Supervised Learning Algorithms for Letter Recognition and Car Evaluation

Surabhi Amit Chembra
surabhichembra@gatech.edu

Abstract

This paper applies supervised learning algorithms of Decision tree, neural network, boosting, support vector machines and K-nearest neighbors on two distinct datasets: letter recognition and car evaluation. Weka GUI was used to build and test the models. The performance of these models is recorded, analyzed and compared in this paper.

Dataset description

Both the datasets are classification problems and are downloaded from the UCI machine learning repository. The **letter recognition dataset** (UCI, n.d)[1] has 20000 instances and 16 attributes. Statistical moments and edge-counts mainly constitute the attributes, and each attribute is scaled to a range of integer values from 0 to 15. Each of the instances can be classified into one among the 26 capital letters (A to Z). As in Fig.1, the class distribution is **balanced** with an average of 750 instances per letter. The **car evaluation dataset** (UCI, n.d)[2] is relatively smaller with 1728 instances and 6 attributes. The attributes have 3 or 4 nominal values each and together helps in evaluating the quality of the car prior to purchase. As in Fig.2, the instances are classified into 4 class values: unacceptable (blue), acceptable(red), good(cyan) and very good (dark green). The dataset is **unbalanced** with around 70% of instances belonging to unacceptable class, 22% being acceptable and 4% each belonging to good and very good classes. Both the datasets have no missing attribute values.

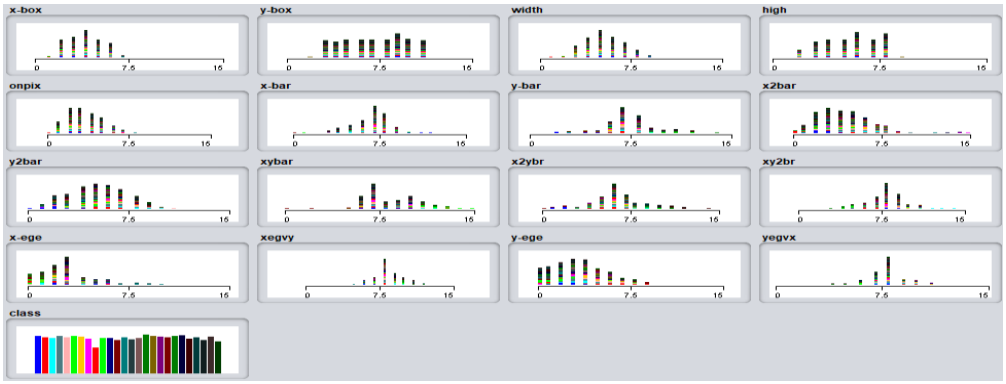

Fig.1.Distribution of classes for each attribute in Letter recognition dataset

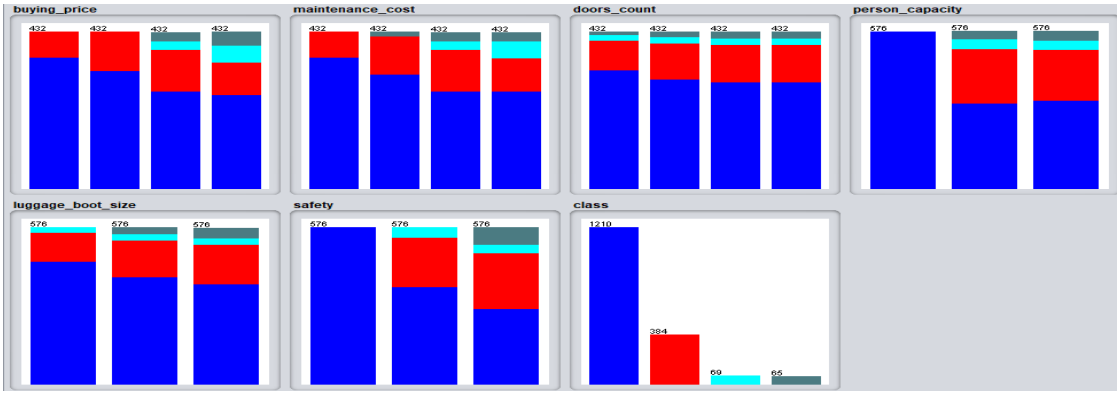

Fig.2.Distribution of classes for each attribute in Car Evaluation dataset

Why are the datasets interesting?

From application point of view: Recognizing handwritten letters helps to decipher handwritten notes and store the content electronically online, thus avoiding the loss of knowledge when the paper-notes are lost. More the notes stored online, lesser the trees cut to produce paper needed for individual hard-copies. Moreover, AI agents capable of understanding handwritten notes can interact better with humans. Thus, this dataset is interesting and useful from

the application point of view. Buying a car of unacceptable quality is a lossy transaction and leads to safety problems like accidents. The car evaluation dataset helps in building a model that can predict whether the car to be purchased is of acceptable quality. This model is thus an interesting application which proves useful to laymen and large-scale car-dealers alike.

From Machine Learning Point of view: Initially, I considered **several datasets** and chose four among them with different content, sample size, features count and other traits like being balanced. They were letter recognition, car evaluation, phishing and abalone datasets from UCI machine learning repository. After running the five supervised learning algorithms on each of them, I chose **Letter recognition** data set because it showed commendable performance (above 95%) on cross-validation accuracy on KNN, SVM and Boosting, while giving relatively poor performance on Decision Tree and ANN (below 86%). Similarly, **Car evaluation** dataset gave cross-validation accuracy of 98.6% with ANN and only 91.7% with KNN. These differences in performances were intriguing and it was interesting to find out why the different algorithms behaved differently with these datasets. Moreover, I wanted to investigate how the large number of classes (26) in the letter recognition dataset affects its performance on different algorithms. Also, it was exciting to find out the performance on which algorithms will be most affected by the small training data size in car evaluation dataset and why.

Regarding other datasets, **Abalone dataset** had 4K samples with 9 attributes and 29 classes. The sheer number of classes with very less amount of training data caused it to perform poorly and gave cross-validation accuracy of below 30% on all algorithms. **Phishing** dataset had 11000 instances with 31 attributes and 2 classes, and it performed very well on all algorithms, giving cross-validation accuracy of above 95% on all algorithms.

Methodology

Both the datasets were preprocessed using Weka GUI, and the steps are explained in README. The preprocessing involved removing duplicate entries, randomizing the order of instances in the data, and dividing into training and testing sets using a 70:30 split. **10-fold cross-validation** was performed on the training set and the accuracy metric for training set and cross-validation set were recorded to plot the model-complexity curve. Cross-validation helps in assessing the predictive power of the model on unseen data and thus helps in choosing a **generalized model** rather than a model overfitted to the training data. The **CVParameterSelection** and **gridSearch** filters were used to find the best-fitting hyperparameters for each dataset-algorithm pair. Weka GUI was used to build models and classify data points, and MS excel was used to plot model complexity plots and learning curves.

1.Decision Tree

J48 classifier in Weka GUI was used to implement Decision Tree. This classifier is based on C4.5 algorithm and is an extension of ID3. The algorithm uses **entropy**, which is a measure of data impurity, to calculate information gain. **Information gain** of an attribute tells how effectively that attribute partitions the instances according to the target class. To account for attributes like date that is specific for each training instance, **SplitInformation**, which conveys how broadly and uniformly the attribute splits data, is used with Information gain. This gives a better metric called **GainRatio** for best-attribute selection. At each node, the attributes with above-average information gain are selected, GainRatio test is applied on them and the attribute with maximum Gain Ratio is chosen to split the instances on. **Overfitting** happens when the model achieves high accuracy in in-sample testing (training set) by accommodating noise and coincidental irregularities but shows relatively poor performance in the validation set (out-of-sample testing). It happens in model-complexity plot (Fig.3) where training set accuracy increases while validation set accuracy decreases. **Pruning** is used to avoid overfitting by reducing tree-size and generalizing the tree. In J48 Weka classifier, pruning can be performed by adjusting **leaf-size** (min_num_obj) or by tuning **confidence value**. Adjusting leaf-size is pre-pruning and it stops the growth of a decision-tree path when the number of instances associated with that path is less than or equal to the specified leaf-size. J48 uses **Rule post-pruning** where decision tree paths are converted into rules and the rule preconditions are pruned if the estimated rule accuracy is not reduced. C4.5 estimates the rule accuracy pessimistically by considering the lower-bound estimate for a given confidence interval. Lesser the confidence value, higher the pruning incurred.

Using GridSearch and CVParameterSelection, the best-fit values for leaf-size and confidence value were obtained as 1 and 0.45 respectively for Letter Recognition dataset. For car evaluation dataset, the best confidence value was 0.55 and leaf size was 1. The performance for various leaf-size values were recorded to plot the model-complexity curve

(Fig.3). In Fig.3, the **underfitting region** is where the training accuracy and the cross-validation accuracy are both low with the later one more lesser. This region characterizes **high bias** caused due to **over-pruning**. The intuitive reasoning behind this bias problem is that when the leaf size is very high, the decision tree is highly pruned. For example, when the leaf size is set as 10 during training, if any 10 instances come down the same decision tree path, the path stops growing and they all get lodged at the same leaf node, irrespective of whether they can be categorized as same or require further classification. This can cause dissimilar training instances to be accumulated at each leaf node, making the tree **simpler but less accurate**. If **overfitting** is present, when the pruning is substantially decreased, the cross-validation accuracy will fall at the cost of training set accuracy. However, for this dataset, even when the leaf size is set as 1, the cross-validation error did not decrease. Thus, there is **no overfitting or variance problem** even in the absence of pruning in both datasets. Absence of random noise and coincidental regularities is the reason for the absence of overfitting.

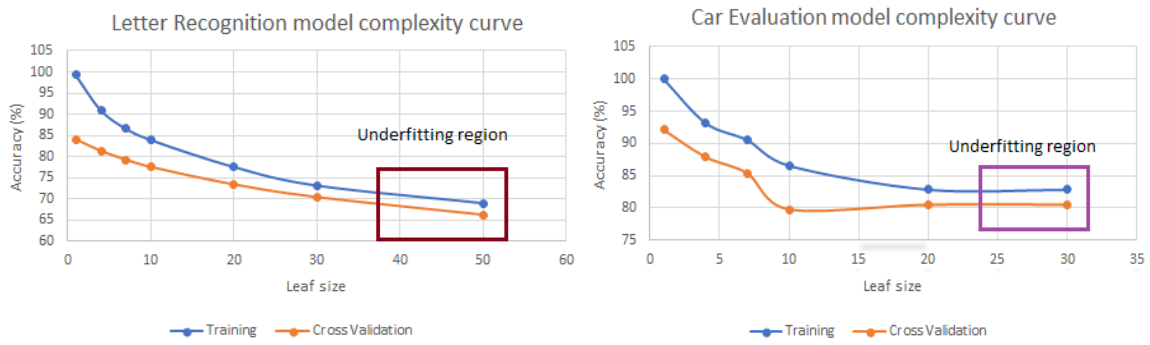

Fig.3. Model complexity curves for Letter Recognition dataset[LEFT] and Car Evaluation dataset[RIGHT]

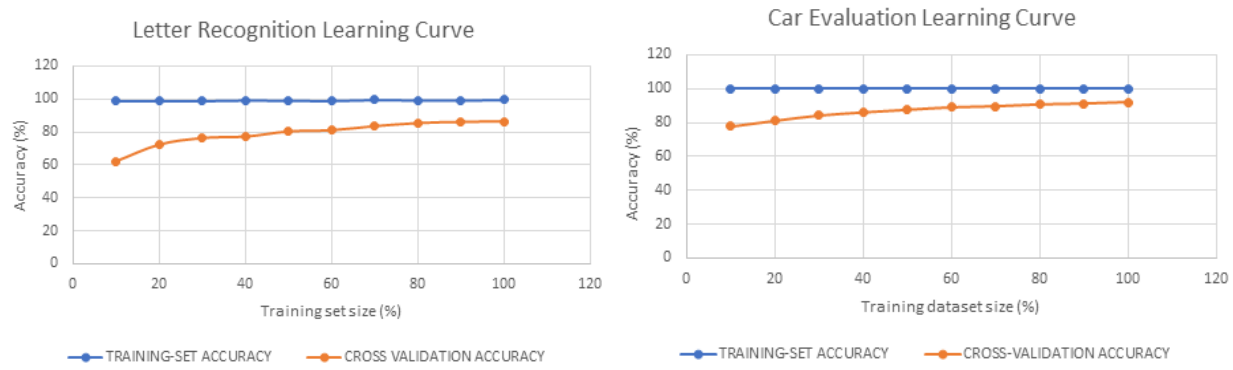

Fig.4. Learning curves for Letter Recognition dataset[LEFT] and Car Evaluation dataset[RIGHT]

Since best-fitting hyperparameters are chosen to plot learning curves, it does **not** show any **high bias problem**. This is because the chosen hypothesis can accommodate the training data complexity. If the hyperparameters were selected to facilitate **high pruning** such that the built model is too simple to accommodate the required predictiveness, the accuracy for both training and cross-validation sets will **flatten ou**t at a lower accuracy value, unlike here. The training set accuracy is high throughout because there is **no pre-pruning** and only nominal post-pruning (using rule based pruning and confidence value). The cross-validation accuracy is initially low because the small size of training set does not build a model sufficient enough to classify the previously-unseen validation set. However, as the size of training set increases, the cross-validation accuracy shows great improvement as the predictive model is fed more noise-free training data thus making it robust and accurate. Since the two accuracies have plateaued out at around 90% of the data in both plots, it can be inferred that there is no significant high variance problem. So, the amount of training data and the chosen hypothesis are good enough for decision tree on both datasets.Since both the datasets are devoid of noise and the resultant overfitting, in order to **demonstrate overfitting** in model-complexity plot, I injected some noise in the car evaluation dataset. The resulting model-complexity plot after **noise injection** is shown in Fig.5.

