SUPERVISED LEARNING

1.Introduction

The goal of this study is to implement five supervised learning machine algorithms to analyze two classification problems. Supervised learning consists in learning the link between two datasets: the observed data X and an external variable y that we are trying to predict, usually called “label”, or “target”, or “class”.

The organization of this paper is as follows. Section 2 describes the algorithms implemented in this study. Section 3 discuss the classification problems, explains their importance, structure and required preprocessing steps. It also contains the results of the application of the learning algorithms and analysis. Section 4 contains conclusions.

2. Implementation

The machine learning algorithm were implemented using scikit-learn [1] and PyBrain Python libraries [2]. These algorithms are (1) Decision Tree (pruning mechanics are explained later in this Section 3, (2) Artificial Neural Network (BackPropagating Neural Network), (3) Boosting (Gradient Boositng, AdaBoost), (4) Support Vector Machines (“linear” and “rbf” kernals), (5) k-nearest Neighbor. In addition to these 5 main algorithms two more ensemble methods were implemented: Random Forest and K-Fold cross-validation. Brief description of the algorithms and calculations set-up follows.

Decision Tree (DT): **`DecisionTreeClassifier(Regressor)`** was used as a Decision Tree algorithm within scikit-learn library. The criterion I used for my DTs was an **`“enthropy`**” to achieve maximum informational gain at each split. DT classifier within scikit-learn library does not automatically support pruning, so, it was done “manually” as discussed in the following section.

Artificial Neural Networks (ANN): ANN algorithm was implemented within PyBrain Python library. **`BackpropTrainer`** was used to train the parameters of the neural network by back propagating the errors through time. The parameters that were varied during the study: number of **`hidden layers`**, **`learningrate`**, **`momentum`** and the number of iterations – **`epochs`**.

Boosting: **`GradientBoostingClassifier(Regressor)`** and **`AdaBoostClassifier(Regressor)`** are ensemble class algorithms that can be implemented

for both: regression and classification problems. They combine weak learners into a single strong learner, in an iterative fashion [3]

Support Vector Machines (SVM): SVM algorithms classify the examples of the separate categories by choosing the widest margin between those categories. SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Two kernels were used as parameters of the **`svm.SVC(SVR)`** classifier: 'liner' and 'rbf'. For the "rbf" kernel I studied the influence of two parameters: **`gamma`** and **`C`**. Gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' (an analogy can be made with high k in k-nearest neighbor algorithm) and high values meaning "close". The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth; while a high C aims at classifying all training examples correctly. Both parameters need to be used with care, since they can cause the overfitting.

k-nearest Neighbor (KNN): The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. In k-nearest neighbor algorithm the number of samples is a user-defined constant and is included in the **`KNeighborsClassifier(Regressor)`** as an **`n_neighbors`** parameter. In this kNN study I used both versions of the **`weights`** parameter: **`'uniform'`**, when the k-neighbors are uniformly weighted, and **`'distance'`**, when the weighting is inversely proportional to the distance to the nearest neighbors.

Additional ensemble methods: in **`RandomForestClassifier(Regressor)`** each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. When the number of features is low, the Random Forest doesn't perform enough splitting for effective averaging and can actually perform worse than the original Decision Tree algorithm, which was exactly my case. Another cross-validation method that was implemented in this study: **`KFold`** for different number of folds.

Notes on data preprocessing and standardization: For training and testing purposes the datasets were split in 75%-25% proportion. Standardization of datasets is a common requirement for many machine learning estimators (Gaussian with zero mean and unit variance). The standardization procedure is not an issue for decision trees because the relative magnitude of features does not affect the classifier performance and, thus, was not implemented for DT. However, it can affect significantly

the performance of the other algorithms (for example SVM) and was used throughout the study unless noted otherwise.

Notes on results representation: for Learning Curves I made a decision to plot an accuracy (score) rather than an error as a function of the dataset size. The reason is the consistency between the plots and the summary tables where I compared the performance of the algorithms against each other.

3. Results and Discussion

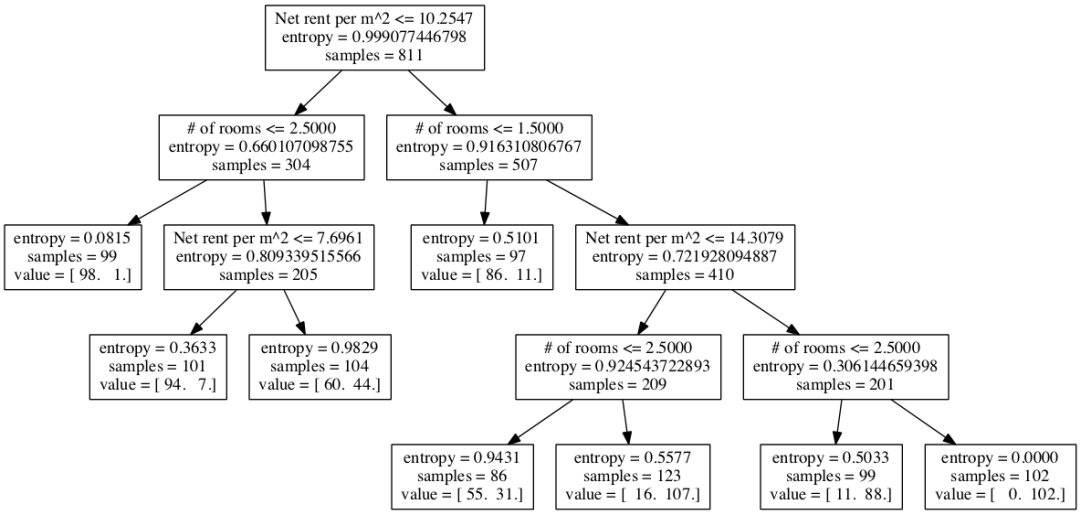
Classification problem #1 Munich rent standard 1994 [4].

The data was originally used to help tenants, landlords, renting advisory boards and experts to determine the local comparative rent rate in Munich, Germany in 1994. The dataset includes the answers of 1082 households (instances) interviewed over 17 questions (features). The idea is to be able to predict a rent price based on the different combinations of features. The features are of binary and continuous type and are listed as: Floor Space (m^2), Year of Construction, Bathroom in Appartment (y/n), Central Heating (y/n), Hot Water Supply (y/n), Good Address (y,n), Bad Address (y/n), Good Res. Area (y/n), Bad Res. Ares (y/n), Tiled Bathroom (y/n), Window type, Kitchen type, Lease duration, Age category, Floor Space Category, Net Rent (per m^2), Number of Rooms. Sorting features by their importance using Decision Tree algorithm and parameter feature_importance_ revealed two most important parameters based upon which the algorithm would make a prediction: Floor space and Net rent price per m^2. This makes total sense (and the problem trivial) since Rent = NetPrice x FloorSpace with the accuracy of prediction as high as 99.8% (with SVM "rbf" kernel). So, to get rid of triviality I have removed entirely the 'Floor Space' and 'Floor space category' data from the dataset, challenging the algorithms to use less obvious data to make an accurate prediction. Repeated feature importance analysis re-ranked the features in the next order: (1) Net rent per m^2: importance 0.3312, (2) Number of rooms: 0.2037, (3) Lease Duration: 0.1459, (4) Year of construction: 0.0858... etc. It worth noting, that in the absence of Floor Space data the algorithm picked up a Number of Rooms as a next most influential feature after the Net Rent. It seems logical, since the number of rooms can be indicative of the floor area. Another aspect that makes the study interesting is the opportunity to run it as a binary classification as well as a regression problem. I focused mostly on the binary Classification problem (rent higher or lower than 750 DM). For the sake of keeping the report within the page limit I included only the brief summary of the Regression version of the problem.

Decision Tree. The pruning was done "manually" by (1) visually assessing the decision tree plot and choosing the node with the least amount of entropy gain, (2) noting the number of the samples at that node and then (3) restricting the **`min_sample_leaf`** parameter to this number in

order to eliminate the node. Figure 1a shows the fully developed DT, where the node that was showing enthropy = 0.99 with 77 samples was eliminated by restricting the min_sample_leaf to 80. Thus, the tree has transitioned to its different much shallower version shown in Figure 1a.

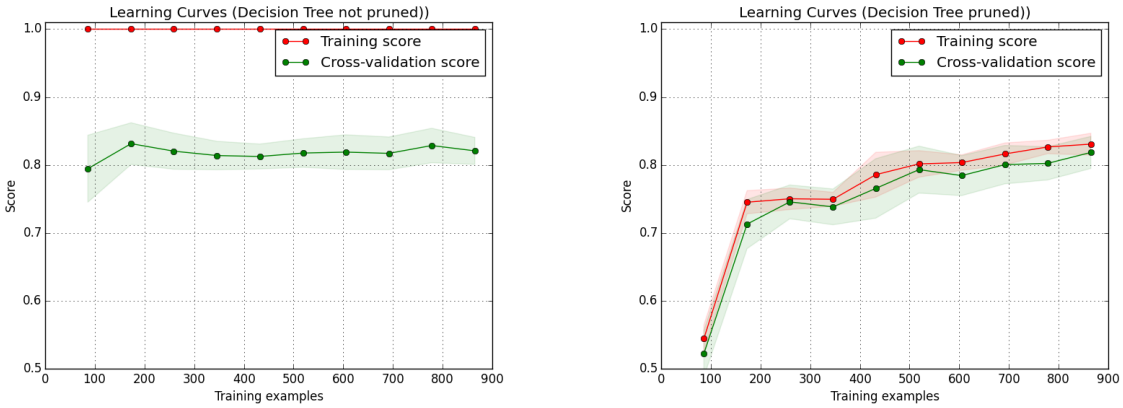
(a)



(b)

Fig. 1. Decision Tree algorithm representation (a) not pruned: min_sample_leaf = 1 max_depth = 30, (b) pruned: min_sample_leaf = 80.

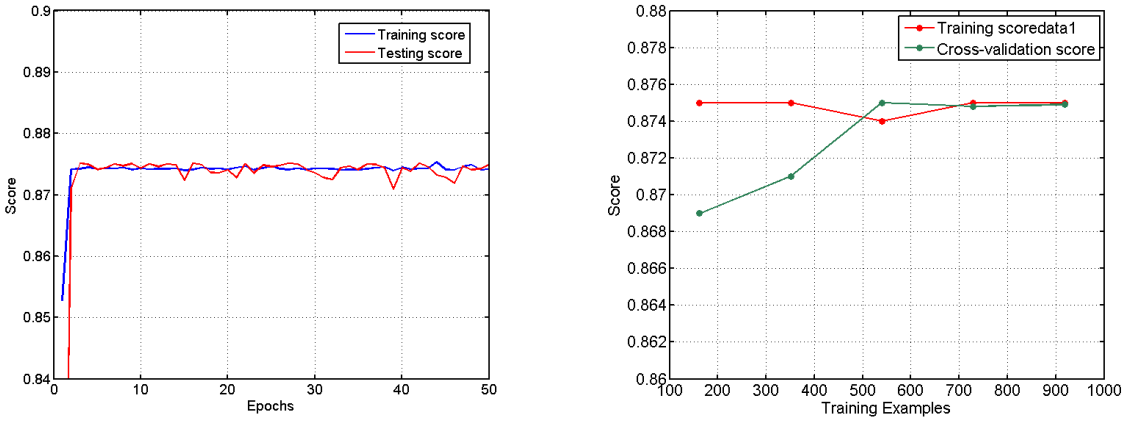
Good example of how pruning can take care of the overfitting is shown in Figure 2. Each datapoint on the learning curve is a result of averaging of 10 runs. The shaded green and red areas are standard error of the mean. Figure 2a shows the learning curves from not pruned tree from Figure 1a. Note the gap between training and cross-validation score, and most importantly, the fact that the scores are not converging with the dataset size increasing. I interpret this as a High Variance case. Removing low entropy gain nodes get rid of overfitting (Figure 2b). Both training and cross-validation scores are now the functions of the dataset size. The cross-validation score slightly decreased compare to not pruned version of DT (from 0.85 to 0.83), however, the tree became much smaller and if we would have a larger dataset for training, the accuracy would improve accordingly.



(a) (b)

Fig. 2. Learning curves for the Decision Tree algorithm at different depth and splitting parameters: (a) fully developed tree, (b) pruned and shallower tree.

Artificial Network. The algorithm was executed for 3 hidden layers, learning_rate = 0.01, and momentum=0.001. Figure 3(a) shows the evolution of the accuracy on training and validation sets as a function of epochs (iterations). Figure 3(b) shows the Learning curves. The algorithm reaches target accuracy within several iterations, and Learning Curves show training and cross-validation scores converging at the training test size around 500. Near this point the testing set accuracy exceeds the training score, I attribute this to the deviation around the mean for each test point (note how nosy the testing score is in Figure 2(a) and in the case of ANN algorithm I did not do the averaging over 10 runs at each datapoint). So far ANN algorithm demonstrated good performance at the accuracy 0.87 but rather longer running time (~82 s) for 50 epoches.

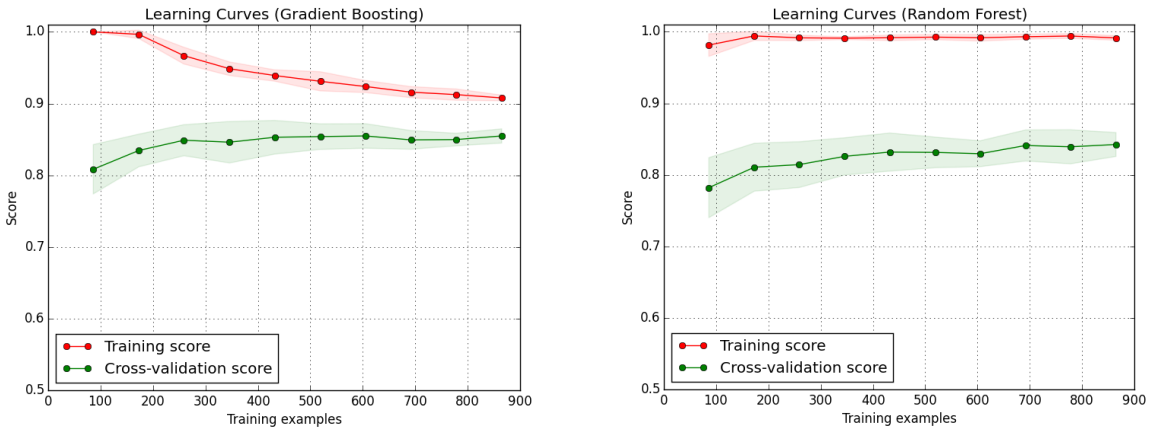


(a) (b)

Fig. 3. Artificial Neuron Network (a) score evolution as a function of time, (b) Learning Curves

Ensemble algorithms. Figure 4 shows learning curves from two ensamble algorithms: (a) Gradient Boosting and (b) Random Forest. Gradient Boosting algorithm gives a descent accuracy of 0.87. The fact that the cross-validation score flattens out for training examples >300 indicates that the

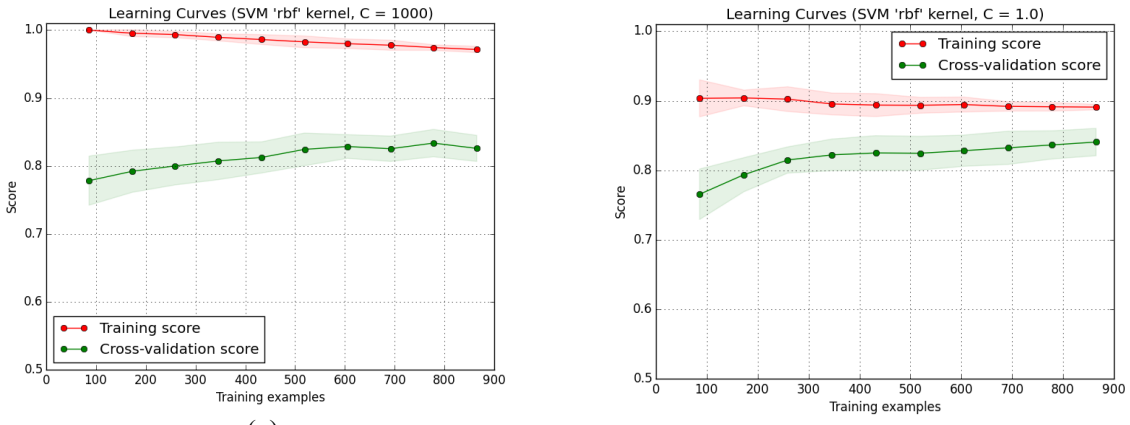
algorithm has reached its capacity and increasing the dataset size will not further affect the performance. The relative inefficiency and tendency to overfit of the Random Forest approach (see Figure 4(b) can be interpreted as a result of the small size of the feature space (15 features) making the averaging not that effective.



(a) (b)

Fig. 4. Learning Curves for (a) Gradient Boosting, and (b) Random Forest.

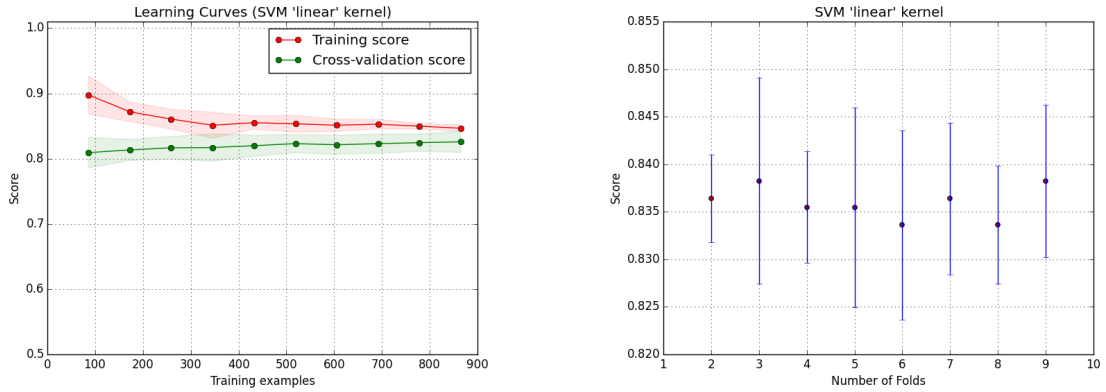
Support Vector Machine (SVM), 'rbf' kernel. I started the experiment with high value for parameter C= 1000 (Figure 5a). The training data fitted almost perfectly, while the testing data shows lower accuracy and does not decrease as a function of the dataset size: large C parameter to blame for. Reducing C from 1000 to 1 helps partially alleviate the problem (see Figure 5b). Now the training and cross-validation errors are converging, so including more data could possibly bring the accuracy even higher. I also experimented with gamma parameter (not shown) just to find out that decreasing gamma (to increase the "sensitivity" of the algorithm) rather hurts the performance in the same manner as high C.



(a) (b)

Fig. 5. Rbf kernel SVM (a) Learning Vurves at C=1000, (b) Learning Curves at C=1.

SVM ('linear kernel') By looking at the Fig 6a one can see that the performance of the SVM algorithm on training and testing set flattens and almost doesn't change with the size > 400. The accuracy even on the training set is lower than with other algorithms. I interpret it as a High Bias problem and insufficient hypothesis space. That means the dataset's classes are not precisely linearly separable and no matter how large our training dataset would be the algorithm would never be able to perform better than 0.84. Figure 6b shows the results of application of the K-fold method (accuracy as function number of folds), which does affect the score, and again, proves that linear kernel SVM algorithm is not a best classifier to apply to this particular dataset.



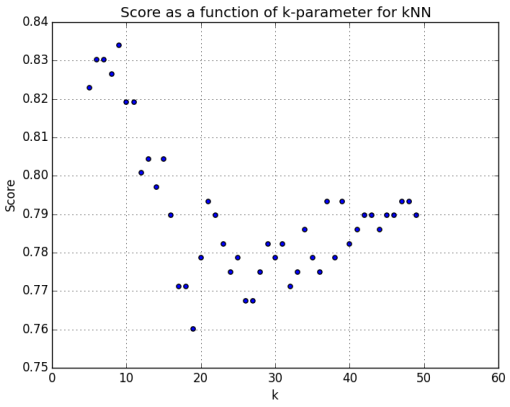
(a) (b)

Fig. 6. Linear kernel SVM (a) Learning Curves, (b) score as a function of k-folds in KFold method.

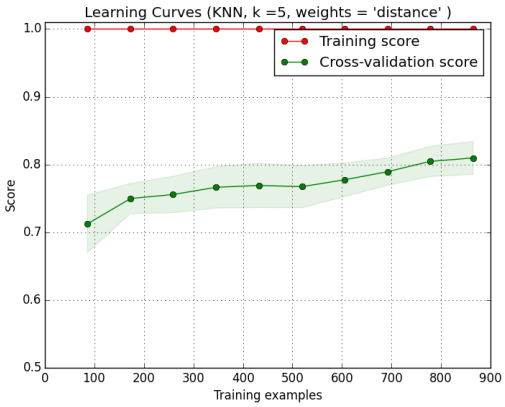
Figure 7a shows the performance of kNN algorithm as a function of k. I found it curious that the performance sinks noticeably around k=25 but picks up later for k >30. In theory, increasing the number of neighbors can also decrease the complexity of the problem and bring the accuracy up [5], but in the case of Munich Rent dataset the best performance is still at low k-numbers. Figure 7(b) shows the learning curve for kNN algorithm using 5 neighbors with weighted parameters (distance). The gap between train and cross-validation score is large, however, the fact that cross-validation accuracy is still increasing with the dataset size, indicates that larger dataset could potentially fix the problem. Figure 7c shows the result of changing the weight parameter to "uniform" and thus, oversimplifying the model (performance of the testing set does not improve either). Between these two far from ideal cases I would still prefer the High Variance (Fig. 7b). My argument is: in case of much larger data set we would have a chance of improving the cross-validation score which is not bounded from above by the performance of the oversimplified model like in Fig. 7c.

The running times and accuracy of the best performing versions of the supervised learning algorithms implemented to study Munich Rent 1994 dataset are summarized in Table I. Each run

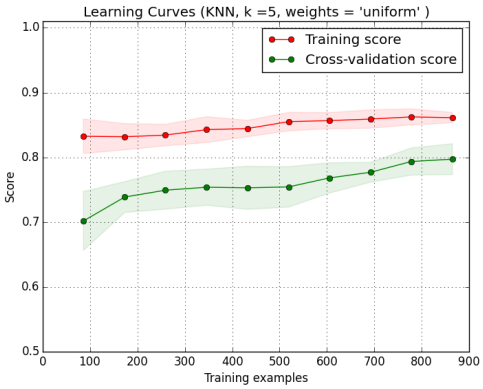
(except ANN) also produced classification report similar the one shown in Table II. Confusion matrices for the algorithms showed a very close guess for both binary classes, which means the algorithms were capable to predict relatively accurate both high and low rent rates. With the parameters tuned for each algorithm, the resulting accuracy is within 83-87%.



(a)



(b)



(c)

Fig. 7. *k*-nearest neighbors: (a) *k*=5, weights = 'distance', (b) *k*=5, weights = 'uniform', (c) score as a function of *k*.

Table I. Summary of runs for Munich Rent 1994 dataset Classification.

Algorithm	*Running time (training + prediction) (s)*	*Accuracy (score)*
Decision Tree	0.05	0.83
Boosting Gradient 100 estimators	0.27 for 100 of weak learners	0.86
SVM rbf	0.06	0.86
SVM linear	0.03	0.84
5NN (weight=distance)	0.02-0.30 (depends on #k)	0.83
ANN (3 layers)	82	0.87
Random Forest	100 estimators	0.85

Table II. Classification report and Confusion matrix for Decision Tree Algorithm.

DT classification report	precision	recall	f1-score	Support samples	Confusion Matrix
0 (< 750 DM)	0.86	0.86	0.86	127	[[109 18]
1 (>= 750 DM)	0.88	0.88	0.88	144	[18 126]]
Total	0.87	0.87	0.87	271	

Similar analysis for all but ANN algorithm was done using the Regressive form of the dataset (target is a list of continuous numbers representing rent price). The summary of the best runs, shown in Table III. The accuracy of the Regression study suffered compare to the Classifier version, presumably due to the increase of the complexity. The accuracy is within 61-76%.

Table III. Summary of runs for Munich Rent 1994 dataset Regression.

Algorithm	*Running time in seconds*	*Accuracy (score)*
Decision Tree (depth 5, leafs15)	0.02	0.76
Boosting gradient, 100 estimators, learning rate = 1	0.09	0.76
Adaboost, 100 estimators	0.18	0.67
SVM rbf, C =100	0.09	0.7
SVM linear	0.08	0.67
5NN, weights = 'distance'	0.02	0.61

Classification problem #2: recognizing Olivetti faces [6].

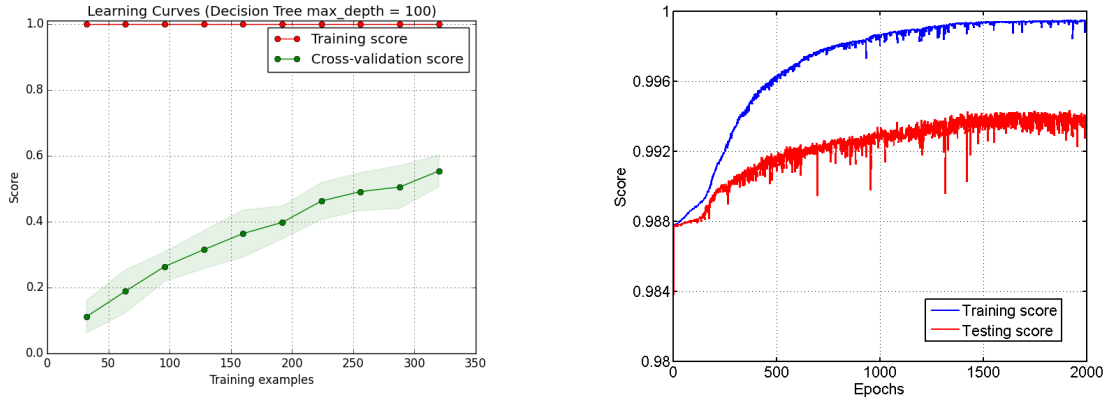
The dataset consists of the images of 40 people (10 images per person). The partial snapshot of Olivetty faces is shown in Figure 8. Image recognition problem is usually associated with clustering or unsupervised type of learning. Here, I am applying the supervised learning concept to train the algorithm on the partial dataset to see if it will recognize the person when offered an unseen image. The features upon which the algorithm is learning are 8-bit integers, associated with pixels. Each of 400 images consists of 64x64 pixel matrix. This makes a feature space quite substantial (4096) and the problem interesting to study.

Fig. 8. Olivetti faces.

Decision Tree. Since I need to classify 40 people, my DT classifier should be at least 39 layers deep to classify exactly one person per node. The ideal DT would be long but skinny. In practice there were some additional branching (but overall the tree stayed long and compact). Learning curves for DT shallower than 40 layers were showing underfitting (as expected). After the max_depth parameter was set up to >40 the underfitting went away, however, the accuracy still suffered and stayed close to the chance. Looking at the Learning Curve (Figure 9a) I make a

conclusion that having larger dataset (more images per person), could improve the accuracy of the algorithm, given the fact that cross-validation score is increasing steeply with the dataset size.

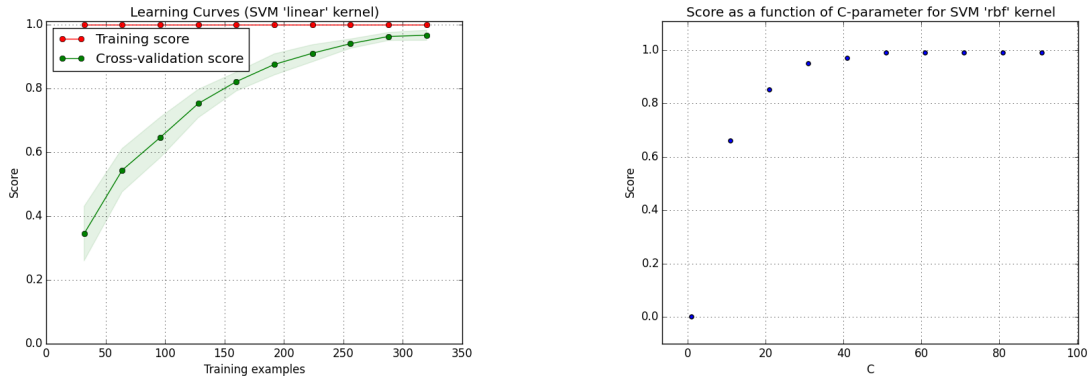
ANN algorithm was applied to classify the Olivetti faces problem using 25 hidden layers with learning rate of 0.01 and momentum 0.01. Figure 9b shows the evolution of the accuracy of the training and validation sets as a function of epochs (iterations). Both training and cross-validation scores start saturating around epochs = 1500. ANN demonstrated perfect performance with the score of 0.994, however the running time suffered (8100s for 2000 epochs).



(a) (b)

Fig. 9. (a) Learning Curves for DT classifier; (b) ANN score evolution.

SVM algorithms were expected to demonstrate the best performance, since they are well suited to deal with high-dimensional problems. Figure 10a shows the Learning Curve for the 'linear' kernel SVM with the score of 0.99. The 'rbf' kernel is similarly effective, but need to be run at high values of parameter C > 50, to increase complexity of classification surface (see Figure 10b).



(a) (b)

Fig.10. SVM Learning Curves for 'linear' kernel, (b) 'rbf' kernel, score as a function of C-parameter

For kNN algorithm I started my analysis by plotting the performance as a function of k. Since there only 10 images per person, the best chance to guess right would be by "looking" at the immediate neighbor and disregard the information that comes from the far neighbors, I was expecting the best accuracy to be associated with fairly small k numbers (confirmed in Figure 11a). The

algorithm performs slightly better if k neighbors are weighted with the distance, which even more restricts the influence of the further neighbor (Figure 11b and c).

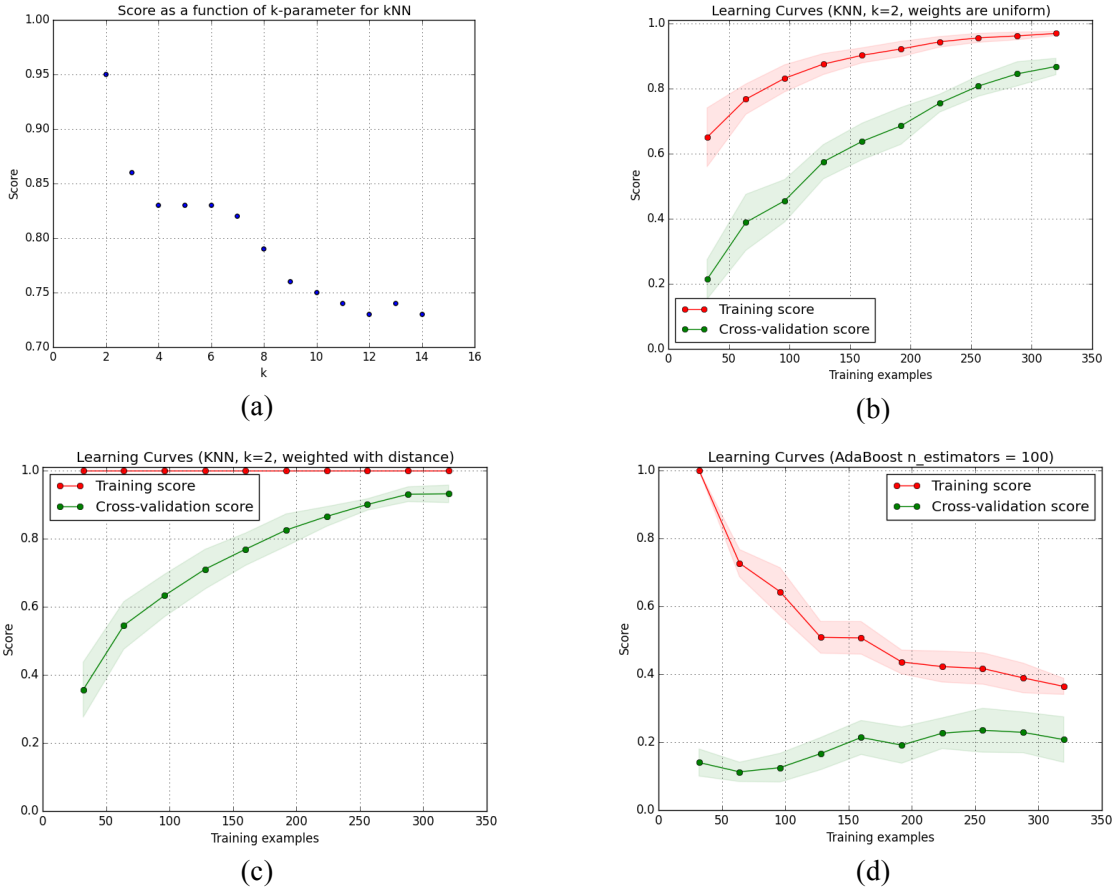


Fig. 11. kNN algorithm (a) score as a function of k, (b) learning curves at uniform weights, (b) Learning Curves when weighted with distance.

None of the Boosting algorithms Gradient Boosting or AdaBoost could perform better than a chance (see Figure 11d). The algorithms were tried with different learning rates and numbers of estimators (up to 1000) without much success. I would speculate that when the boosting algorithms are trying to classify the person based on 10 images only, the weak learning approach is simply not effective enough and high bias/low variance classifiers would naturally perform better.

The running times and accuracy of the algorithms implemented to study Olivetti faces dataset are summarized in Table IV. Best performing algorithm based on these two factors is SVM.

Table IV. Summary of runs for Olivetti faces Classification.

Algorithm	*Running time (training + prediction) (s)*	*Accuracy (score)*
Decision Tree	2.6	0.56
GradientBoost and AdaBoost	10.8-55.8	0.23
SVM rbf	2.3	0.99
SVM linear	2.0	0.99

2-nearest neighb. (weighted)	0.3	0.86
ANN	80343 for 2000 epochs	0.99

4. Conclusions

The analysis of the performance of 5 learning algorithms applied to study two classification problems are presented. I chose the datasets that are different in problem formulation (questionnaire vs image recognition) as well as size and feature types. Munich Rent problem was run as classification and regression, even though I focused mostly on the classification version. I compare the results of the study in Table V.

	Munich Rent	*Olivetti faces*
Problem formulation	Prediction of the rent price based on questionnaire	Recognizing person from the unseen image based on available training set
Classification type	Binary classification, regression	Multi-class classification
Dataset size	[1080,15]	[400,4096]
Features type	Binary, continuous	Continuous
Best algorithm	Boosting - score 0.86, (ANN good score of 0.87 but long running time)	SVM - score 0.99! (ANN score of 0.99 but long running time)
Comments	The Learning Curves for DT, SVM and kNN algorithms suggest the insufficient dataset size.	Proves the fact that SVM is well suited to deal with large feature spaces. Boositng algorithms could potentially do better on a larger dataset (more images per person).

References

[1] Scikit-learn: http://scikit-learn.org/stable/index.html

[2] PyBrain http://pybrain.org

[3] R. Schapire et al, The Boosting Approach to Machine Learning. An Overview, MSRI Workshop on Nonlin. Est. and Classific., (2002)

[4] Munich rent standard 1994, Department of Statistics, University of Munich, Germany: http://www.statistik.lmu.de/service/datenarchiv/miete/miete_e.html

[5] W. Richert, L.P. Coelho, Building Machine Learning Systems with Python, Packt Publ. Birmingham-Mumbai, (2013)

[6] The Olivetti faces dataset, AT&T Laboratories Cambridge, available through scikit-learn: http://scikit-learn.org/stable/datasets/olivetti_faces.html