.

	precision	recall	f1-score	support
0	0.94	0.92	0.93	2267
1	0.94	0.96	0.95	2603
2	0.87	0.85	0.86	2350
3	0.84	0.84	0.84	2383
4	0.86	0.87	0.87	2144
5	0.83	0.83	0.83	2107
6	0.89	0.90	0.90	2294
7	0.90	0.91	0.90	2455
8	0.84	0.82	0.83	2196
9	0.83	0.84	0.83	2301

The confusion matrix further shows which classes are mis-predicted, for example it shows that the digit 9 is frequently confused with 4.

```
[[2078    1   28   13   11   40   46   10   19   21]
 [   6 2493   18   22    6   12    7   12   23    4]
 [  19   34 2009   44   22   26   54   56   50   36]
 [  10   13   72 2006    8   93   21   46   65   49]
 [  11   10   25   16 1864   14   28   19   44  113]
 [  30   18   17  102   24 1752   44   13   58   49]
 [  17   16   23   15   47   51 2071    8   31   15]
 [   4   16   62   31   28   15    4 2232   13   50]
 [  22   33   52   82   37   57   31   22 1790   70]
 [  20   13   16   43  111   39   14   71   42 1932]]
```

Neural Networks

Tic-tac-toe Dataset

The default multi-layer perceptron implementation with stochastic gradient descent, and logistic activation performed moderately on the dataset with accuracy close to decision trees.

A grid search showed Adam solver with RELU activation performing much better.

Figure 7 MLP classifier with SGD, and logistic activation

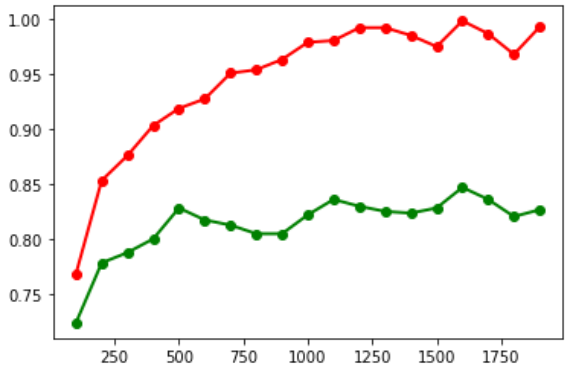
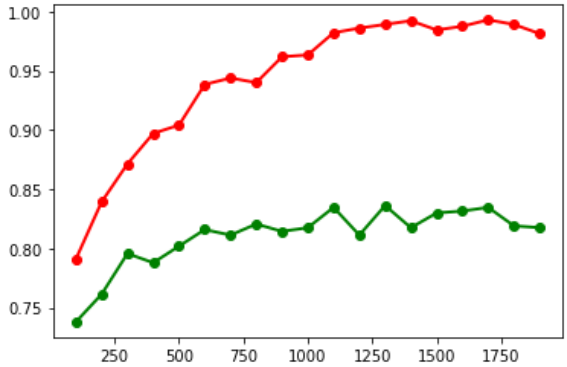
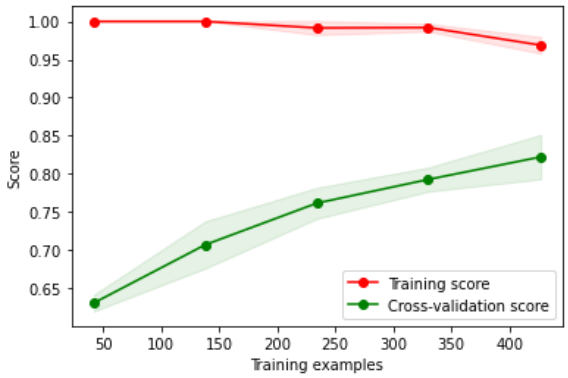


Figure 8 MLP classifier with Adam and RELU



Increasing the maximum iterations is causing overfitting in both solvers with slight improvement in validation accuracy. The Adam solver is learning faster, with more stable validation curve.

Figure 9 Learning curve Adam with RELU, max_iter 900



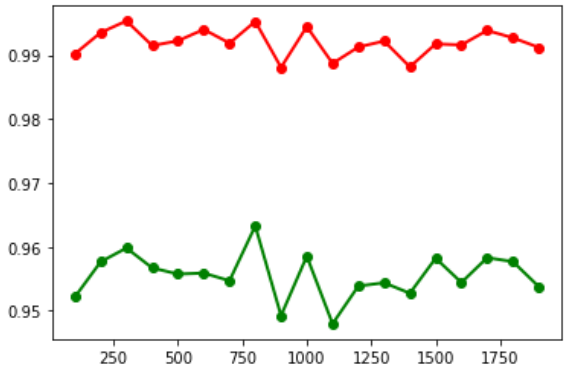
The grid search as well as the validation curve yielded a tuned max iteration of 900. The learning curve is showing improvement with more data and suggests the learner can still improve with more data.

MNIST Dataset

The default MLP classifier with stochastic gradient descent solver and logistic regression performed fine on the MNIST dataset with 95% cross validation score. A grid search showed that Adam solver with RELU activation, and 800 maximum iteration is performing best.

The validation curve of the SGD showing instability in validation accuracy beyond 800 iterations suggesting overfitting, with the maximum iteration can be used for regularization.

Figure 10 Validation curve SGD with max iterations



The validation curve with Adam solver and RELU has a more stable validation curve, but the performance is still best at 800 iterations.

Figure 11 Validation curve Adam with max iterations

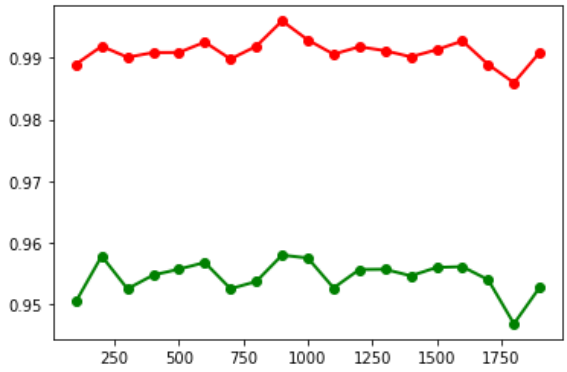
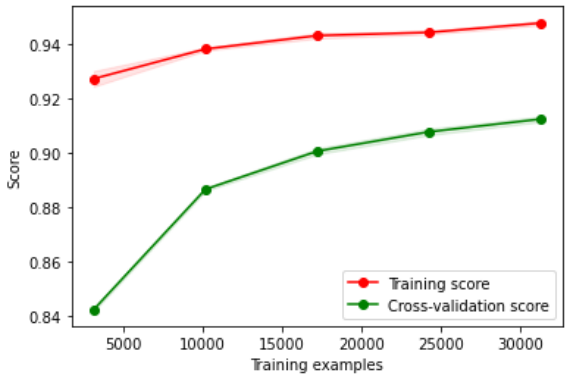


Figure 12 Learning curve Adam, RELU, max iter 800



Analysis

The MLP is performing poorly on tic-tac-toe dataset. The learning curve is suggesting overfitting, with slight improvement as the learner gets more instances. This suggests the algorithm is not dealing well with small datasets. The accuracy, precision, and recall is performing slightly less than the decision tree.

	precision	recall	f1-score	support
0	0.76	0.71	0.73	102
1	0.86	0.89	0.88	214
accuracy			0.83	316
macro avg	0.81	0.80	0.80	316
weighted avg	0.83	0.83	0.83	316

The multi-layer perceptron, while still a simple network is performing well on the MNIST dataset. The network is learning the data fast, with slight overfitting that could be regularized with limiting the maximum iterations, and early stopping. There is still room for improvement with more complex networks but was beyond the time available for this exercise. The class precision and recall variance is much lower than decision trees. The confusion matrix is showing that the maximum confusion with 44 errors is between 9 classified as 4.

	precision	recall	f1-score	support
0	0.98	0.97	0.97	2267
1	0.98	0.98	0.98	2603
2	0.96	0.96	0.96	2350
3	0.96	0.95	0.95	2383
4	0.97	0.96	0.97	2144
5	0.95	0.95	0.95	2107
6	0.97	0.98	0.98	2294
7	0.96	0.97	0.97	2455
8	0.95	0.94	0.94	2196
9	0.95	0.96	0.95	2301
accuracy			0.96	23100
macro avg	0.96	0.96	0.96	23100
weighted avg	0.96	0.96	0.96	23100

```
[[2196    1   12    0    4    6   15    7   24    2]
 [   1 2556   12   10    1    1    1    6   14    1]
 [  10    8 2261   13    7    4    5   21   16    5]
 [   1    0   23 2267    2   36    2   18   18   16]
 [   1    4    5    2 2060    1    8   12    7   44]
 [   5    3    2   38    4 2008   22    2    9   14]
 [   4    3    8    0   10    9 2249    1    9    1]
 [   2    8   22    7    6    1    0 2384    3   22]
 [  17   13   13   22   10   23   12    8 2057   21]
 [  12    4    0   13   14   14    1   25   17 2201]]
```

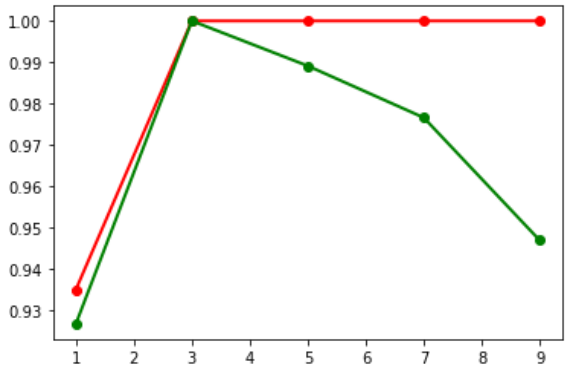
Boosting

Tic-tac-toe Dataset

In this experiment, we are using AdaBoost algorithm with Decision Tree as base estimator. Using boosting allows us to be aggressive with pruning. We have seen in the decision tree experiment that the tree was overfitting, and pruning did not improve the performance. In this experiment with a grid search suggested an aggressive pruning with max depth of 3 compared to 12 that was used with the single decision tree.

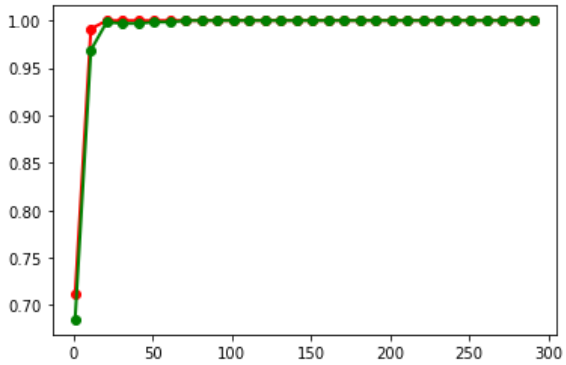
The validation curve with max depth showing perfect performance at depth of three, while smaller depth caused underfitting, and larger depth caused overfitting.

Figure 13 Validation curve with max depth



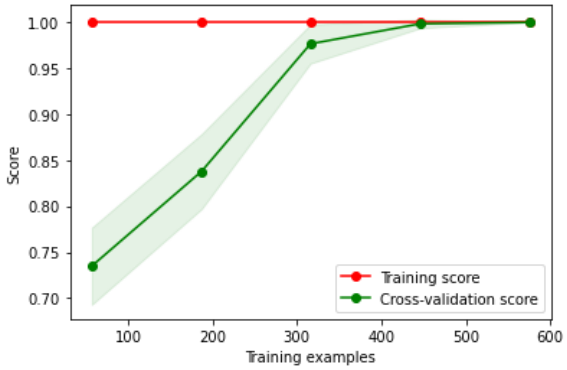
The number of estimators validation curve shows sharp rise of performance till 30 estimators, then sustained performance after.

Figure 14 Validation curve with numer of estimators



The learning curve is also showing a very good performance with perfect prediction at 450 instances.

Figure 15 Learning curve, max depth 3, n_estimators 201

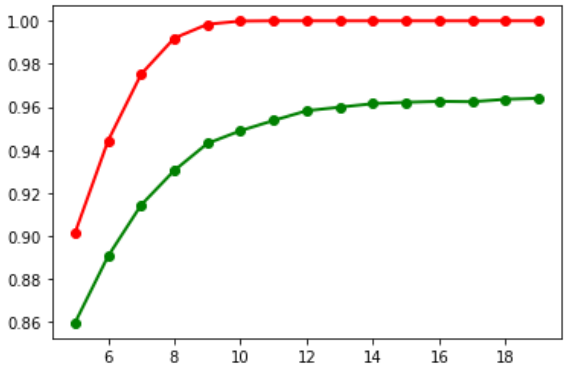


MNIST Dataset

AdaBoost algorithm did perform much better than the decision tree with the MNIST dataset. A grid search suggested a max depth of 9, and 601 estimators.

