CS 7641 Machine Learning
Robert Bobkoskie

Assignment 1:
Supervised Learning

I. INTRODUCTION

The goal of this assignment is to explore classifying data from supervised Machine Learning (ML) models (learners): Decision Tree (d_tree), Decision Tree with boosting and bagging, Support Vector Machines (SVM), k-nearest neighbor (KNN) and neural networks (nrl_net). The models were implemented in python using scikit-learn[1] (sklearn) and pandas[2]. Many of the sklearn functions for implementing the models was straight forward and well documented. Citation are referenced within the code, as well as at the end of this paper. The real work of this assignment was in identifying 'good' data, preprocessing the data to a grammar understandable to sklearn, and tuning modeling parameters to achieve 'interesting' classification problems.

II. Data

As mentioned, much work was required for data identification and preprocessing. With a goal to discover 'interesting' classification problems given my knowledge of ML and a tight project deadline, I selected two data sources[3] for classification that would achieve this goal. A brief note on the definitions of 'good' and 'interesting'. Initially, both terms were very nebulous, however, as I got deeper into the project, the definitions became more clear. Briefly, the data should be non-trivial, such that the classification problems produce interesting results across the range of SL models under consideration for this assignment.

The first data set was the built-in IRIS data set that came with sklearn. As I had no prior experience with ML, I selected the IRIS data as a window into exploring the various SL models required for this assignment. This data set would baseline my models, as well as increase my confidence and understanding as I pursued more challenging data sets. The IRIS data set is well known, thus, I will not deeply discuss this data set. Briefly, the data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. There are four attributes (features): sepal length (cm), sepal width (cm), petal length (cm) and petal width (cm). The objective (problem) is to classify the species of iris plant based on the four attributes.

The second data set (Data_2), **`Bank Marketing Data Set`**, was obtained from the **`UCI Machine Learning Repository`**. The data originates from a direct marketing campaign of a Portuguese banking institution. The marketing campaign was initiated to sell a bank term deposit (product) to clients. The data consists of 4119 instances with 20 attributes of the client: [education, marital status, existing loans, day of week, etc.]. The objective is to classify (label), and predict the outcome, 'client subscribed': ('yes') or ('no'). Data_2 proved to be more raw, with many strings and Null values that required modification before inputting to the ML models. I used pandas to: map strings to ints, eliminate data that had 'undetermined' values and grab the target column. Numpy slicing was used to grab the data set, which were all the columns excepts for the target column. After preprocessing, the data set had the following characteristics: two

classes (‘1 = Yes’, ‘0 = No’) with 3811 instances, and 18 attributes. One attribute, ‘duration’ was eliminated based on recommendation on attribute information provided by UCI, ‘Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no')’. The classification problem is to use the attribute values to predict if the client will subscribe (yes/no) to a term deposit.

III. Tune the Hyper-Parameters

All the models were evaluated with, and without cross validation (x_val). Why run without cross validation? The objective was to obtain a ‘baseline’ of results with default (or close to default) parameter settings. These results are used in comparison against the models running with tuned hyper-parameters (h-params).

A 10-fold cross validation was performed using: **`sklearn.model_selection.GridSearchCV`** (GS) for all supervised learning classifiers: Decision Tree, KNN, SVM. Sklearn GS allowed for a systematic evaluation, essentially ‘tuning’ and returning the optimal h-params for each classifier. Note that applying sklearn GS still requires some subjectivity in defining the ranges of h-params for evaluation. For example, the h-params used for decision tree classification are shown below:

Example 1: Hyper-parameter values for evaluation using cross validation.

```
hyper_params = {"criterion": ["gini", "entropy"],
                            "min_samples_split": [2, 10, 20, 100, 1000],
                            "max_depth": [None, 2, 5, 10, 100],
                            "min_samples_leaf": [1, 5, 10],
                            "max_leaf_nodes": [None, 5, 10, 20]}
```

I could have either included more h-params for evaluation, or added more values for each h-param under consideration. The choice was a balance between an exhaustive evaluation, run time and improved classification vs. the data. For example, the IRIS data set is quite simple, and, as will be shown, does not require an exhaustive evaluation to achieve good classification. On the other hand, Data_2 is quite large, and, although an exhaustive evaluation may improve results, the run time of the evaluation is not trivial. For example, I attempted to evaluate a polynomial (poly) fit of degree three with SVM. The evaluation ran, without returning for several hours. The computational requirement for a poly fit of degree three were too great, so I was capped at a poly of degree 2.

IV. DISCUSS THE RESULTS

The results will be discussed and presented in both tabular and graphical contexts for each model. All models were evaluated against two data sets with and without cross validation. Each of the two data sets were split evenly, dedicating 50% of the data for training and test sets respectively. Running (mostly) default h-params and against both classification problems (data

sets) without cross validation was designed to evaluate the influence that tuning h-params can have on the learners' performance (accuracy).

Decision Tree (with ensemble methods: Boosted, Bagging)

I implemented a decision tree model based on the ID3 algorithm. To minimize overfitting, I implemented pruning (constrained the growth of the tree) by adjusting the hyper-parameters. Post pruning, as per the C4.5 method was not implemented. For comparison, I generalized both problems with a decision tree classifier, as well as ensembles of: decision tree plus bagging, and decision tree plus boosting.

Refer to example 1 for the ranges of hyper-parameter values under consideration with cross validation. The h-param settings without x_val for d_tree were mostly default, except for the following: `{'max_leaf_nodes': None, 'min_samples_leaf': 1, 'random_state': None, 'criterion': 'entropy', 'min_samples_split': 50, 'max_depth': None}`.

Learning curves show performance on both training and test data as a function of training size. Convergence in the graph can be analyzed to ascertain the need for additional training data. Observing the learning curves for both problems, the cross validation score converges to a higher training score as the number of training examples increases. This implies that the number of training examples is sufficient for good generalization of the problem. The convergence of the IRIS data set shows that both training and x_val almost parallel each other. This is most likely a function of the small size of the data set (75 instances), vs. the large size of the h-param `'min_samples_split': 50`. The full decision tree for the IRIS data set in figure 1b shows a leaf node originating from the root node for class [0] based on the feature 'petal width'. Due to its size, the full decision tree was not able to be plotted for Data_2 in figure 2c. Although only partial, the d_tree for Data_2 exhibits signs of overfitting, e.g., the number of nodes with small split sizes.

Figure 1a: Learning Curve
IRIS (d_tree, x_val = False)

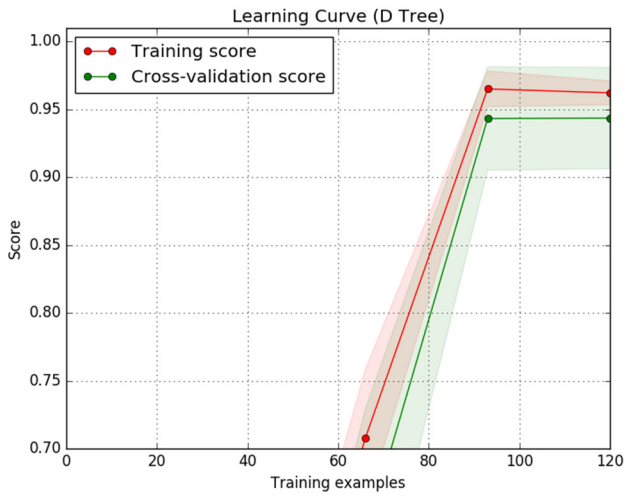

Figure 1b: Decision Tree
IRIS (d_tree, x_val = False)

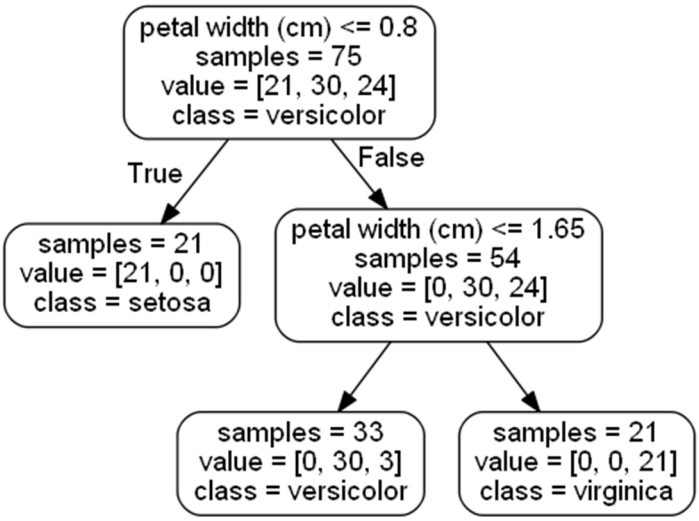

Figure 2a: Learning Curve
Data_2 (d_tree, x_val = False)

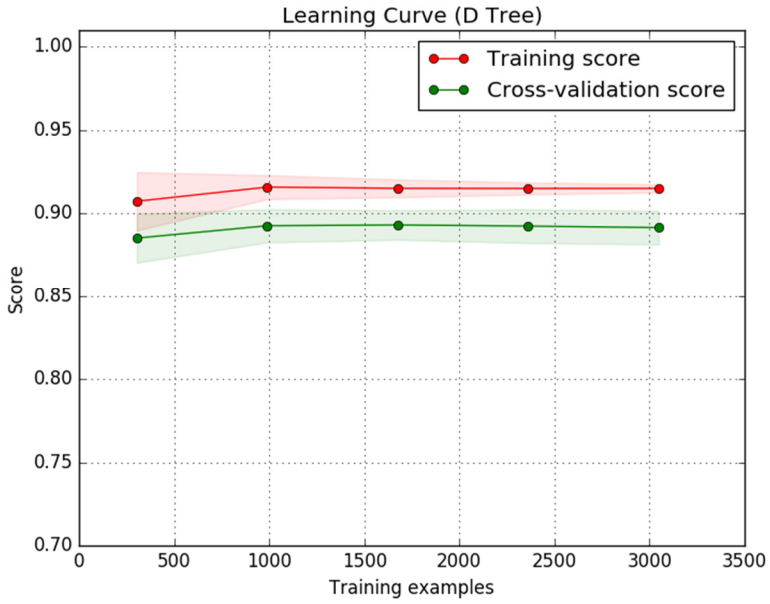

Figure 2b: Boosting Decision Tree
Data_2 (d_tree, x_val = False)

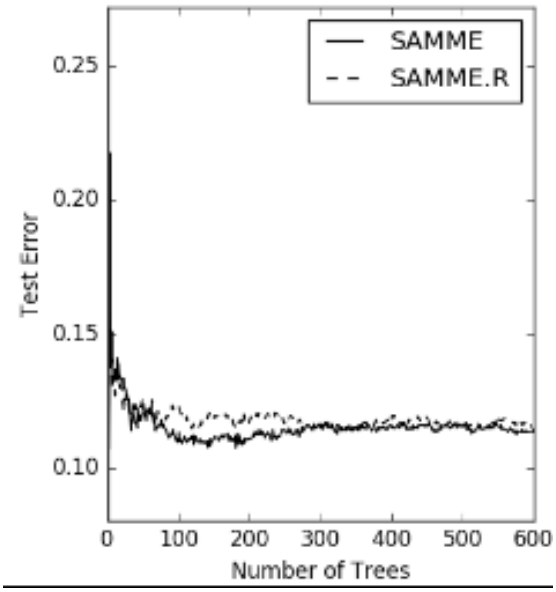

Figure 2c: Decision Tree (partial)
Data_2 (d_tree, x_val = False)

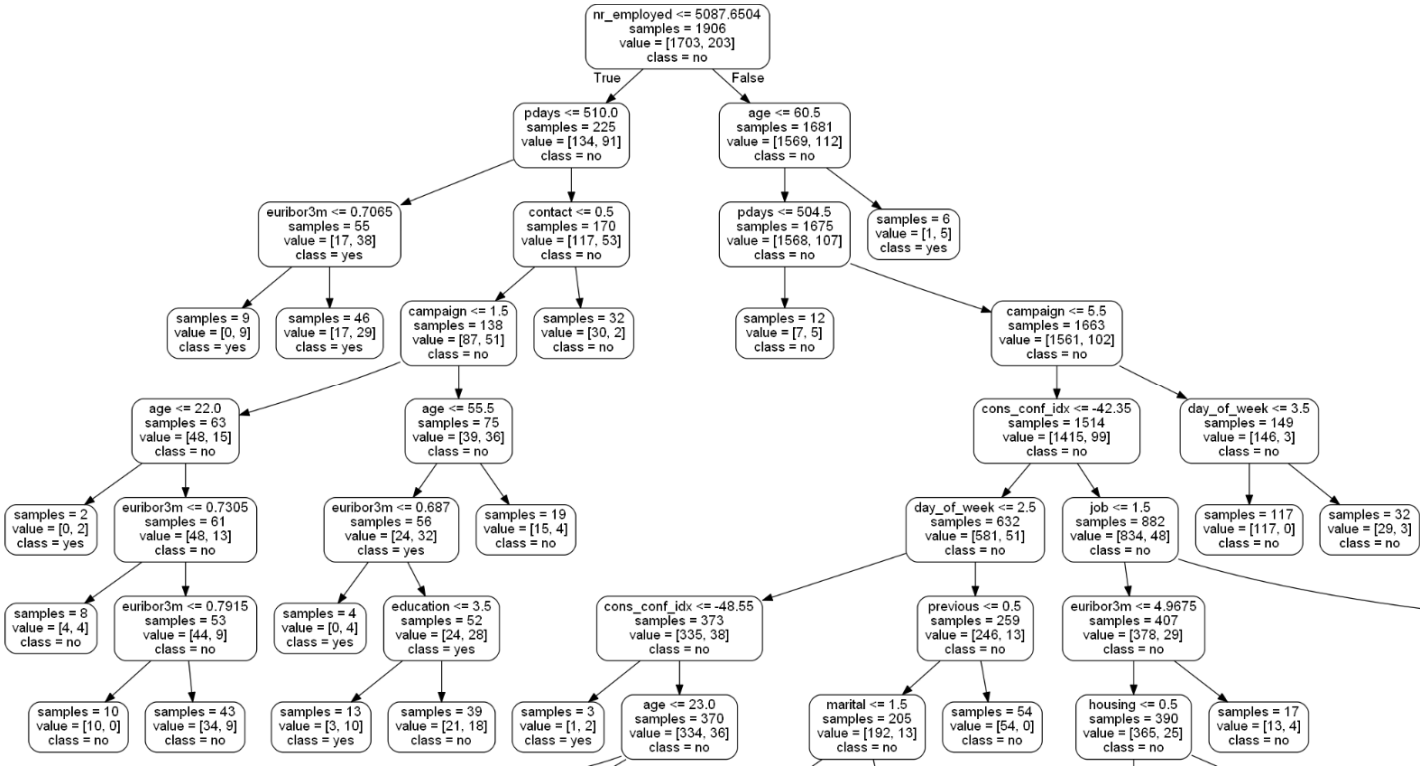

Figure 3a (bot): Learning Curve
IRIS (d_tree, x_val = True)

Figure 3b (rgt): Decision Tree
IRIS (d_tree, x_val = True)

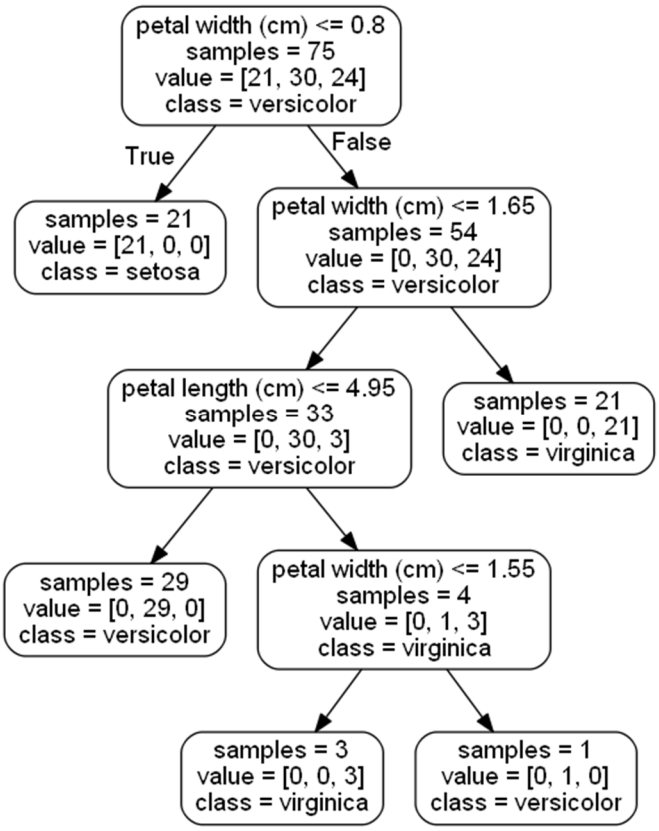

Figure 4a: Learning Curve
Data_2 (d_tree, x_val = True)

Graphical analysis for the decision tree learner with cross validation (figs 3*, 4*) shows many of the same characteristics as d_trees without cross validation (figs 1*, 2*). That is, both data sets evaluated with cross show convergence to the training set, thus, the number of training samples is sufficient.

In fact, additional evaluation with smaller training sets would be a worth wile experiment.

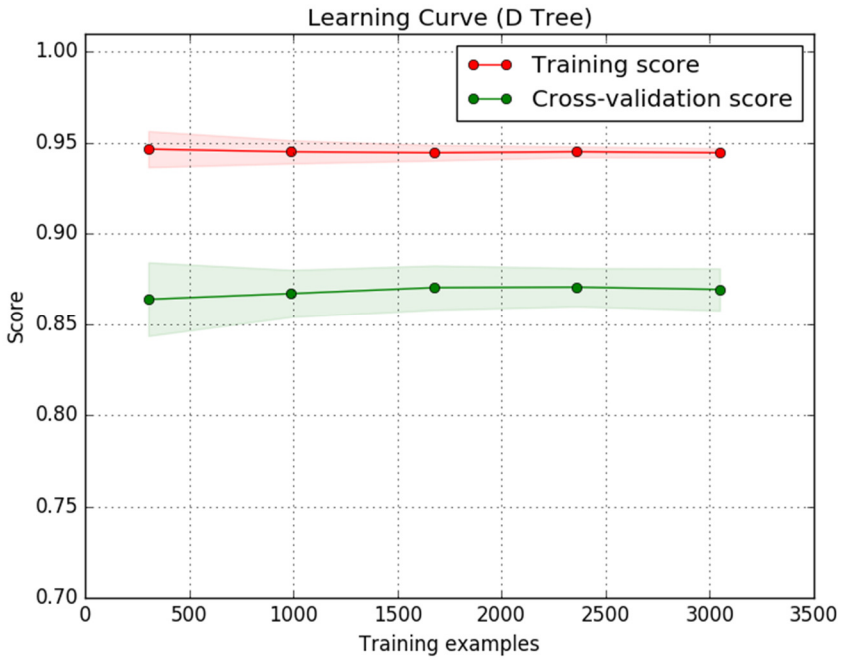

Figure 4b: Decision Tree (partial)
Data_2 (d_tree, x_val = True)

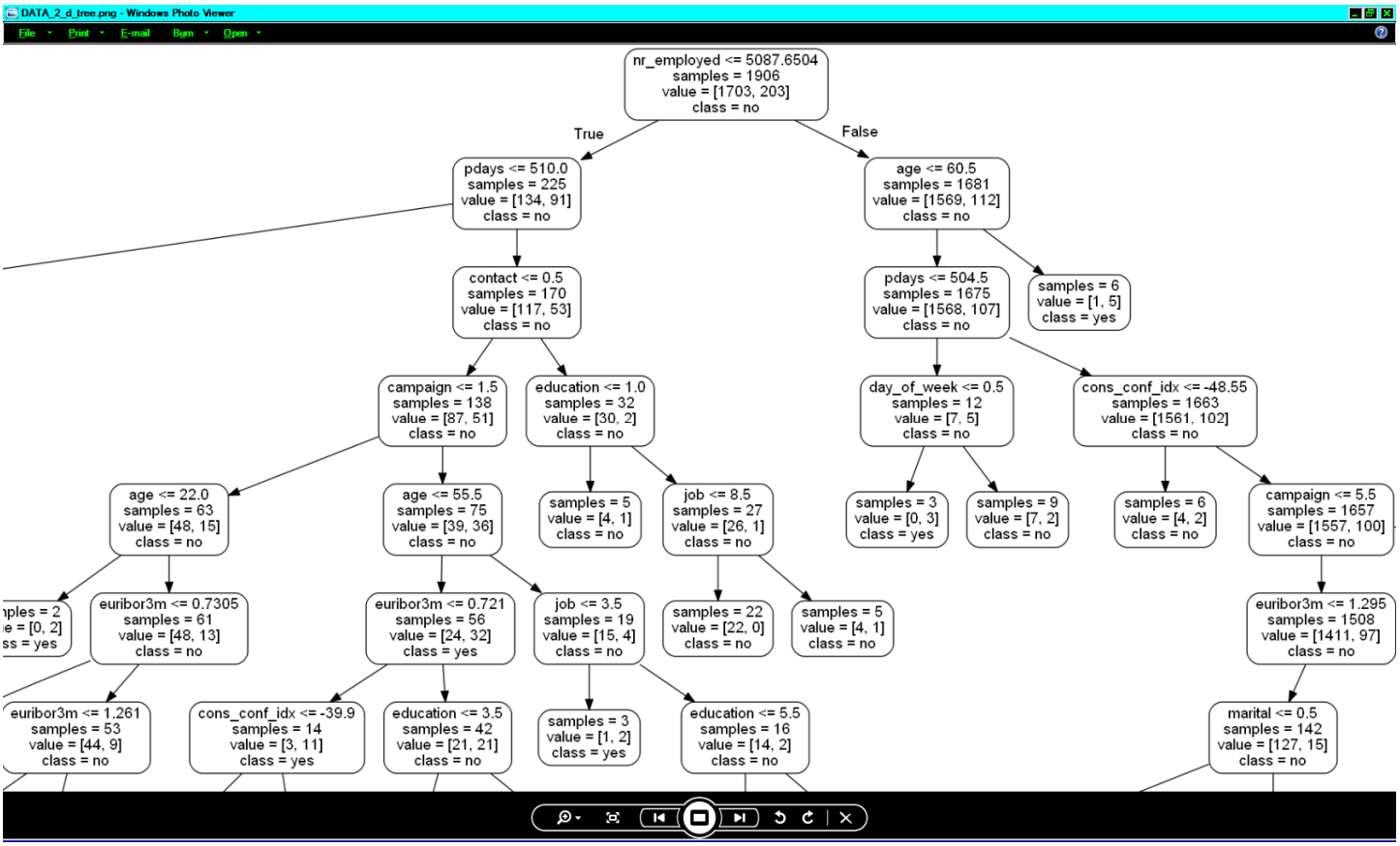

Table 1: Summery of Accuracy for Decision Tree Classifier with bagging and boosting. The results include: a measure of accuracy on the training set, the number of misclassified and correctly classified examples on the test set, and a measure of accuracy for the learners' generalization on the test set. The corresponding h-params, including tuned (x_val = True) are noted in the last column. Note that only the h-params modified for classification are shown. The remaining parameters were default values.

Table 1 **Decision Tree**	**Dec Tree**	**Dec Tree (Rnd Frst)**	**Dec Tree (Bagged)**	**Dec Tree (Add Bst)**	**Dec Tree (Grd Bst)**	**Hyper-Parameters**
x_val (False)						
IRIS: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *50* *30* *45* *60*	 *39* *54* *21* *28*	 *36* *54* *21* *28*	 *58* ***9*** ***66*** ***88***	 *85* *1369* *8631* *86*	`{'max_leaf_nodes': None, 'min_samples_leaf': 1, 'random_state': None, 'criterion': 'entropy', 'min_samples_split': 50, 'max_depth': None}`
DATA_2: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *90* *198* *1708* *90*	 *89* *191* *1715* *90*	 ***90*** ***184*** ***1722*** ***90***	 *88* *217* *1689* *89*	 *85* *1369* *8631* *86*	`{'max_leaf_nodes': None, 'min_samples_leaf': 1, 'random_state': None, 'criterion': 'entropy', 'min_samples_split': 50, 'max_depth': None}`

x_val (True)						
IRIS: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 ***96*** ***3*** ***72*** ***96***	 *93* *5* *70* *93*	 *95* *8* *67* *89*	 ***96*** ***3*** ***72*** ***96***	 *88* *1041* *8959* *90*	`{'min_samples_split': 2, 'max_leaf_nodes': None, 'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1}`
DATA_2: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *87* *254* *1652* *87*	 ***89*** ***190*** ***1716*** ***90***	 *89* *205* *1701* *89*	 *89* *209* *1697* *89*	 *81* *1803* *8197* *82*	`{'min_samples_split': 10, 'max_leaf_nodes': None, 'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1}`

Using cross validation was more effective at tuning the hyper-parameters for the IRIS data set. Although this is a function of setting the `'min_samples_split': 50` to a value much too large for the size of the data set, it shows the importance of cross validation to tune the h-params. The most accurate results (96%) were found for decision tree, as well as the decision tree plus addaboost.

The generalization of the learners on Data_2 was interesting in that the best results did not come from using cross validation to identify the hyper-parameters. The most accurate result was actually found with hard coded values for decision tree plus bagging (90%, 1722 correctly classified). Although it was expected that x_val would produce the most accurate model, this could be explained by the small number of hyper-parameters selected to tune, and perhaps the ordering of the h-params for evaluation by **`sklearn.model_selection.GridSearchCV`**. This seems plausible, as a couple of the h-params selected by x_val [`'min_samples_split': 10, 'max_depth': None`] seem odd for such a large data set (3811/2 instances x 18 features). One would expect that x_val would more aggressively prune (constrain the growth) the tree by selecting a larger value for `'min_samples_split'`, or setting a value for the `'max_depth'` h-param. Perhaps implementing post pruning, as per the C4.5 method would have improved performance as well. Also noteworthy, I did not choose to cross validate, or adjust the parameters for bagging or boosting. Adjustments to either approach (pruning, h-param selection) has the potential to improve the relative underperformance of x_val on the d_tree learners.

Additional Learners (SVM, KNN, Neural Network)

I implemented the classifiers: SVM, KNN, Neural Networks using sklearn. As with d_trees, to minimize overfitting, I constrained the learners by adjusting the hyper-parameters using **`sklearn.model_selection.GridSearchCV`** and cross validation.

Table 2: Default (x_val = false), and ranges of hyper-parameter values under consideration with cross validation.

Table 2 **h-params**	**Default Hyper-Parameters** **X_val = False**	**Hyper-Parameters for consideration** **X_val = True**

KNN	`{'n_neighbors': 5,`
	`'weights': 'uniform',`
	`'leaf_size': 30}`
		`{"n_neighbors": [int(0.10*np.shape(X_train)[0]),`
		`int(0.25*np.shape(X_train)[0]),`
		`int(0.50*np.shape(X_train)[0]),`
		`int(0.75*np.shape(X_train)[0])],`
		`"weights": ["uniform", "distance"],`
		`"leaf_size": int(0.10*np.shape(X_train)[0]),`
		`int(0.50*np.shape(X_train)[0]),`

		int(0.75*np.shape(X_train)[0])]}
SVM *	{'kernel': 'rbf', 'degree': 2}	{"kernel": ["linear", "sigmoid", "rbf", "poly"], "degree": [2]}
Neural Network	{'activation': 'relu', 'learning_rate': 'constant', 'solver': 'adam'}	{"activation": ["identity", "logistic", "tanh", "relu"], "solver": ["lbfgs", "sgd", "adam"], "learning_rate": ["constant", "invscaling", "adaptive"]}

* Note that `'degree'` only applies when 'kernel = poly'.

As with the decision tree classifiers, the learning curves for both problems show convergence to a higher training score as the number of training examples increases. This implies that the number of training examples is sufficient for good generalization of the problem.

Figure 5a: Learning Curve
IRIS (knn, x_val = False)

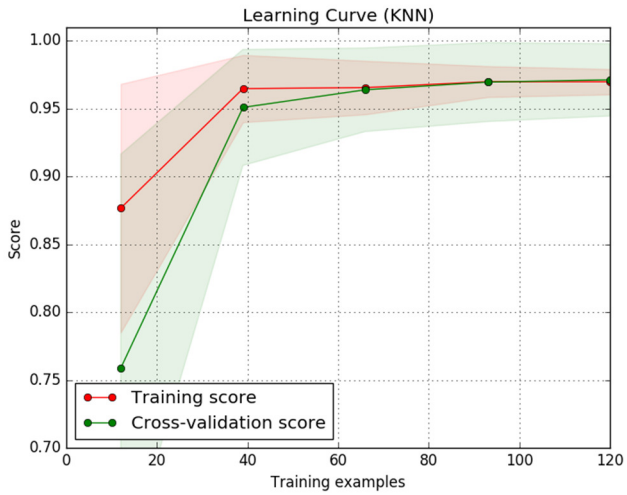

Figure 5b: Learning Curve
IRIS (knn, x_val = True)

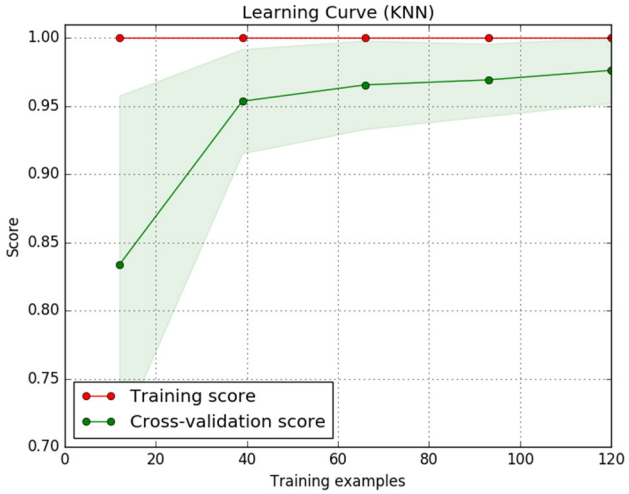

Figure 6a: Learning Curve
Data_2 (knn, x_val = False)

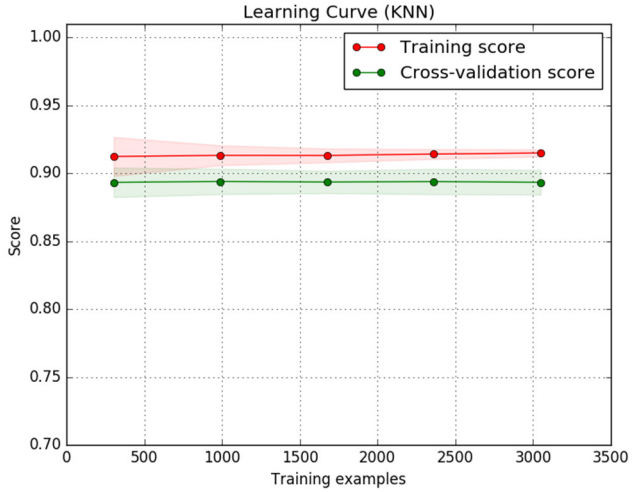

Figure 6b: Learning Curve
Data_2 (knn, x_val = True)

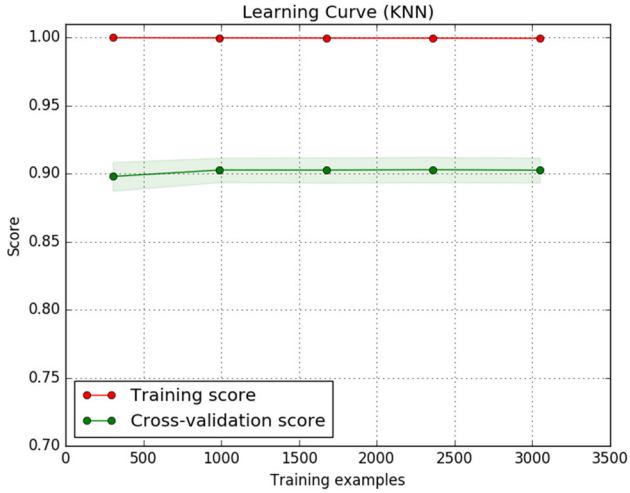

Figure 7a: Learning Curve
IRIS (svm, x_val = False)

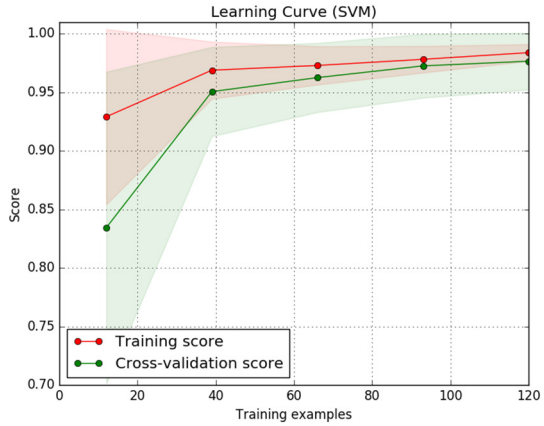

Figure 7b: Learning Curve
IRIS (svm, x_val = True)

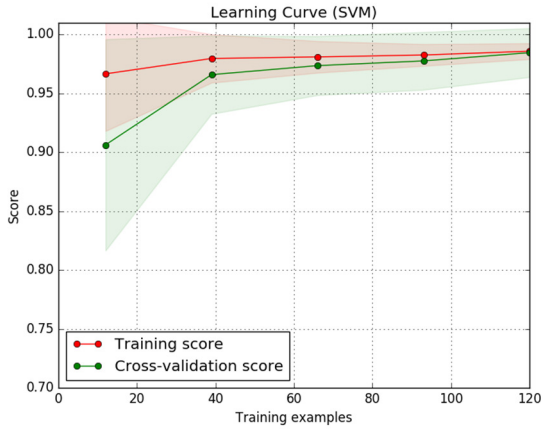

Figure 8a: Learning Curve
Data_2 (svm, x_val = False)

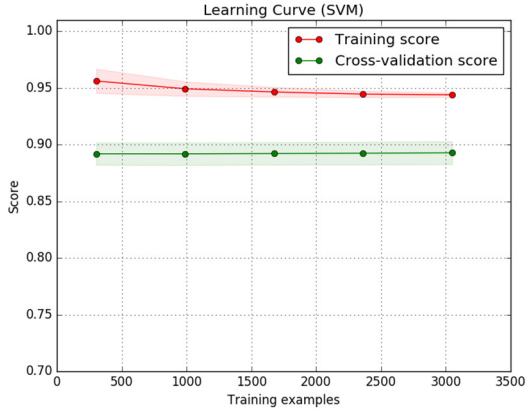

Figure 8b: Learning Curve
Data_2 (svm, x_val = True)

Unable to generate this learning curve with the computational resources available to me.

Figure 9a: Learning Curve
IRIS (Nrl Net, x_val = False)

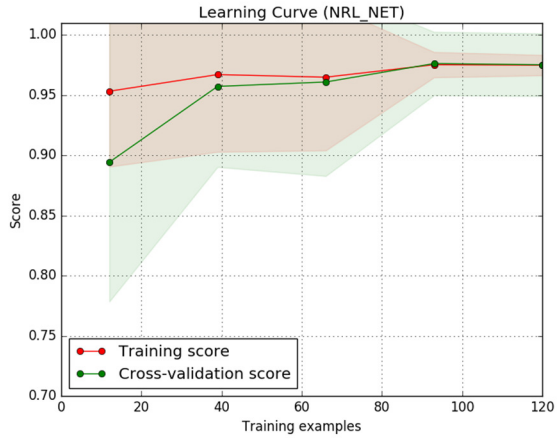

Figure 9b: Learning Curve
IRIS (Nrl Net, x_val = True)

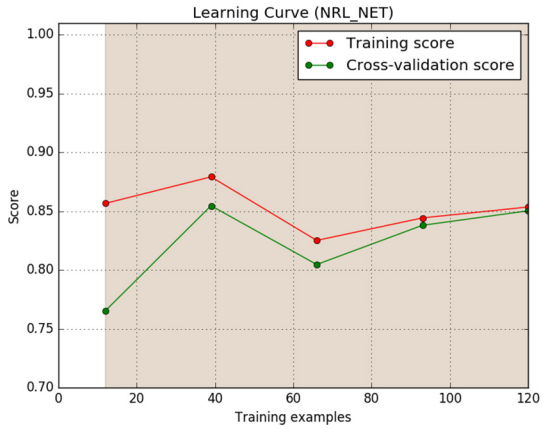

Figure 10a: Learning Curve
Data_2 (Nrl Net, x_val = False)

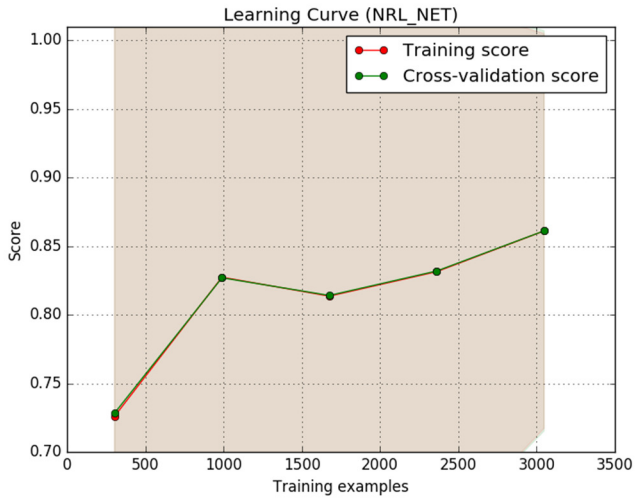

Figure 10b: Learning Curve
Data_2 (Nrl Net, x_val = True)

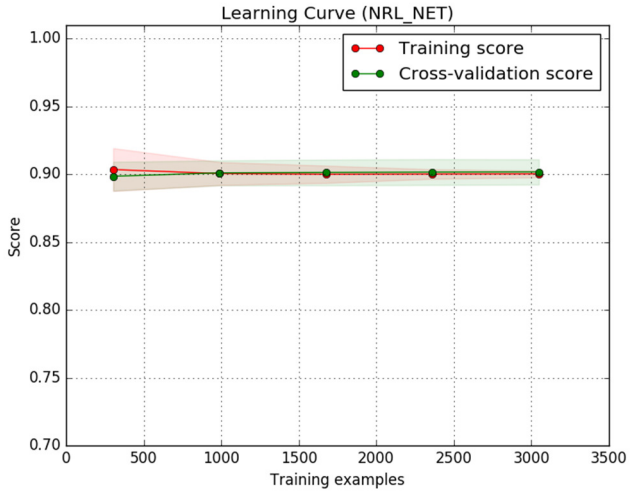

Table 3: Summery of Accuracy for learners: SVM, KNN and Neural Network. The results include: a measure of accuracy on the training set, the number of misclassified and correctly classified examples on the test set, and a measure of accuracy for the learners' generalization on the test set. The corresponding h-params, including tuned (x_val = True) are noted below the results for each learner. Note that only the h-params modified for classification are shown. The remaining parameters were default values.

Table 3 **KNN, SVM, Nrl Net**	**KNN**	**SVM** Note that `'degree'` only applies when 'kernel = poly'	**Nrl Net**
x_val (False)			
IRIS: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *96* *3* *72* *96*	 *96* *5* *70* *93*	 *99* *6* *69* *92*
DATA_2: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *89* *193* *1713* *90*	 *89* *206* *1700* *89*	 *89* *205* *1701* *89*
H_param	`{'n_neighbors': 5, 'weights': 'uniform', 'leaf_size': 30}`	`{'kernel': 'rbf', 'degree': 2}`	`{'activation': 'relu', 'learning_rate': 'constant', 'solver': 'adam'}`
x_val (True)			
IRIS: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *96* *3* *72* *96*	 *97* *2* *73* *97*	 *97* *2* *73* *97*

DATA_2: *Accuracy Train %* *# Misclassified* *# Correct* *Accuracy Test %*	 *90* *181* *1725* *91*	 *90* *194* *1712* *90*	 *90* *184* *1722* *90*
H_param	**IRIS**: {'n_neighbors': 7, 'weights': 'distance', 'leaf_size': 7} **Data 2**: {'n_neighbors': 190, 'weights': 'distance', 'leaf_size': 190}	**IRIS**: {'kernel': 'linear', 'degree': 2} **Data 2**: {'kernel': 'poly', 'degree': 2}	**IRIS**: {'activation': 'identity', 'learning_rate': 'constant', 'solver': 'adam'} **Data 2**: {'activation': 'logistic', 'learning_rate': 'constant', 'solver': 'lbfgs'}

CONCLUSION

Table 4: Summery of Accuracy for all learners: SVM, KNN and Neural Network. The results include: a measure of accuracy on the training set, the number of misclassified and correctly classified examples on the test set, and a measure of accuracy for the learners' generalization on the test set. The corresponding h-params, including tuned (x_val = True) are noted below the results for each learner. Note that only the h-params modified for classification are shown. The remaining parameters were default values.

From this table, notice that, in general, the SVM and Neural Network learners had the best generalization of the problems (data sets). This was most evident for the IRIS data set. Further exploration of tuning the hyper-parameters has the potential to yield even better results. Although not captured, the computation resources (CPU, MEM) of the decision trees with boosting, and certainly SVM with higher degrees of polynomial functions are to be considered. In my case, I was not able to run a poly fit greater than 2 degrees for a large data set. I was also unable to generate a learning curve for SVM (poly = 2 deg) with a large data set.

Table 4 **Summary of Accuracy**	**Dec Tree**		**Dec Tree (Rnd Frst)**		**Dec Tree (Bagged)**		**Dec Tree (Add Bst)**		**Dec Tree (Grd Bst)**		**KNN**		**SVM**		**Nrl Net**	
	IRIS	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**	**IRIS**	**DATA2**
Accuracy Train %	*96*	*90*	*93*	*89*	*95*	*90*	*96*	*89*	*88*	*85*	*96*	***90***	***97***	*90*	*97*	*90*
Accuracy Test %	*96*	*90*	*93*	*90*	*89*	*90*	*96*	*89*	*90*	*86*	*96*	***91***	***97***	*90*	*97*	*90*
*X_val (T / F / B) **	*T*	*F*	*T*	*T*	*T*	*F*	*T*	*T*	*T*	*F*	*B*	***T***	***T***	*T*	*T*	*T*

* x_val (T) means best score achieved with cross validation enabled, x_val (F) means best score achieved with cross validation disabled, x_val (B) means best score x_val (enabled == disabled).

V. APPENDIX

The use of computationally demanding learners (SVM: poly > 2) presented issues when solving problems with moderately sized data sets (Data_2). The scope of work covered in the appendix addresses that need. Accuracy, though considered, was not the primary concern. The ensemble included SVM and bagging. Bootstrap aggregating (bagging) is typically used to reduce overfitting, however, in this context, bagging was used to improve computational efficiency.

Additional Ensemble Methods for Computationally Demanding Learners

Table 5: Summery of Accuracy for an ensemble of SVM and bagging. As noted in the conclusion, I was unable to obtain results for problems (with a relatively large size, Data_2) using SVM with a poly fit greater than degree two. To address this issue, I combined SVM with bagging to distribute the work in solving the problem. Using the SVM plus bagging ensemble not only allowed SVM to compute a poly fit greater than degree 2, it enabled poly fits of at least degree 10. The computational efficiency was noticeable, solving problems with the SVM ensemble (poly = 10) returned results much faster than SVM alone (poly = 2). One word of caution, the degree of polynomial needs cross correlation to avoid underfitting / overfitting.

In fact, poly fits greater than 2 for the IRIS data set begin to show signs of both underfitting (poly2, poly8) and overfitting (poly3, poly5) for the IRIS data set. The deg of polynomial does not seem to induce either underfitting or overfitting for Data_2.

Although the objective for implementing an ensemble method was to improve computational efficiency, note that accuracy for Data_2 using ensemble SVM was on par with standalone SVM (see table 3, SVM column). Further, it is expected that similar computational efficiency would be achieved for ensembles of other learners, e.g., KNN and neural networks.

Table 5 **Ensemble SVM**	**Poly 2***		**Poly 3***		**Poly 5***		**Poly 8***		**Poly 10***	
	IRIS	DATA2	IRIS	DATA2	IRIS	DATA2	IRIS	DATA2	IRIS	DATA2
Accuracy Train %	*65*	*90*	*91*	*90*	*89*	*89*	*71*	*90*	*71*	*89*
# Misclassified	*22*	*184*	*15*	*194*	*15*	*190*	*19*	*192*	*48*	*187*
# Correct	*53*	*1722*	*60*	*1712*	*60*	*1716*	*56*	*1714*	*27*	*1719*
Accuracy Test %	*71*	*90*	*80*	*90*	*80*	*90*	*75*	*90*	*36*	*90*

* Degree of polynomial fit.

[1] scikit-learn: http://scikit-learn.org/stable/
http://www.scipy-lectures.org/index.html
http://scikit-learn.org/stable/tutorial/basic/tutorial.html
http://scikit-learn.org/stable/auto_examples/

http://scikit-learn.org/stable/supervised_learning.html#supervised-learning
http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html
http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_digits.html
http://scikit-learn.org/stable/modules/ensemble.html
http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html
http://scikit-learn.org/stable/modules/svm.html
http://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification

[2] pandas: http://pandas.pydata.org/pandas-docs/stable/10min.html
http://pandas.pydata.org/pandas-docs/stable/basics.html
http://pandas.pydata.org/pandas-docs/stable/dsintro.html
http://pandas.pydata.org/pandas-docs/stable/indexing.html
http://pandas.pydata.org/pandas-docs/stable/io.html

[3] Data: 1. The scikit-learn built-in iris data set
2. The UCI repository: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing