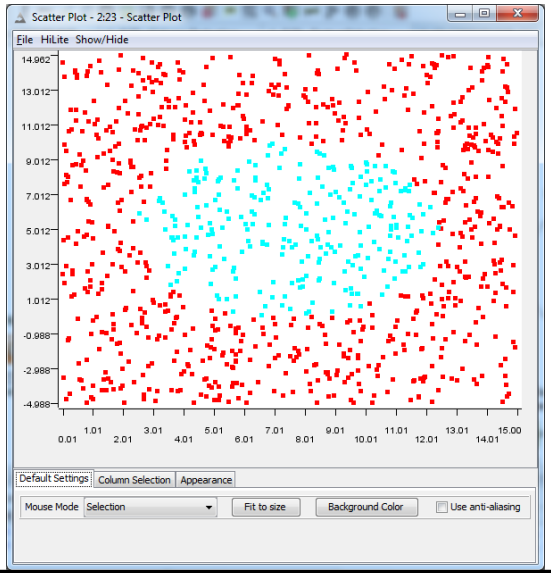
Data Mining

Unit # 6

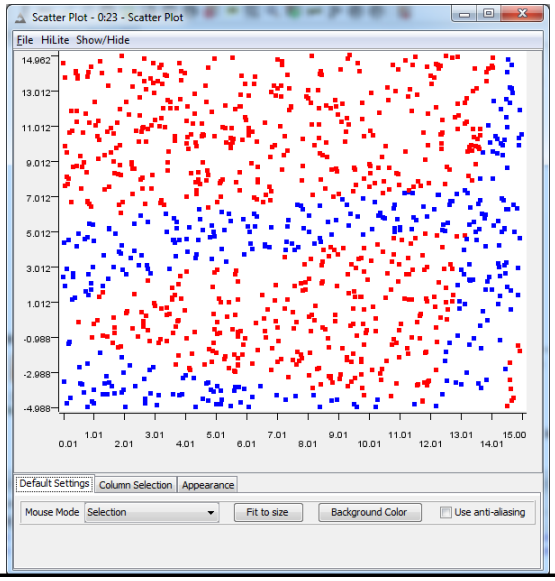
Nonlinear Classification

- Classes may not be separable by a linear boundary
- Suppose we randomly generate a data set as follows:
  - X has range between 0 to 15
  - Y has range between -5 to 15



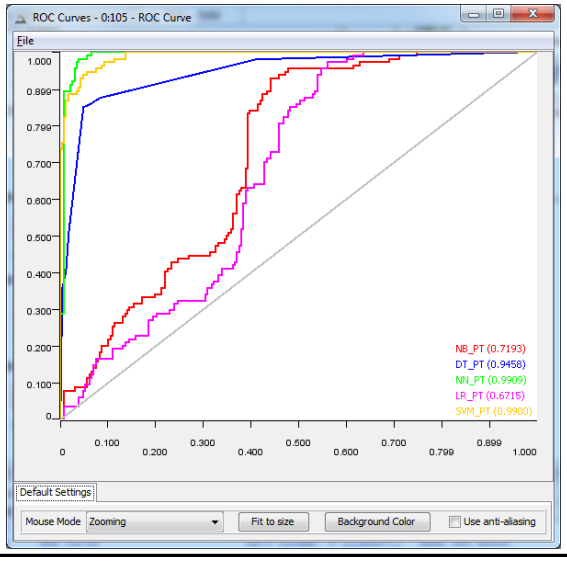
Parabola Like Function

- True in Blue color
- False in Red Color

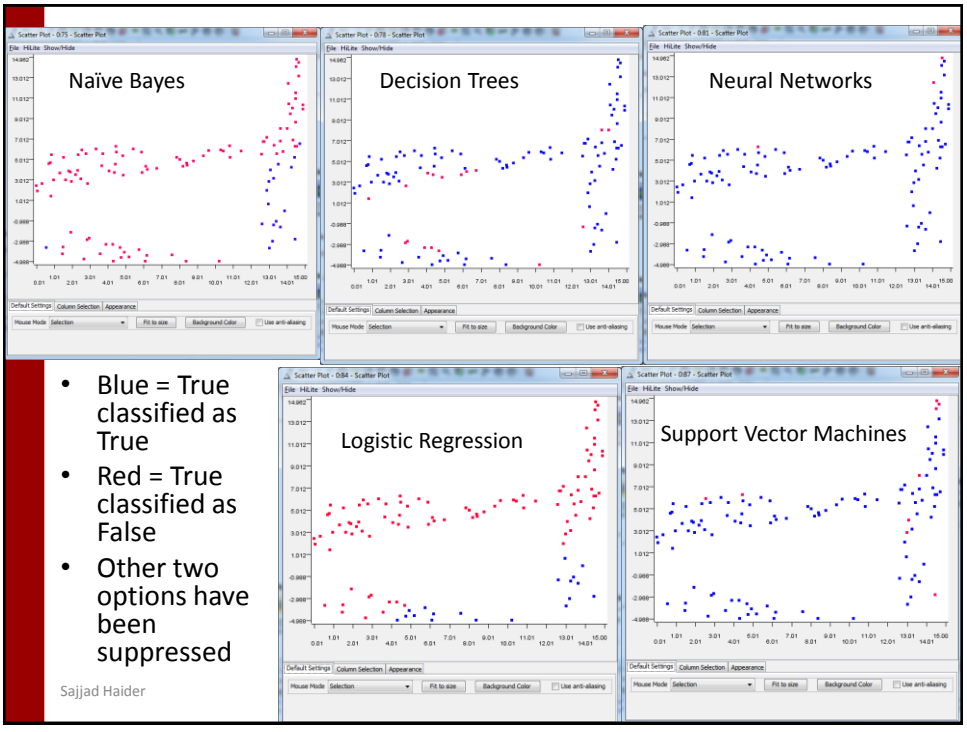


Sajjad Haider

ROC Curves of 5 Classifiers



Sajjad Haider

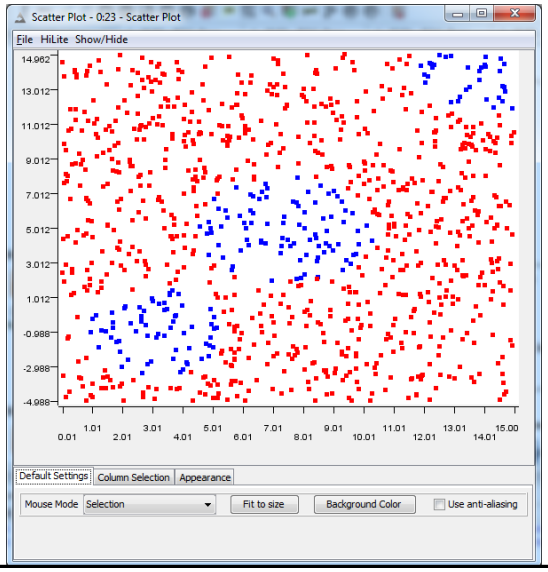


- Blue = True classified as True
- Red = True classified as False
- Other two options have been suppressed

Sajjad Haider

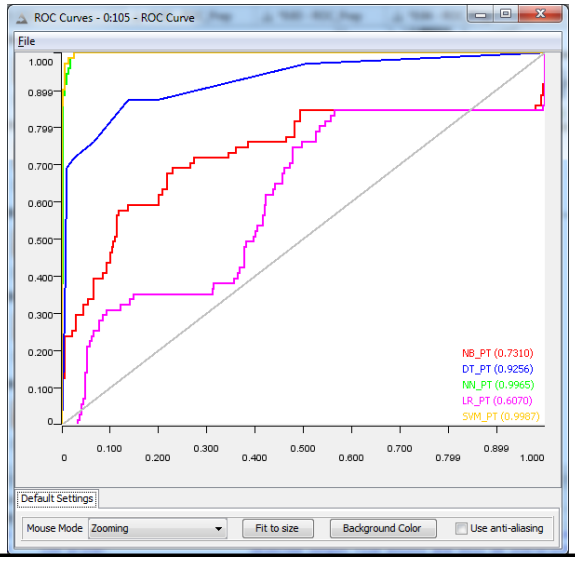
Multiple Circles

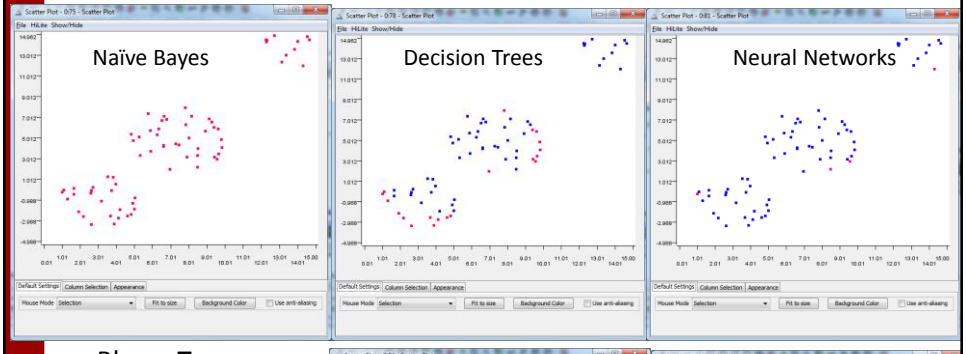
- True in Blue color
- False in Red Color



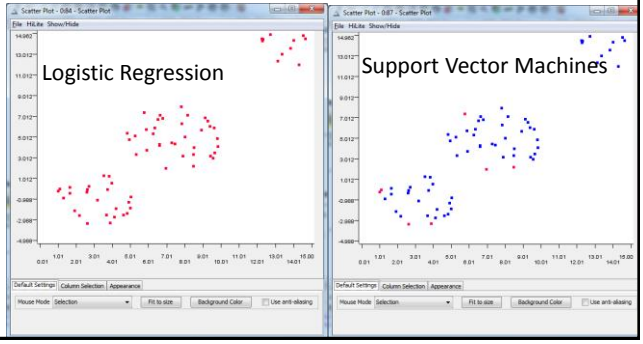
Sajjad Haider

ROC Curves of 5 Classifiers



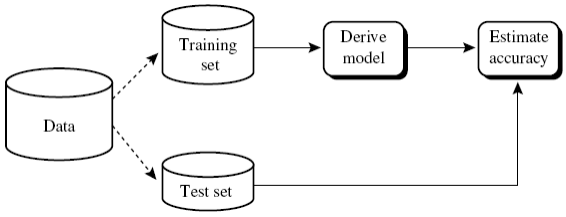


- Blue = True classified as True
- Red = True classified as False
- Other two options have been suppressed



Evaluating the Accuracy of a Classifier

- Holdout, random subsampling, crossvalidation, and the bootstrap are common techniques for assessing accuracy based on randomly sampled partitions of the given data.



Holdout Method

- The holdout method is what we have alluded to so far in our discussions about accuracy.
- In this method, the given data are randomly partitioned into two independent sets, a training set and a test set.
- Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set.
- The training set is used to derive the model, whose accuracy is estimated with the test set.
- The estimate is pessimistic because only a portion of the initial data is used to derive the model.

Performance Evaluation

- We evaluate performance of the classifier on the testing set
- With large labeled dataset, we would typically take 2/3 for training, 1/3 for testing
- What can we do if we have a small dataset?
  - Can't afford to take 1/3 for testing
  - Small testing means predicted error will be far from true error

Cross Validation

- Cross Validation helps to get a more accurate performance evaluation on "small" dataset
- K-fold Cross Validation
  - Divide the labeled data set into k subsets
  - Repeat k times:
    - Take subset i as the test data and the rest of subsets as the training data. Train the classifier and asses the testing error on subset i.
  - Average the testing error from the k iterations
- Leave one out Cross Validation
  - Cross validation with k = 1

Bootstrap

- The bootstrap method samples the given training tuples uniformly with replacement.
- That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set.
- There are several bootstrap methods. A commonly used one is the .632 bootstrap, which works as follows.
  - Suppose we are given a data set of d tuples.
  - The data set is sampled d times, with replacement, resulting in a bootstrap sample or training set of d samples.

Bootstrap (Cont'd)

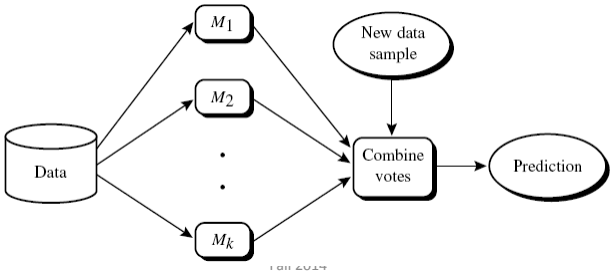
- It is very likely that some of the original data tuples will occur more than once in this sample.
- The data tuples that did not make it into the training set end up forming the test set.
- Suppose we were to try this out several times.
- As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap, and the remaining 36.8% will form the test set

Bootstrap (Cont'd)

- We can repeat the sampling procedure k times, where in each iteration, we use the current test set to obtain an accuracy estimate of the model obtained from the current bootstrap sample.
- The overall accuracy of the model is then estimated as $Acc(M) = (i=1 \text{ to } K) \Sigma (0.632 \times Acc(Mi)_{train\_set} + 0.368 \times Acc(Mi)_{test\_set})$
- where $Acc(Mi)_{test\ set}$ is the accuracy of the model obtained with bootstrap sample I when it is applied to test set i. $Acc(Mi)_{train\ set}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to the original set of data tuples.
- The bootstrap method works well with small data sets.

Ensemble Models

- Bagging and boosting are examples of ensemble methods, or methods that use a combination of models.
- Each combines a series of k learned models (classifiers or predictors), M1, M2, …, Mk, with the aim of creating an improved composite model, M*.
- Both bagging and boosting can be used for classification as well as prediction



Bagging

- The name *Bagging* came from the abbreviation "Bootstrap AGGregatING" (1996). As the name implies, the two ingredients of Bagging are *bootstrap* and *aggregation*.
- Baggin applies bootstrap sampling to obtain the data subsets for training the base learners.
- Given a training data set containing *m* number of examples, a sample of *m* will be generated by *sampling with replacement*.

Bagging (Cont'd)

- By applying the process *T* times, *T* samples of *m* training examples are obtained.
- Then for each sample a base learner can be trained by applying a learning algorithm.
- Bagging adopts the most popular strategies for aggregating the outputs of the base learners, that is, voting for classification and averaging for regression.

Algorithm: Bagging. The bagging algorithm—create an ensemble of models (classifiers or predictors) for a learning scheme where each model gives an equally-weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a learning scheme (e.g., decision tree algorithm, backpropagation, etc.)

Output: A composite model, $M*$.

Method:

(1) for $i = 1$ to k do // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i to derive a model, M_i;
(4) endfor

To use the composite model on a tuple, X:

(1) if classification then
(2) let each of the k models classify X and return the majority vote;
(3) if prediction then
(4) let each of the k models predict a value for X and return the average predicted value;

Bagging (Cont'd)

- The bagged classifier often has significantly greater accuracy than a single classifier derived from D, the original training data.
- It will not be considerably worse and is more robust to the effects of noisy data.
- The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.
- For prediction, it was theoretically proven that a bagged predictor will always have improved accuracy over a single predictor derived from D.

Boosting

- In boosting, weights are assigned to each training tuple.
- A series of k classifiers is iteratively learned.
- After a classifier Mi is learned, the weights are updated to allow the subsequent classifier,Mi+1, to "pay more attention" to the training tuples that were misclassified by Mi.
- The final boosted classifier, M, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.

Algorithm: Adaboost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) for $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Equation 6.66)
(6) if $error(M_i) > 0.5$ then
(7) reinitialize the weights to $1/d$
(8) go back to step 3 and try again;
(9) endif
(10) for each tuple in D_i that was correctly classified **do**
(11) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(12) normalize the weight of each tuple;
(13) endfor

2

Boosting Algorithm (Cont'd)

To use the composite model to classify tuple, X:

(1) initialize weight of each class to 0;
(2) for $i = 1$ to k do // for each classifier:
(3) $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) endfor
(7) return the class with the largest weight;

Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially distribution of weights is uniform
- At successive iterations, the weight of misclassified examples is increased, forcing the weak learner to focus on the hard examples in the training set.

Bagging vs. Boosting

- *Because of the way boosting focuses on* the misclassified tuples, it risks overfitting the resulting composite model to such data.
- Therefore, sometimes the resulting "boosted" model may be less accurate than a single model derived from the same data.
- Bagging is less susceptible to model overfitting.
- While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

Random Forest

- Random Forest (2001) is a representative of the state-of-the-art ensemble methods. It is an extension of Bagging, where the major difference with Bagging is the incorporation of randomized feature selection.
- During the construction of a component decision tree, at each step of split selection, RF first randomly selects a subset of features, and then carries out the conventional split selection within the selected feature subset.

Random Forest (Cont'd)

- A parameter K controls the incorporation of randomness. When K equals the total number of features, the constructed decision tree is identical to the traditional decision tree; when K = 1, a feature will be selected randomly.
- The suggested value of K is the logarithm of the number of features.
- Notice that randomness is only introduced into the feature selection process, not into the choice of split points on the selected feature.

Stacking

- Stacking is a general procedure where a learner is trained to combine the individual learners.
- Here, the individual learners are called the *first-level learners*, while the combiner is called the *second-level learner*, or *meta-learner*.
- The basic idea is to train the first-level learners using the original training data set, and then generate a new data set for training the second-level learner, where the outputs of the first-level learners are regarded as input features while the original labels are still regarded as labels of the new training data.

Stacking (Cont'd)

- The first-level learners are often generated by applying different learning algorithms, and so, stacked ensembles are often heterogeneous, though it is also possible to construct homogeneous stacked ensembles.

Stacking (Cont'd)

- In the training phase of stacking, a new data set needs to be generated from the first-level classifiers. If the exact data that are used to train the first level learner are also used to generate the new data set for training the second-level learner, there will be a high risk of overfitting.
- Hence, it is suggested that the instances used for generating the new data set are excluded from the training examples for the first-level learners.

KNIME Demo

Ensemble Model