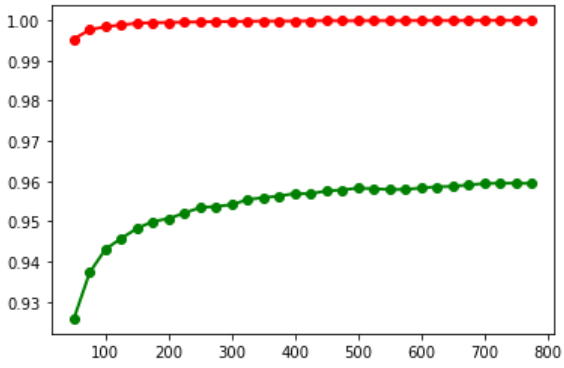
The validation curve against max depth settings showing overfitting starting at depth of 9 and underfitting before that. The overall performance is comparable with MLP neural network, but learning and prediction is much slower in the case of AdaBoost.

Figure 16 Validation curve max depth



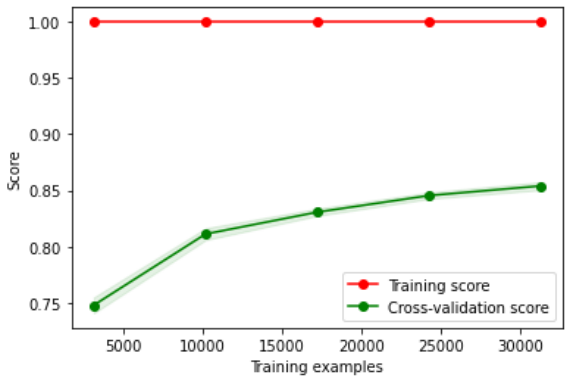
The number of estimators is showing overfitting as well at 200 estimators. Having more estimators did not improve the performance.

Figure 17 Validation curve with number of estimators



The learning curve is showing the overfitting clearly starting from small number of instances, and continuing as the algorithm sees more instances.

Figure 18 Learning curve max depth 9, n_estimators 601



Analysis

The boosting significantly improved the performance and eliminated the overfitting with the tic-tac-toe dataset.

The AdaBoost algorithm performed perfectly on both the training, and test sets and was able to achieve 100% score over all metrics. The aggressive pruning from 12 to 3 was effective in eliminating the overfitting seen with the decision tree classifier.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	102
1	1.00	1.00	1.00	214
accuracy			1.00	316
macro avg	1.00	1.00	1.00	316
weighted avg	1.00	1.00	1.00	316

On the other hand the AdaBoost classifier was not able to handle the overfitting at different values of pruning and increasing the number of estimators did not help as well. This suggests that AdaBoost was not able to deal with high dimensionality of the dataset. Further investigation is needed to see if other boosting algorithms may perform better like gradient boosting.

Although the overall performance is comparable to MLP neural network, but the model is performing much worse on some classes like {8, 9} with 93%, and 91% respectively.

	precision	recall	f1-score	support
0	0.99	0.97	0.98	2267
1	0.99	0.99	0.99	2603
2	0.95	0.95	0.95	2350
3	0.95	0.95	0.95	2383
4	0.96	0.96	0.96	2144
5	0.95	0.96	0.95	2107
6	0.99	0.97	0.98	2294
7	0.97	0.96	0.96	2455
8	0.93	0.95	0.94	2196
9	0.91	0.96	0.94	2301
accuracy			0.96	23100
macro avg	0.96	0.96	0.96	23100
weighted avg	0.96	0.96	0.96	23100

```
[[2194    0   16    2    5   21    6    2   18    3]
 [   0 2566    8   10    2    2    1    4    4    6]
 [   1    8 2243   16    9    5    9   15   38    6]
 [   3    1   25 2258    1   30    3   14   27   21]
 [   2    2    7    0 2055    0    1    5    5   67]
 [   3    2    1   31    3 2018    8    1   21   19]
 [   0    1   10    2   15   18 2231    0   16    1]
 [   1    6   25    3   10    2    0 2345    7   56]
 [   1    7    9   32    9   22    2    5 2080   29]
 [   6    7   10   15   23    4    0   15   12 2209]]
```

Support Vector Machines

Tic-tac-toe Dataset

In this experiment, SVM implementation with scikit-learn library SVC was used with three kernels (linear, poly, rbf)

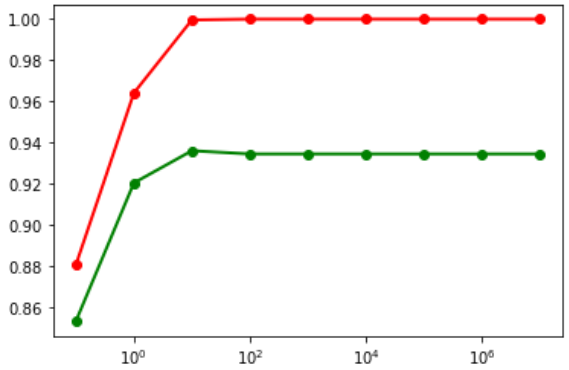
The linear kernel did not perform well on the dataset suggesting the data is not linearly separable.

Poly, and rbf kernel performed comparably, but the poly kernel training was much faster.

A grid search across linear, poly, with different values of C, and gamma suggested a poly kernel with gamma 0.1 and C 100000.

The validation curve against different values of C parameter showing overfitting starting at values of C greater than 10, and underfitting with lower values.

Figure 19 Validation curve poly kernelagainst C values



Gamma values showing a significant jump at values greater than 0.01, and overfitting at values greater than 1.

Figure 20 Validation curve ploy kernel against gamma values

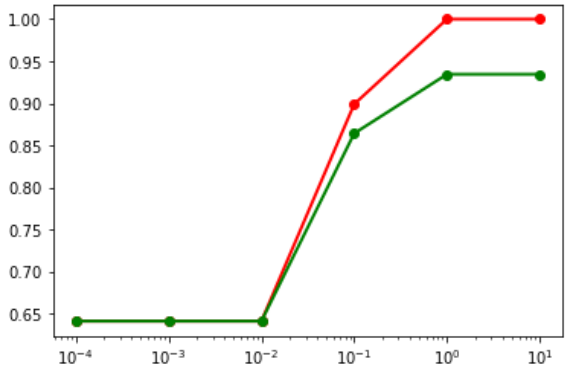
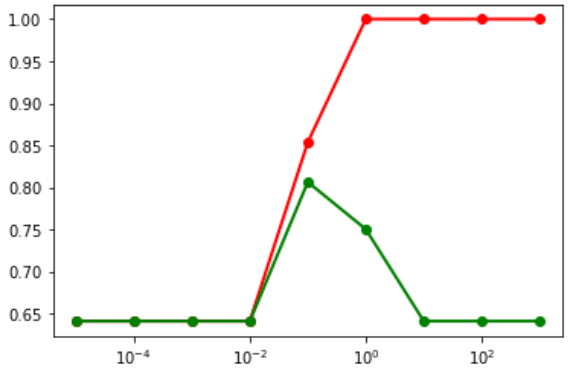


Figure 21 Validation curve rbf against gamma



The learning curve with the grid search suggested poly kernel, C=100000, gamma 0.1 is still showing overfitting on the dataset starting from low number of instances.

Figure 22 Learning curve poly kernel, c 100000, gamma 0.1

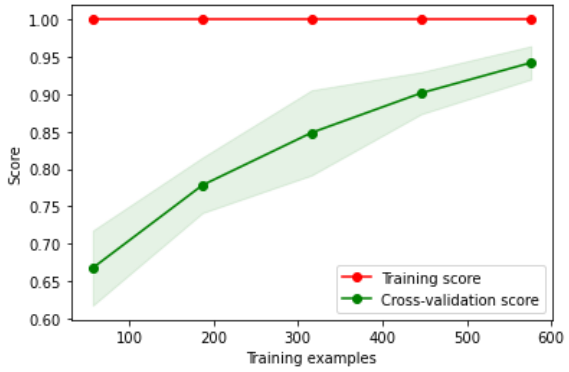
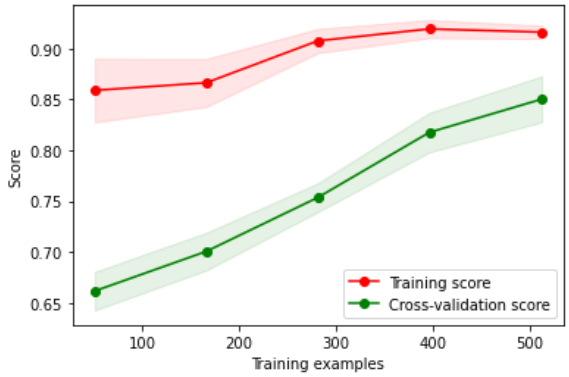


Figure 23 Learning curve, rbf kernel, C 100000, gamma 0.01



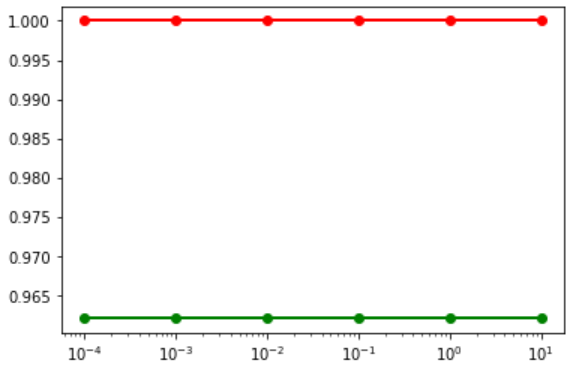
MNIST Dataset

In this experiment, {poly, rbf} kernels have been tried with different {C, gamma} with scikit-learn SVC classifier.

A grid search across kernels and candidate C, gamma values suggested polynomial kernel with a gamma value of 0.01 as the best estimator.

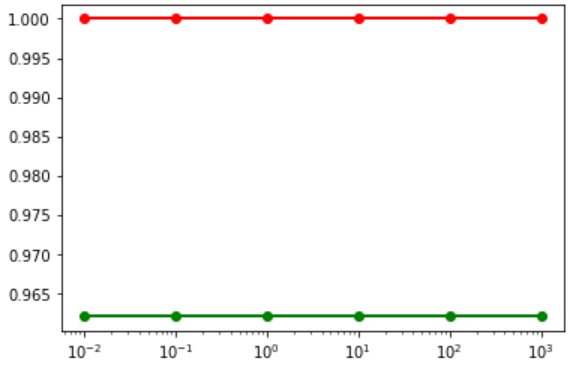
The validation curve of SVM performance with poly kernel, and different values of gamma that on this dataset the gamma values did not have an impact on performance.

Figure 24 Validation curve, poly kernel against gamma



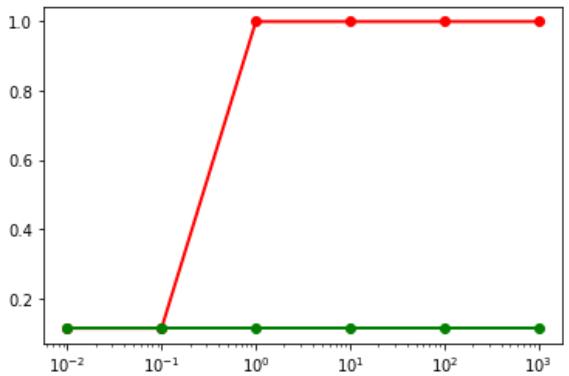
Performing the same experiments for C values, and the output was a constant as well with no impact from different C values.

Figure 25 Validation curve, poly kernel against C values



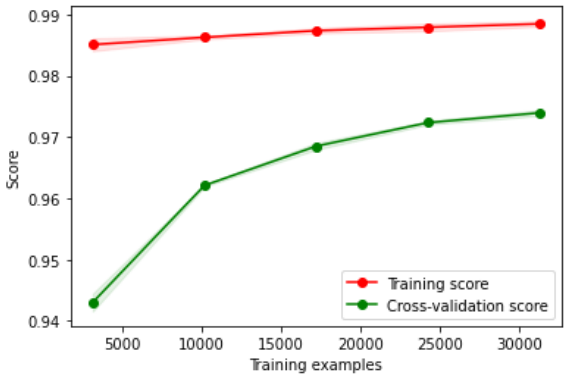
The RBF kernel seems to perform better on the training set, but no impact on the validation performance for different C values.

Figure 26 Validation curve RBF kernel against C values



The learning curve for the best estimator with a polynomial kernel, and gamma 0.01 is showing slight overfitting with more data.

Figure 27 Learning curve, poly kernel, 0.01 gamma



Analysis

SVM is showing very good performance on the tic-tac-toe dataset with accuracy of 97%. Other metrics is very good as well except for the precision of the negative class with 94%.

	precision	recall	f1-score	support
0	0.94	0.98	0.96	102
1	0.99	0.97	0.98	214
accuracy			0.97	316
macro avg	0.97	0.98	0.97	316
weighted avg	0.98	0.97	0.97	316

Despite the overfitting, SVM is performing well on the test set with 98% overall accuracy.

The variance in classification precision is very good as well, ranging from 97% to 99%, although the errors seems a little bit odd with the top two confusions are between {3, 5} and {3, 8}

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2267
           1       0.98      0.99      0.99      2603
           2       0.97      0.97      0.97      2350
           3       0.97      0.97      0.97      2383
           4       0.98      0.98      0.98      2144
           5       0.97      0.97      0.97      2107
           6       0.99      0.99      0.99      2294
           7       0.98      0.98      0.98      2455
           8       0.97      0.96      0.97      2196
           9       0.97      0.96      0.97      2301

    accuracy                           0.98     23100
   macro avg       0.98      0.98      0.98     23100
weighted avg       0.98      0.98      0.98     23100

[[2236    1    7    0    3    3    9    2    5    1]
 [   0 2582    7    5    0    0    0    5    2    2]
 [  11   13 2286    3    4    4    5   11    9    4]
 [   3    1   28 2305    1   17    0    5   14    9]
 [   4    4    4    2 2100    0    2    8    3   17]
 [   6    5    3   21    4 2042   10    0    9    7]
 [   1    1    2    0    8    9 2268    0    3    2]
 [   1    9   19    2    9    1    0 2401    2   11]
 [   4   11    8   20    4   18    4    8 2111    8]
 [  12    8    4   15   16    5    0   15   10 2216]]
```

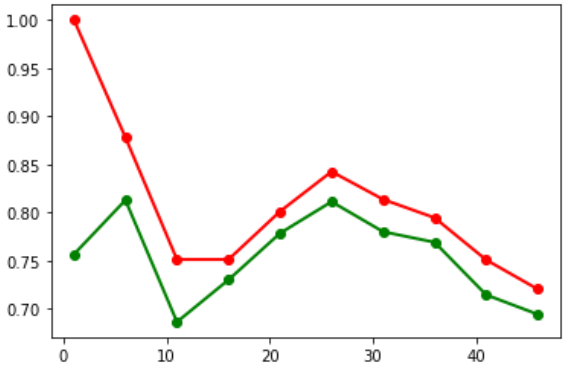
k-Nearest Neighbors

Tic-tac-toe

In this experiment, KNN classifier of scikit-learn library with both uniform and weighted classifiers with varying number of k.

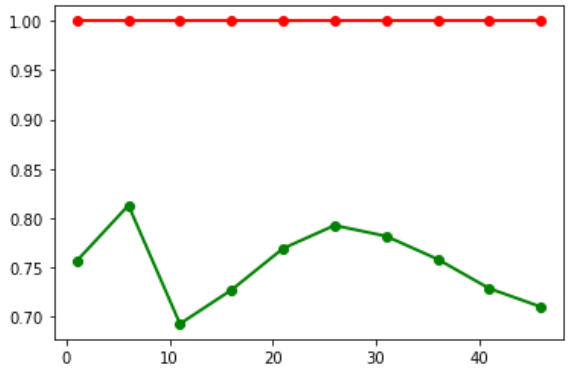
The validation curve of different value of K with a uniform classifier showing best performance at k=25

Figure 28 Validation curve uniform against k



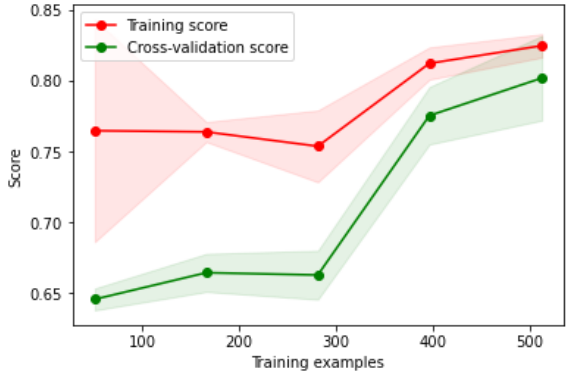
The weighted classifier performed worse with overfitting across all values of k

Figure 29 Weighted KNN against k



The learning curve of the uniform classifier with k=25 is showing high variance across the training and validation folds.

Figure 30 Learning curve uniform k=25

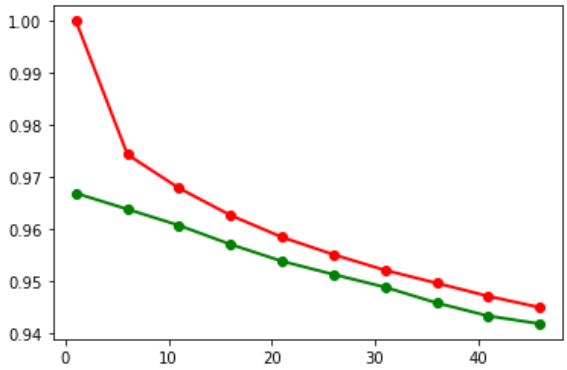


MNIST Dataset

The MNIST dataset was trained with KNN classifier with both uniform and weighted K and different numbers of k.

With the uniform classifier, the performance seems to decrease with larger numbers of k, but with less overfitting at k=6

Figure 31 Validation curve, uniform against k



The weighted classifier seems to overfit across all numbers of k

Figure 32 Validation curve, weighted KNN

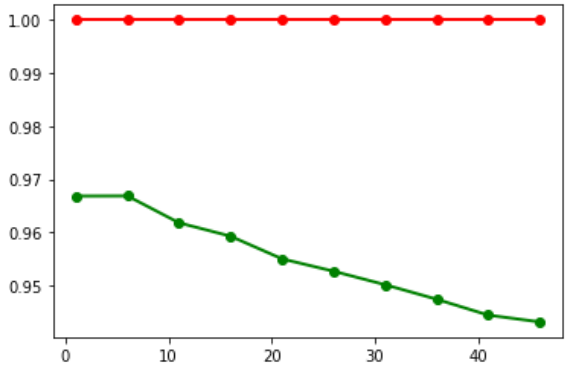
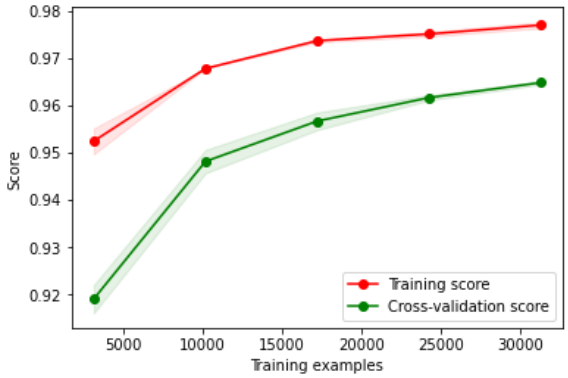


Figure 33 Learning curve, uniform KNN, k=6



Analysis

KNN is performing badly on the tic-tac-toe dataset. The number of instances did not help the algorithm neither. The precisions of the positive class that have a higher population is even lower. Out of the five classifiers examined, KNN performed the worst.

	precision	recall	f1-score	support
0	0.83	0.47	0.60	102
1	0.79	0.95	0.86	214
accuracy			0.80	316
macro avg	0.81	0.71	0.73	316
weighted avg	0.80	0.80	0.78	316

KNN have a moderate performance on the MNIST dataset. Although the average accuracy is 97% but the variance in class precision is high ranging from 95% for classes like {1, 9} and 99% for classes like {8}. The learning curve is showing signs of overfitting as well.

	precision	recall	f1-score	support
0	0.98	0.99	0.99	2267
1	0.95	1.00	0.97	2603
2	0.98	0.96	0.97	2350
3	0.97	0.96	0.97	2383
4	0.98	0.97	0.97	2144
5	0.97	0.97	0.97	2107
6	0.98	0.99	0.98	2294
7	0.96	0.97	0.97	2455
8	0.99	0.93	0.96	2196
9	0.95	0.96	0.96	2301
accuracy			0.97	23100
macro avg	0.97	0.97	0.97	23100
weighted avg	0.97	0.97	0.97	23100

```
[[2251    1    3    0    1    1    6    1    1    2]
 [   0 2590    4    2    1    1    0    4    0    1]
 [  15   29 2245    9    3    1    9   29    5    5]
 [   2    4   17 2298    1   22    2   13   10   14]
 [   3   17    1    1 2074    0    5    6    1   36]
 [   5    5    0   23    5 2036   20    0    5    8]
 [   6    4    0    0    4    7 2273    0    0    0]
 [   2   36    9    1    3    0    0 2381    1   22]
 [   8   25    7   31    8   37    9   13 2038   20]
 [   9    9    2   12   23    3    1   28    1 2213]]
```

Conclusion

After examining the performance of the different algorithms showed that there is no silver bullet nor a free lunch. Each algorithm had different characteristics dealing with dataset features.

The first dataset we used had a relatively small number of instances with categorical attribute. DT, NN, and KNN tended to overfit on this dataset with poor learning scores.

Boosting on the other hand learned the data perfectly, and was able to have 100% prediction accuracy, and precision.

SVM performed very well too with a polynomial kernel.

Algorithm	Accuracy	F1-score
DT	84%	81%
NN	83%	80%
Boosting	100%	100%
SVM	97%	97%
KNN	80%	73%

The second dataset had a different challenge. The number of instances were 70,000 giving more data for the algorithm to learn but strains the learning time performance. The high dimensionality of the dataset is another challenge with the chosen MNIST dataset.

The best error rate on the dataset achieved in this experiment was 2% using SVM. This is far from the start of the art achieved with neural networks designed for high dimensionality like CNN, but SVM has a descent performance with relatively low training costs.

Algorithm	Accuracy	F1-score
DT	88%	87%
NN	96%	96%
Boosting	96%	96%
SVM	98%	98%
KNN	97%	97%