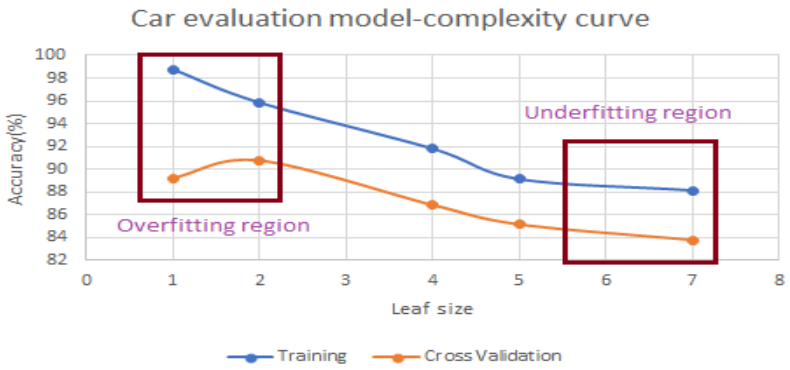

Fig.5.Car Evaluation Model Complexity Plot after **noise injection** to **demonstrate overfitting**

When the leaf-size is 1, the noisy training data adds extra nodes to the decision tree to accommodate the misclassified training instances, thus leading to a larger complex tree. Even though this complex tree performs well in the in-sample testing, it gives lesser accuracy for the cross-validation set because of the erraneous leaf nodes created to accomodate the noise. This results in **overfitting/high variance** problem. When the tree is pruned, however the effect of this noise is subsided as the misclassified training instances cannot have their own specific leaf nodes now. As the **pruning becomes higher**, the decision tree model is too simple to classify validation set accurately, hence causing **underfitting**. Regarding **execution time**, the time taken to build the model decreases with increasing pruning as the built model is smaller and simpler. The query time is mostly dependent on the validation set size.

2. Artificial Neural Networks (ANN)

Multilayer perceptron classifier in Weka was used to implement ANN on both the datasets. This algorithm has an input layer, at least one hidden layer and an output layer. For **linearly separable data**, hidden layer is not needed, and a standard perceptron algorithm will suffice. Each attribute forms a node in the input layer. Each possible class forms a node in the output layer. Every node in one layer is connected to every node in the next layer and the weights of each of these connections are set using **gradient descent** and **backpropagation**. The **weight changes** are computed by multiplying the gradient with the **learning rate** and summing it up with the product of the previous weight change and **momentum**. Smaller learning rates help in taking smaller steps so as to avoid overshooting the error function minima, but increase the time needed to converge. Momentum attenuates the oscillations in the gradient descent and prevents converging to local minima. Higher momentum values create instability by overshooting the minimum and very small values need not reliably avoid local minima. So, apt selection of learning rate and momentum is required for the algorithm to perform its best. A node gets activated when the weighted sum of its connections exceeds its **activation threshold** (bias). The number, size and activation function of hidden layers, and the values of learning rate and momentum are the hyperparameters that can be tuned.The complex structure of the neural network makes it prone to **overfitting** the training data. To prevent this, **early stopping technique** was used where a part of training data is set as validation set and when the performance of the model worsens on this validation set, the training is stopped irrespective of the number of epochs(iterations) done. This was done by setting *ValidationSetSize* = 10 in Weka so that 10% of training data is used for validation. Another method is **regularization**, where a penalty is imposed on the weights of the nodes to avoid the model **overfitting** the training data. This gives a simpler model with a small margin of error (slightly underfits training data), and is done in Weka using a meta cost-sensitive classifier. GridSearch and CVParameterSelection were used to select the best hyperparameter values. A range of values around the best were experimented for model-complexity plot and the table. The best values of hyperparameters are highlighted in Table.1 and 2.

Table.1. Variation of hyperparameters (number of nodes in the hidden layer, learning rate, momentum) for letter recognition dataset (Only one hidden layer was used)

Parameters (nodes, L, M)	Training accuracy	Cross-Validation accuracy
21, 0.1, 0.1	82.58	82.4372
21,0.5,0.59	81.45	79.5573

42, 0.1,0.1	87.837	85.8839
42,0.3,0.2	88.109	84.5467
42, 0.4, 0.3	84.0987	83.7657

Table.2. Variation of hyperparameters (number of nodes in the hidden layer, learning rate, momentum) for car evaluation dataset (Only one hidden layer was used)

Parameters (nodes, L, M)	Training accuracy	Cross-Validation accuracy
5, 0.1, 0.1	99.9172	98.2616
5,0.3,0.2	100	98.0132
5, 0.4, 0.3	100	98.5927
5,0.4,0.4	100	98.2616
4, 0.4,0.3	100	98.1788
10,0.4,0.3	100	98.2616

Initially, the default value of node count, that is, the average of attribute-count and class-count was tried for both datasets. However, for letter recognition dataset (table 1), when the node count was increased to 46 (sum of attribute-count and class-count), the performance got better at the cost of the network being large and complex. As noticeable from table 2 for car-evaluation dataset, it is interesting that when the number of nodes were increased or decreased from 5 (default value), the cross-validation accuracy went down, as indicated by green rows. So, the best fitting node count was 5. This is because node count values less than 5 gave underfitting networks that are too simple to accommodate the data and node counts greater than 5 gave overfitting networks. The learning curves in Fig.6 and Fig.7 were plotted using the best-fitting hyperparameters, which are highlighted using yellow color in Table 1 and 2.

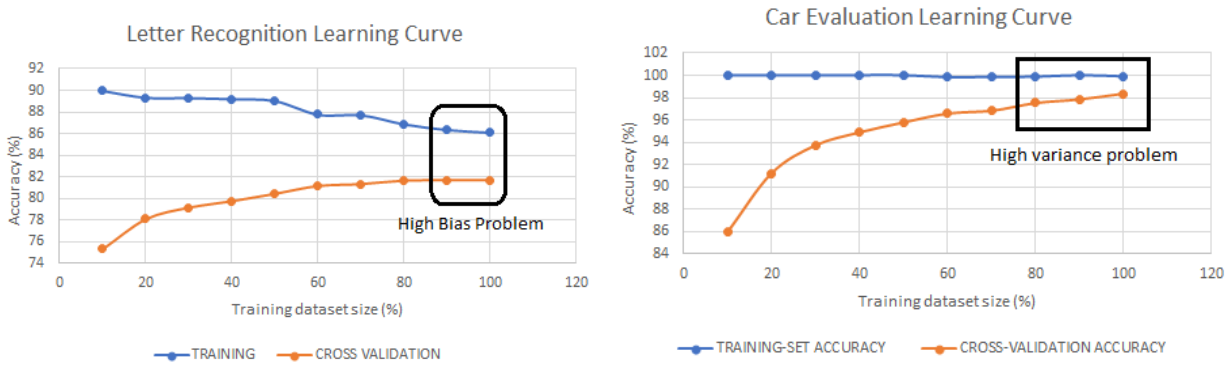

Fig.6.Letter Recognition Learning Curve [LEFT] and Car Evaluation Learning Curve [RIGHT]

The learning curve for **letter recognition** dataset exhibits **high bias problem**. That is, the model plateaus out/flattens instead of fitting the training data perfectly. This dataset was able to get around 100% accuracy with other algorithms, but here it has 86% accuracy only. Initially, the accuracy for training set is better because the model could accommodate the training data. But as the training data got bigger, the model (hypothesis) was not sufficiently complex enough. Thus, to overcome this problem, the network can be made bigger/more complex. Increasing iteration count did not improve the results. The learning curve for **car evaluation** dataset shows **high variance problem**, and the model performs well on training data but not on the validation set. Variance gives an estimate of the generalization power of the model. It is, effectively, the gap between the two accuracies in the learning curve. In Fig.6 [RIGHT], it is noticeable that the gap is high and as training dataset percentage increases, the gap decreases. Hence, feeding more training data to the model can help in overcoming high variance problem. If regularization and early stopping technique were not used, the variance would have been even higher. The **learning curves** were also plotted with respect to the **number of iterations** of ANN model. For letter recognition dataset, the cross-validation accuracy flattened at around 600 iterations. This can be because the weights in the network got their best estimated values within 600 iterations. For car evaluation dataset, it took only 20 iterations to do the same, mainly because the dataset is small with a smaller number of features. The **execution time** increases with increasing number of nodes and layers, as the network gets more complex and takes more time to be built. It also increases

when the learning rate and momentum decreases, as more and smaller steps will be taken towards convergence to the minimum.

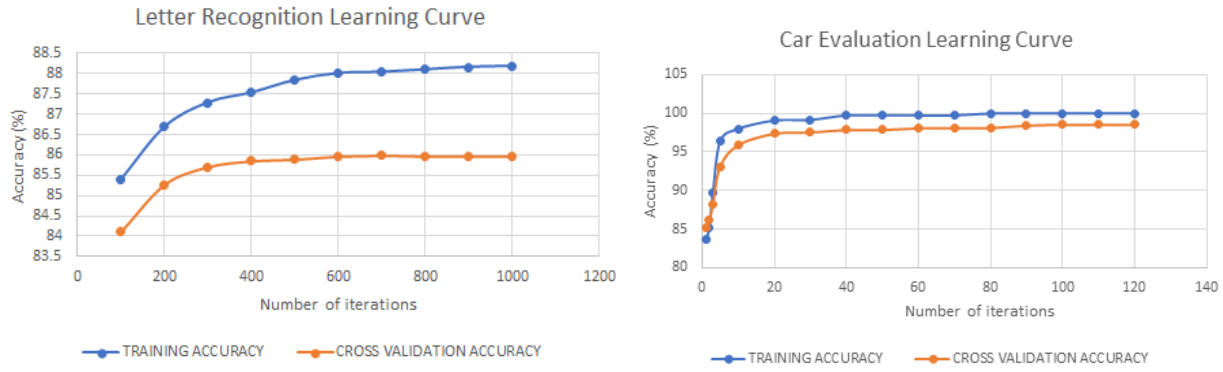

Fig.7.Letter Recognition [LEFT] and Car Evaluation [RIGHT] learning curves as a function of training iterations.

3. K-Nearest Neighbors

IBK classifier in Weka was used to implement KNN, which is an instance-based learning or **lazy learning**. In KNN classification, the output class of each instance is the most common among its k neighbors. If **distance weighing** is used, each neighbor is weighed inversely proportional to its distance from the data point. This algorithm can perform well even when data is **not linearly-separable**. GridSearch and CVParameterSelection were used to determine the best k and type of distance weighing for both the datasets. For **letter recognition dataset, k =4** and weighing the neighbors proportional to **(1-distance)** gave the best results, as in Fig.8[LEFT]. This is because the values of k lower than 4 cause **overfitting** to the training data. For example, if k=1, all validation data points just pick the output class of the nearest training data point, and this can be inaccurate if those two data points were on opposite sides of a separation boundary. Higher values of k give an **underfitting model**, as many neighbors are considered to classify each data point, and some of them can be very far-off thus not belonging to the same class as the concerned data point. For **car evaluation dataset**, **k=1** and **no distance weighing** gave best performance on cross-validation. KNN does **not** show **overfitting** with this dataset. The neighboring data points seem to belong to the same classes. However, when k is increases, it shows **underfitting** for similar reasons as of the other dataset.

During my experimentations, I noticed **an interesting trend** and I am showing it in Fig.9, where the neighbors are weighed inversely proportional to their distance from the data point for **car evaluation** dataset. It was interesting that the training data always shows 100% accuracy for any value of k. This is because during in-sample testing, the model has the same data point being queried and the distance between them is 0, thus the weight(1/distance) will be infinity. So other data points are not considered and hence in-sample testing always gives **perfect 100% training accuracy**. This is true for all datasets but for demonstration purposes, I used car evaluation dataset.

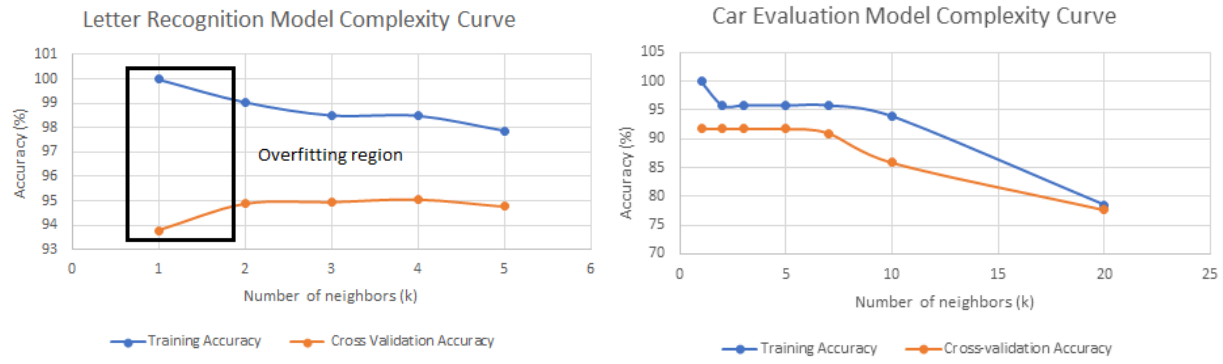

Fig.8.Model Complexity curves for Letter Recognition [LEFT] and Car Evaluation [RIGHT]

Supervised Learning Algorithms for Letter Recognition and Car Evaluation
Surabhi Amit Chembra
surabhichembra@gatech.edu

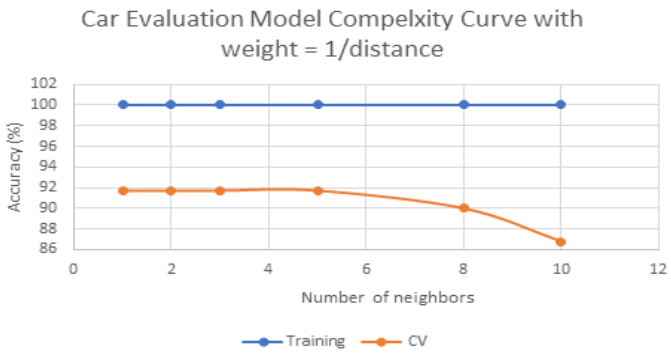

Fig.9.Model Complexity Curve for Car Evaluation with distance weighing

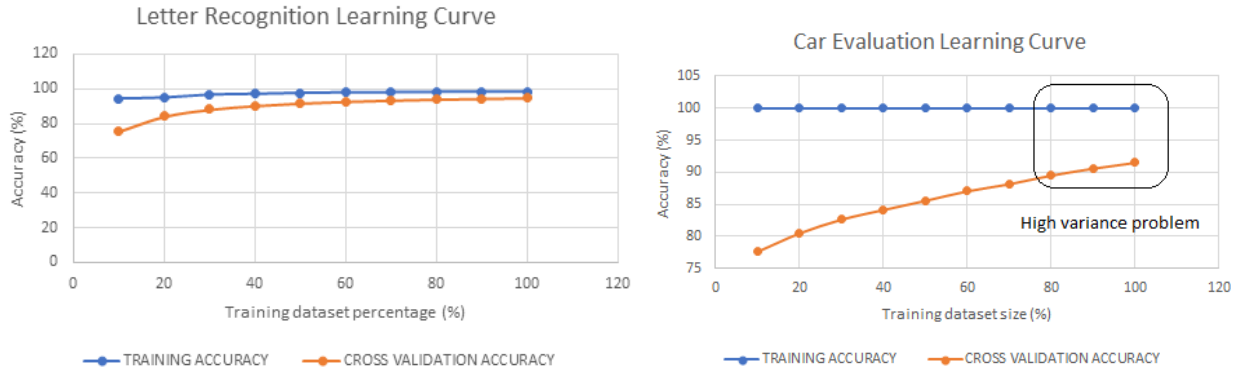

Fig.10. Learning Curves for Letter Recognition [LEFT] and Car Evaluation [RIGHT]

Both learning curves were plotted using the best-fitting hyperparameters listed earlier. The learning curve for letter recognition does **not** exhibit **high variance or high bias problem**. This can be because the training data set is large and balanced, thus correctly defining the separate decision boundaries. Also, setting k to 4 **prevented overfitting** to the training data. Meanwhile, the learning curve for car evaluation shows a **high variance problem**, probably because the training data has only 1208 records and more training data could fix this issue and increase the cross-validation accuracy. One **intuitive reasoning** for this can be that when the training data size is less, those data points were scattered, and the separation boundaries were slightly misplaced. However, when more training instances are fed into the algorithm, they correct these boundaries and so, the testing samples get classified more accurately.

4.Support Vector Machine

LibSVM in Weka is used to implement Support Vector Machines. SVM maps the training data into a space with separate categories divided by a gap(margin) with maximum possible width, and validation instances are mapped to this same space and their class is predicted based on which category they fell into. Other than this linear classification using **maximum-margin hyperplanes**, SVM can also deal with **non-linear data** using **kernel trick** which maps the linearly non-separable features into high dimensional feature spaces. The **linear kernel**, radial basis function (**RBF**) and **polynomial kernel** functions are tried for each of the datasets. The SVMType used is C-SVC (classification) and cost is set to 1.0 in Weka. Since SVM assumes that the input features are normalized and to avoid SVM getting **over focused** on dimensions with large range of values, the features of both the datasets were **normalized** prior to training.

Table.3. Variation of SVM kernels for Letter Recognition dataset

Kernels	Training accuracy	Cross-validation accuracy
Linear kernel	86.7417	84.1835
RBF with gamma = 0.0	99.4409	96.8137

Polynomial with degree= 4 and gamma=0.5	100	94.1176

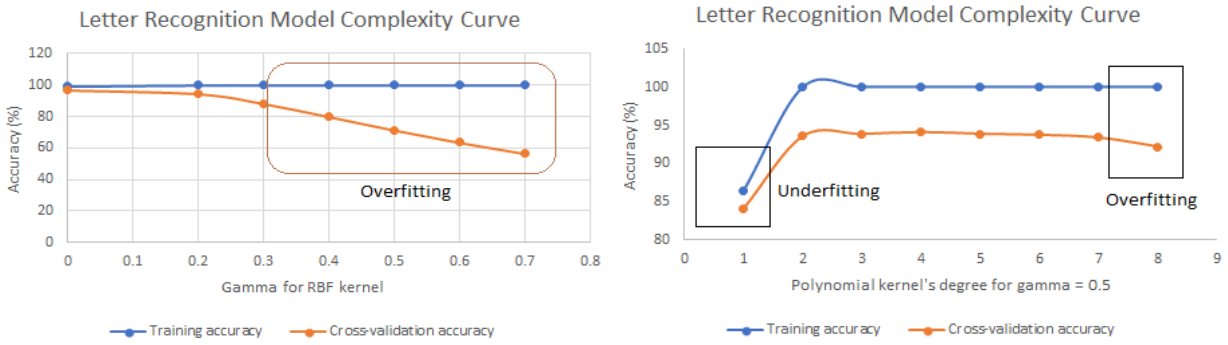

Fig.10. Model complexity Curves for Letter recognition using RBF kernel [LEFT] and polynomial kernel [RIGHT]

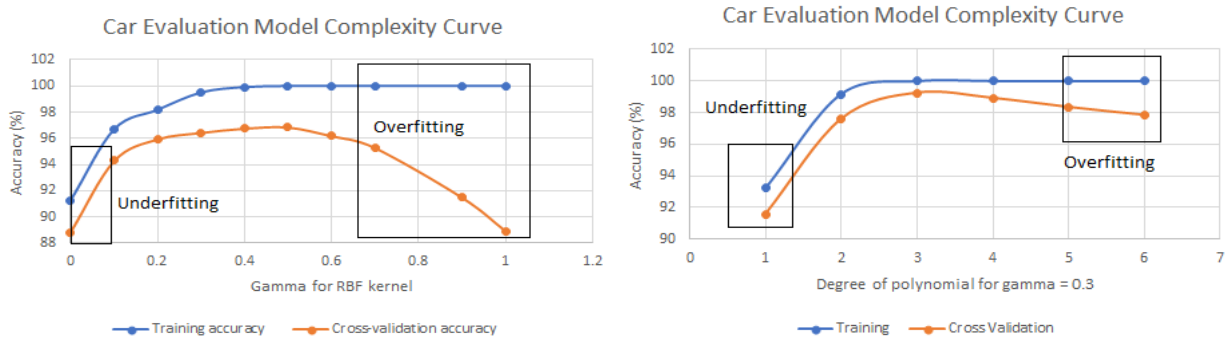

Fig.11. Model complexity Curves for Car Evaluation using RBF kernel [LEFT] and polynomial kernel [RIGHT]

Table.4. Variation of SVM kernels for Car Evaluation dataset

Kernels	Training accuracy	Cross-validation accuracy
Linear kernel	94.2881	93.2119
RBF with gamma = 0.5	100	96.8543
Polynomial with degree =3 and gamma = 0.3	100	99.255

For both datasets, the best-fitting hyperparameters were found using GridSearch and CVParameterSelection. For **letter recognition dataset**, **RBF kernel** with **gamma=0.0** gave the best result. As gamma is increased, RBF kernel shows overfitting with this dataset. For polynomial kernel, degree=4 and gamma=0.5 works the best. Though it gives 100% training accuracy, the cross- validation accuracy is higher with RBF kernel. For **car evaluation dataset**, polynomial kernel with **degree=3** and **gamma=0.3** worked the best. **Linear kernel** showed better performance for this dataset than for letter recognition dataset. The intuitive reasoning behind this can be that car evaluation dataset is more linearly separable than the letter recognition dataset. Regarding polynomial kernel function, both datasets show similar trends in Fig.10 and 11, where **lower degrees** lead to **underfitting** as the hypothesis is not complex enough to accommodate the dataset and **higher degrees** give complex models that **overfit** the training data. The best fitting hyperparameters from the model complexity plot (listed earlier) were used in plotting learning curves. The learning curves for both the datasets does not show any signs of bias problem, which says that the model suits the data. There is a slight high variance problem in both datasets, which can be fixed by using more training data.

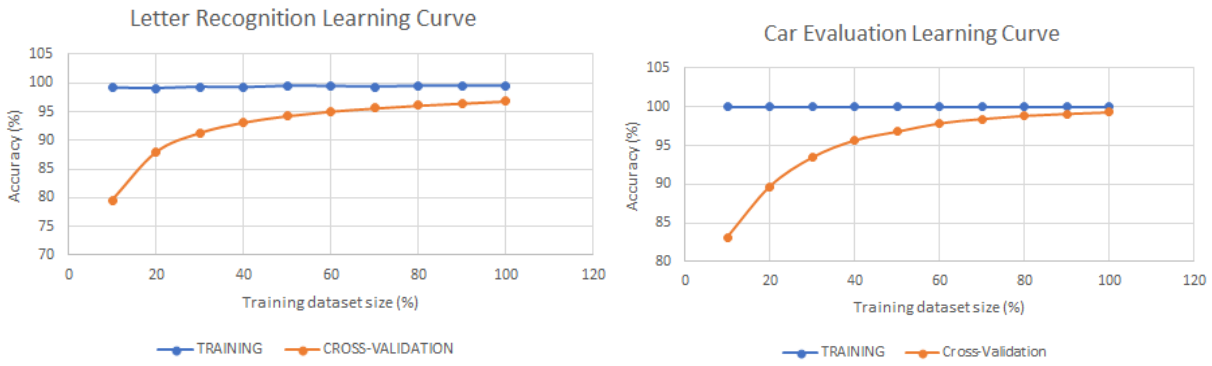

Fig.12.Learning curves for letter recognition [LEFT] and Car Evaluation [RIGHT]

The **execution time** increases with increasing gamma for RBF kernel and increasing degree for polynomial kernel, mainly because the model gets more complex.

5.Boosting

Adaboost M1 is used in Weka to implement Boosting. It is an **ensemble machine learning** algorithm used for classification problems, which uses a **group of weak learners** added in series. The training data is passed through a model and the wrongly-classified data points are given a higher weightage and passed to train the next model. The number of models to be created can be specified using the parameter *numIterations* in Weka. **J48 decision tree** is used as the weak underlying learner. The best-fitting hyperparameters for J-48 are found using CVParameterSelection and GridSearch. For the **letter recognition** dataset, best **confidence value** is **0.05** and best **leaf-size** is **2**. When J-48 was used as a single model, the best leaf size was 1 and confidence value was 0.45 for this dataset. For **car evaluation** dataset, best **confidence value** is **0.15** and best **leaf-size** is **3**. When J-48 was used as a single model, the best leaf size was 1 and confidence value was 0.55 for this dataset. This is interesting because it shows how boosting supports **aggressive pruning**. The main reason for using boosting is to **avoid overfitting** caused by a single learner. Though the **individual learners** can **overfit or underfit** certain data points, the output prediction for any data point **considers all the learners**, thus the combined average/mode should neither overfit nor underfit as the bias of individual learners gets cancelled by each other. However, **outliers can be problematic** for boosting because boosting build next learner based on data points misclassified by previous learners(residuals). Since outliers will be misclassified by early learners, later learners in **gradient boosting** will be overly attentive to these random noise/outliers.

Table.5. Variation of hyperparameters (confidence value and leaf size of underlying decision tree) for Letter recognition dataset.

Parameters (C, M)	M=1	M=2	M=3	M=4
C=0.025	93.9108	94.0411	93.796	94.1253
C=0.05	94.1713	94.1789	94.0257	94.1023
C=0.25	93.9032	94.0564	93.796	93.9338
C=0.35	93.8649	93.9951	93.8572	93.8572
C=0.45	93.9491	94.1406	93.773	93.6964
TRAINING = 100% for all, C=confidence, M = min_num_obj (leaf_size)				

Table.6. Variation of hyperparameters (confidence value and leaf size of underlying decision tree) for Car Evaluation dataset.

Parameters (C, M)	M=1	M=2	M=3	M=4
C=0.05	94.3709	94.9503	95.447	95.447
C=0.1	95.3642	94.7848	95.447	95.0331

C=0.15	94.3709	95.6126	95.6954	95.0331
C=0.25	94.3709	95.1987	95.1987	94.2053
c=0.45	93.7086	94.1125	95.4536	95.447
C=0.55	92.1358	92.1358	94.4536	95.1987
TRAINING = 100% for all, C=confidence, M = min_num_obj (leaf_size)				

The learning curves in Fig.13 and 14 were plotted using the best hyperparameters listed earlier. Regarding learning curves plotted as a function of training data size, both the datasets show similar trends when the training data size was increased. Both datasets exhibit a slight **high variance** problem when the data size is around 100%, which can be fixed by **feeding more training data** to the model. When the learning curve was plotted with respect to the number of underlying learners, as in Fig.14, the cross-validation accuracy in both datasets plateau or flatten out without improvement once the number of underlying learners reaches around 90. To verify that not selecting the best possible boosting model is not the reason, I tried other hyperparameters, but the result was same, and the cross-validation accuracy could not rise above 97%. A possible explanation can be that it is an inherent bias for boosting algorithm with the two datasets selected. The **execution time** increases when the number of iterations /numbers of underlying learners is increased. Also, as the confidence value of the underlying learner decreases or the leaf size decreases, the time taken to build the model increases.

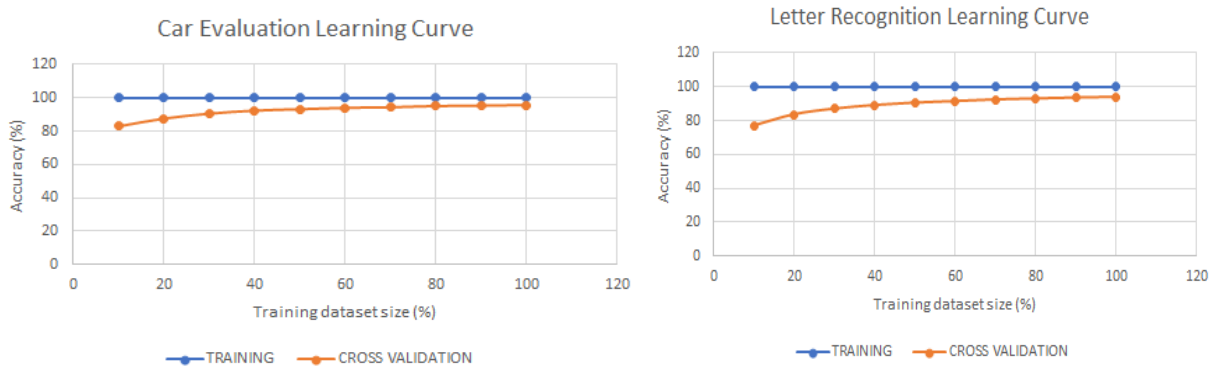

Fig.13.Learning curves for letter recognition [LEFT] and Car Evaluation [RIGHT]

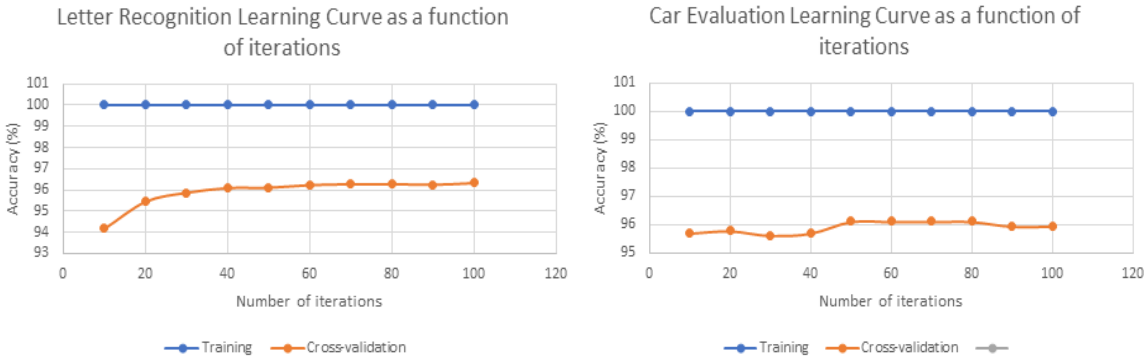

Fig.14.Learning curves for letter recognition [LEFT] and Car Evaluation [RIGHT], as a function of iterations

CONCLUSION

Testing Results: - Accuracy was used as the **metric** to evaluate **performance** because these are classification problems and accuracy shows how much of the instances were correctly classified. Moreover, for balanced datasets like letter recognition dataset, accuracy works very well. The training, cross-validation and testing accuracies of all the five supervised learning algorithms applied on both the chosen datasets are shown in Table.7 and 8. The best-fitting hyperparameters listed earlier in this report for each algorithm-dataset pair were used to do testing on the final test set.

Surabhi Amit Chembra
surabhichembra@gatech.edu

Table .7. Training, Cross-Validation and testing accuracy for letter recognition dataset using best-fitting hyperparameters

Algorithm	Training accuracy	Cross-Validation Accuracy	Testing accuracy
Decision Tree	99.4043	86.4383	86.9565
ANN	87.837	85.8839	85.6201
SVM	99.4409	96.8137	96.2224
KNN	98.4911	95.0674	94.6187
Boosting	100	95.6954	94.1019

Table .7. Training, Cross-Validation and testing accuracy for car evaluation dataset using best-fitting hyperparameters

Algorithm	Training accuracy	Cross-Validation Accuracy	Testing accuracy
Decision Tree	100	92.1358	98.945
ANN	100	98.5927	100
SVM	100	96.8543	98.342
KNN	100	91.7219	99.28
Boosting	100	94.1789	99.9876

Regarding execution time, for both the datasets, KNN was the fastest followed by decision tree, then Boosting, then SVM and finally ANN. This is mainly because **KNN,** is a **lazy learner** (instance based) whereas **ANN** needs to build a **complex network** with weights of each interconnection between nodes tuned. The linear kernel of **SVM** is quicker, however, RBF and polynomial kernels take time to map data points to another dimensional space. **Decision tree** building is quick if certain features give high information gain and thus help build a simple tree with few nodes. Boosting takes time because it has many decision trees added in series. So, if asked which algorithm is best in terms of time needed to build a model, it is KNN. However, if asked in terms of query time, Decision Tree gives fastest results based on my experimentations.

Which algorithm works best for which type of datasets and why?

For the letter recognition dataset, **SVM** shows best performance using RBF kernel, KNN and Boosting show good performance, while Decision Tree and ANN show relatively poor performance. For the car evaluation dataset, **ANN** give best performance, followed by Boosting and KNN. Decision Tree and SVM also gives comparative performance, and thus, all algorithms do well on this dataset. Since the car dataset performed well with algorithms, there were no much information in confusion matrix (almost all instances were correctly classified). However, **letter recognition dataset** performed not-so-good with ANN. And to analyze instances belonging to which class were mostly classified wrong, I decided to use confusion matrix as in Fig.15. It is noticeable that "H" was wrongly classified many times into K and R. So, adding more training instances belonging to H might improve the model well. Also, R gets **misclassified** into K several times. Hence looking at the **confusion matrix**, we can selectively decide which training instances should be additionally fed to the model to improve it. Analysis of Confusion matrices of other algorithm-dataset pairs were not included due to space constraints.

SVM is best suited when the dataset is relatively small. Quoting the **Occam's Razor**, in any algorithm, it is always best to start from the simple hypothesis and get more complex if the hypothesis cannot fit the complexity of the data. Simpler models are easier to build, validate, interpret and improve. For example, when SVM was used on both datasets, I started with linear kernel and since it did not perform well as the data is not linearly-separable, I tried the kernel trick. Performance of SVM is also affected when the number of features is much larger than the number of instances. This **curse of dimensionality** caused by huge number of dimensions of the dataset also affects other algorithms like KNN, ANN and Decision trees which tend to model the non-linear decision boundaries perfectly. SVM performed well in letter recognition, however, because number of samples (20K) was far higher than attribute count (16). ANN, however, needs more training data to build its **complex black-box model** perfectly and hence did

not give the best performance. Generally, increasing the dimensionality gives better performance on the training dataset, but it cannot generalize well over to the testing data, thus leading to overfitting. Thus, the number of training instances used should grow **exponentially** with the number of dimensions used. A typical rule of thumb in machine learning can be that there should be at least 5 training samples per dimension. **Peaking phenomenon** or Hughes phenomenon says that when the number of training samples is fixed, the generalization power of the model first increases with dimensionality but then decreases.

```
=== Confusion Matrix for Letter Recognition dataset with ANN ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z   <-- classified as
 196   1   0   1   0   0   0   0   0   3   3   1   2   7   0   0   3   0   3   1   0   0   0   1   5   0 |   a = A
   0 203   0   4   1   0   0   0   0   0   4   0   0   0   0   0   0   0   1   0   0   0   0   4   3   0 |   b = B
   0   0 198   0   3   0   4   0   0   1   1   2   0   1   0   0   1   0   0   0   3   0   0   0   0   0 |   c = C
   1   3   0 211   0   1   0   0   0   0   0   0   3   2   1   0   0   2   0   0   3   0   0   1   0   0 |   d = D
   0   2   4   0 203   2   2   0   0   0   0   0   0   0   0   0   0   1   0   2   0   0   0   0   1   1 |   e = E
   0   7   8   0   7 189   0   0   0   0   0   0   0   0   0   9   1   0   1   4   0   0   0   0   0   0 |   f = F
   2   3   4   1   8   0 162   0   0   4   5   2   2   0   2   1   8   0   3   0   0   0   6   7   3   0 |   g = G
   1   6   6  20   3  12   6  45   0   1  27   2   3  12  19   4   4  21   1   0   7   1   0  11   0   0 |   h = H
   0   1   0   3   1   5   0   0 120   9   0   0   0   0   0   5   0   0   3   0   0   0   0   3   1   7 |   i = I
   2   0   0   3   0   3   0   0   2 185   2   1   1   0   1   0   0   0   3   0   2   0   0   3   2   5 |   j = J
   0   0   0   2   0   0   0   0   0   0 203   0   0   1   0   0   0   4   0   0   0   0   1   5   0   0 |   k = K
   0   0   6   0   6   0   2   0   0   0   8 172   0   0   0   0   0   1   0   0   0   0   0   5   0   2 |   l = L
   0   3   0   1   2   0   0   0   0   0   0   0 209   4   0   0   0   0   0   0   0   0   1   0   0   0 |   m = M
   0   0   0   3   0   2   0   0   0   0   1   0   1 187   5   2   0   2   0   0   1   0   2   1   0   0 |   n = N
   0   2   4   5   0   0   2   0   0   0   0   0   0   0 178   0   9   4   0   0   0   0  11   0   0   1 |   o = O
   0   5   3   1   2   1   0   0   0   0   1   0   0   0   0 217   1   1   0   0   1   0   0   0   2   0 |   p = P
   0  14   0   0   0   0   2   0   3   0   0   5   0   1   3   1 191   0   1   0   0   0   0   1   0   6 |   q = Q
   1   7   0   6   2   0   0   0   0   0  19   0   0   6   0   0   0 180   0   0   0   0   0   0   1   0 |   r = R
   1   4   0   2   8   3   0   0   0   0   0   0   0   0   0   1   0   0 191   1   0   0   0   2   2   4 |   s = S
   0   4   0   1   6   1   6   0   0   0   5   0   0   0   0   1   0   0   0 187   1   0   0   4   3   6 |   t = T
   0   0   2   0   0   0   0   0   0   0   2   0   4   0   1   0   0   0   0   0 217   0   5   0   0   0 |   u = U
   0   7   0   1   0   0   0   0   0   0   1   0   0   4   0   2   1   1   0   0   0 183  10   0   2   0 |   v = V
   0   0   0   3   0   0   0   0   0   0   5   0   6   4   0   0   0   0   0   0   4   0 196   0   0   0 |   w = W
   0   0   1   3   4   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0 193   0   2 |   x = X
   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   3   0   0   5   0   1   1   1 212   0 |   y = Y
   1   0   0   0   1   1   0   0   0   7   0   0   0   0   0   0   0   0   2   0   0   0   0   2   1 177 |   z = Z
```

Fig.15.Confusion matrix for Letter Recognition dataset with ANN algorithm.

In **KNN**, the underlying distance metric like Euclidean distance becomes meaningless in high dimensions, thus making it highly susceptible to the curse of dimensionality. So, **feature extraction** or feature selection or avoiding irrelevant features is necessary when trying to model data with high dimensionality and comparatively low training instances. Also, letter recognition dataset showed **overfitting with KNN** when k was reduced to below 4. This means that using Euclidean distance metric to choose closest point need not be accurate. **Decision trees** suit datasets whose classification relies vastly on the values of certain features. Selecting those features with **high information gain** at the root nodes or early nodes in the tree, helps in building a simple but predictive model. For example, for the **car dataset**, the attribute "**safety**" was the root node because it classifies the car into unacceptable if safety is low, irrespective of other features. This explains the high performance of decision tree in car dataset. On the contrary, **letter recognition** dataset does not have such strong attributes with commendable information gain, and thus the tree is very complex and less accurate. **Boosting** of decision trees gave better testing accuracy than a single decision tree for both datasets, mainly because it considers the output of several decision trees.

ANN performed well with car evaluation dataset. The smaller number of attributes and classes gave a simpler neural network. The comparatively poor performance of ANN on letter dataset can be attributed to more attributes and classes, but lesser training data and the fact that the other algorithms had advantages on that dataset, as described earlier. **Model interpretability** is another important concern for machine learning models, which is required to validate, troubleshoot and improve the model. If the built-model cannot be a black-box, then algorithms like **ANN** are less suitable. Unlike the nodes in decision trees or the distance metric in KNN, the complex weighted interconnections between several nodes of different layers is hard to interpret.

REFERENCES:

1. Weka: Data Mining Software in Java. (n.d.). Retrieved September 10, 2018, from https://www.cs.waikato.ac.nz/ml/weka/
2. Letter Recognition Dataset. UCI Machine Learning repository. Retrieved September 10, 2018, from https://archive.ics.uci.edu/ml/datasets/letter+recognition
3. Car Evaluation Dataset. UCI Machine learning repository. Retrieved September 10, 2018 from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation