Project 3 Classification Algorithms

December 02, 2017

CSE 601 Data Mining and Bioinformatics

Group 31

Debanjan Paul	50206326	dpau7
Jay Bakshi	50206954	jaybaksh
Shivam Gupta	50206323	sgupta33

Results recorded

Dataset	Results/ Measures	Folds	Algorithms				
			K-nn	Naive Bayes	Decision Tree	Random Forest	AdaBoost
1 k=9	Accuracy	10	0.929605263158	0.822431077694	1.0	0.92772556391	
	Precision		0.929605263158	0.822431077694	1.0	0.92772556391	
	Recall		0.874846279452	0.977119047619	1.0	0.844841561947	
	F1		0.901629204129	0.803205649613	1.0	0.891080332555	
2 k=9	Accuracy	10	0.638714153562	0.681868640148	0.623681776133	0.697132284921	
	Precision		0.638714153562	0.681868640148	0.623681776133	0.697132284921	
	Recall		0.310041218242	0.564717915458	0.431138744297	0.337746175393	
	F1		0.360146084501	0.547445688865	0.436708430458	0.416787961681	
3-train k=9	Accuracy	10	0.9125			0.8625	
	Precision		0.9125			0.8	
	Recall		0.876666666667			0.673333333333	
	F1		0.899047619048			0.757222222222	

3-test k=9	Accuracy	-NA-	1.0			0.8375	
	Precision		1.0			0.7625	
	Recall		1.0			0.61	
	F1		1.0			0.716818181818	
4	Accuracy			0.75	1.0		
	Precision			0.75	0.8		
	Recall			0.8	0.8		
	F1			0.766666666667	0.8		

Our conclusions

Based on the performance figures as measured for the algorithms for classifying the provided datasets, we found the following algorithms to perform best with each dataset:

Dataset 1: Decision Tree, KNN and Random Forest

Dataset 2: Overall we did not achieve a good accuracy, but Random Forest provides marginally better results.

Dataset 3: KNN

Dataset 4: Decision Tree

Classification Basic Concepts

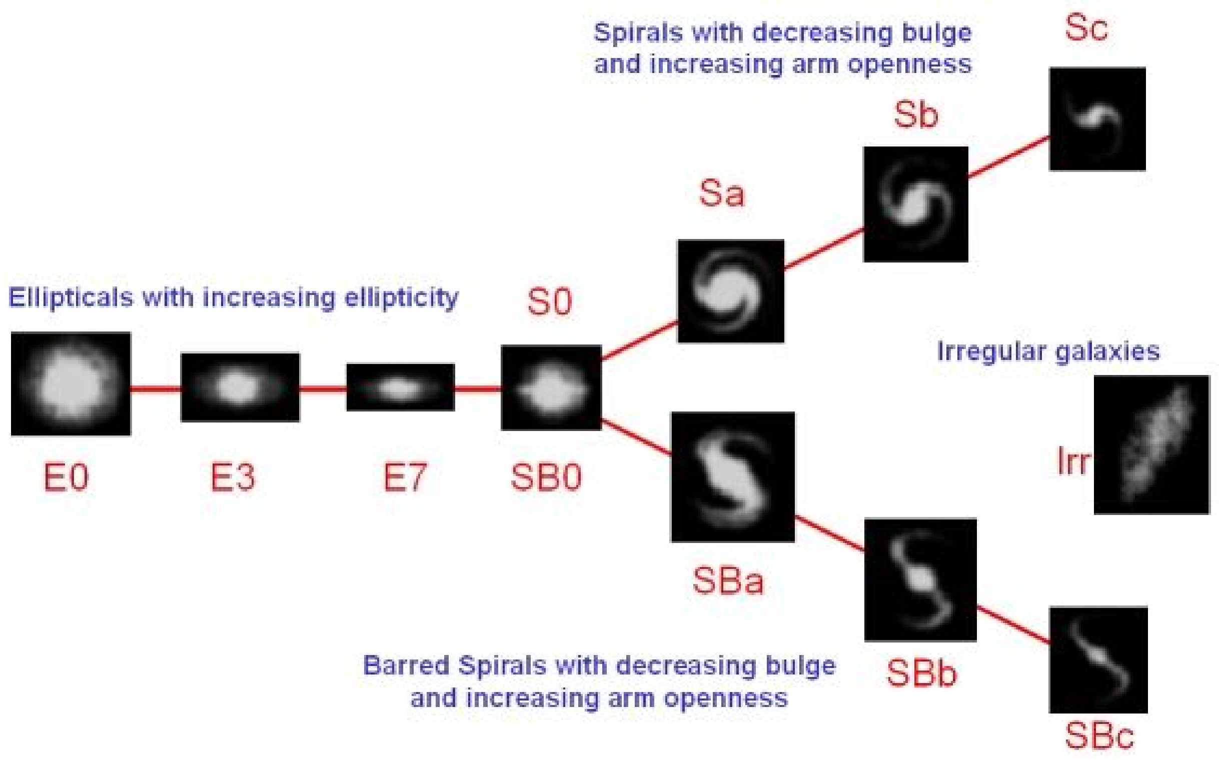

Spirals with decreasing bulge and increasing arm openness
Sc
Sb
Sa
Ellipticals with increasing ellipticity
S0
Irregular galaxies
Irr
E0
E3
E7
SB0
SBa
Barred Spirals with decreasing bulge and increasing arm openness
SBb
SBc

Fig. Hubble classification for galaxies. Source: https://astronomy.swin.edu.au/cosmos/H/Hubble+Classification

Definition

Task to assign current object to one of the several predefined categories.

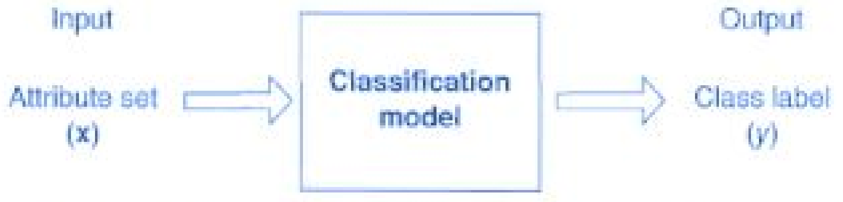


Figure 4.2. Classification as the task of mapping an input attribute set **x** into its class label *y*.

Examples include detecting spam email messages based on the message content and headers, classifying malignant and benign cells based results of MRI scans. And as seen in the figure above classifying galaxies based upon their shapes.

1. Input data is called a collection of **records**.
2. Each record is a **tuple (x,y)** where x is a set of attributes and y is class label to which the attributes belong to.
3. **Classification** techniques require the class labels to be **discrete**. While, **continuous** classes are handled by **Regression** techniques.
4. Acronym **CART** stands for Classification and Regression Techniques.
5. Classification techniques are better suited for datasets with **Binomial or Nominal categories.**
6. A key objective of the learning algorithm is to build models using training data, with good generalization capabilities that can accurately predict the class labels of previously unknown records.
7. Models can be measured for their **effectiveness** and compared as such with metrics like:
    a. $$\boxed{\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}}$$
    b. $$\boxed{\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}}}$$
8. The better model has **high accuracy** and **low error** rate.

K-nn Algorithm and Implementation

Definition

Use “k” nearest neighbors of the data point in question and classify them to the respective class to which maximum of these neighbors belong to.

1. K-nn is a **non-parametric** and **lazy-learning** algorithm that is used to predict classification, based on **feature similarity**.
2. There is **no explicit training phase**, since it does not use the training data to do any generalization, like most of the other classification algorithms.
3. For the same reason, it **requires most** of the training data when it needs to actually predict the class/label during the testing phase.

Algorithm

```
1. Let $k$ be the number of nearest neighbors and $D$ be the set of
   training examples.
2. for each test example z=(x',y')do
3.     Compute d(x',x), distance between z and every example,
    (x,y) ε D.
4.     Select Dz ⊆ D, set of k closest training examples to z.
5.     $y' = argmax\ v \sum_{(xi,\, yi) \in Dz} I(v = yi)$
6. end for
```

Implementation

K - It is important to choose the value of k carefully, because if k is too small it can make the prediction sensitive to noise points. While, if k is too large it can include irrelevant points from other classes that may lead to an erroneous classification eventually. For our implementation we’re taking it as a user input.

Distance - We're using Euclidean distance as a metric for calculating the distances between the test sample point and the k neighbors around it. We've implemented this in form of a function `distance(a,b)` that returns the distance calculated between the supplied 2 parameters `a` and `b`.

Categorical data handling is achieved by having the data first stored in list of lists structure and then casting it to numpy.string type. This allows us the flexibility to use some of the numpy functions as well as not affecting the nature of these feature values.

Continuous data handling is achieved similarly by first storing the data as a list of lists and then casting and extracting it to numpy array. Given that the continuous data have numerical values, numpy is extremely useful and adept at handling these kind of data and we make full use of the package's capabilities to our benefit of data handling.

Results Analysis

1. K-nn provides us with better results than Naive Bayes for our datasets.
2. However, it is a lazy learner and hence much slower than Naive Bayes especially if the datasets is big.
3. With the value of k increasing, like k=100 we observed that it started wrongly classify irrelevant data too. Which is a known thing about this technique.

Pros

1. **Simple** algorithm. Easy to interpret the output of classification
2. The running time is **fairly low** compared to many other classification algorithms like CART-based Decision Tree, Random Forest and Logistic Regression.
3. K-nn can be useful in **both** the predictive problem cases - Classification and Regression but majorly used for Classification in the industry.
4. **Insensitive** to outliers in general.

Cons

1. K-nn algorithm requires attribute scaling to avoid any overfitting towards one of the attributes.

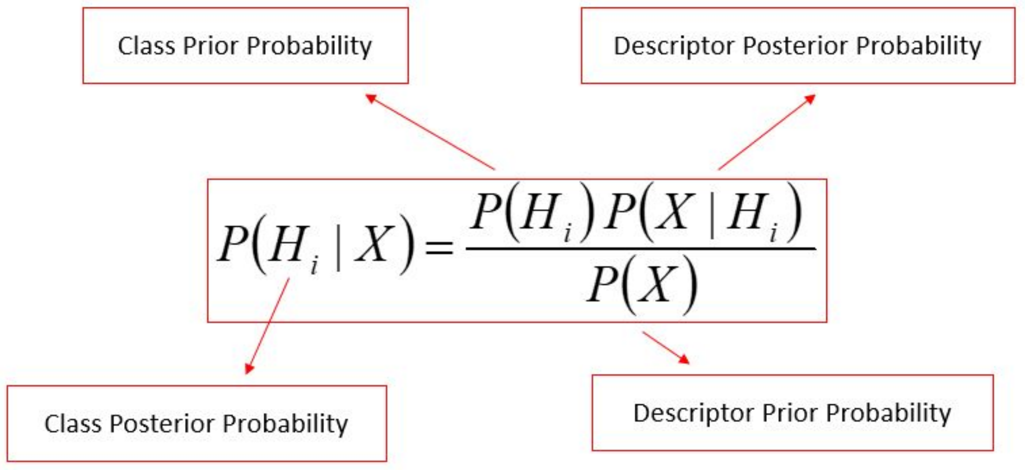
2. Prediction power can be **lower** than other highly-used classification algorithms - CART and Random Forest.
3. Storing of all the training data can cause a bottleneck leading to this algorithm being **computationally expensive** and with a **higher memory** requirement.
4. For really big datasets the running time for **predicting is slow** with better supervised learning algorithm running much faster in comparison.
5. Noise (k is small) and irrelevant features (k is large) can **affect the accuracy** of prediction made for classification.

Naive Bayes Algorithm and Implementation

1. In many applications the relationship between the attribute set and the class variable is non-deterministic because of noisy data or other certain confounding factors.
2. For such cases we resort to modeling probabilistic relationships between the attribute set and the class variable - based upon

Bayes Theorem and Classifier

A statistical principle for combining prior knowledge of the classes with the new evidence gathered from data. The basic equation can be written as -


Class Prior Probability
Descriptor Posterior Probability
$P(H_i \mid X) = \frac{P(H_i)P(X \mid H_i)}{P(X)}$
Class Posterior Probability
Descriptor Prior Probability

Algorithm

```
1. Let T = (X1, X2,... , Xd) denote a total order of the
   variables.
2. for j=1 to d do
3.    Let Xt(j) denote the jth highest order variable in T.
4.    Let π(Xt(j)) = {XT(1), XT(2), … , XT(j-1)} denote the
   set of variables preceding XT(j).
5.    Remove the variables from π(Xt(j)) that do not affect Xj
   (using prior knowledge).
6.    Create an arc between XT(j) and the remaining variables
   in π(Xt(j)).
7. end for
```

Implementation

Continuous data handling is achieved similarly by first storing the data as a list of lists and then casting and extracting it to numpy array. Given that the continuous data have numerical values, numpy is extremely useful and adept at handling these kind of data. We're using Probability Distribution Function to calculate posterior probabilities.

Zero-probability - Posterior probability goes to 0 if any of probability is 0. To correct this we've a check in place that corrects this problem by adding 1 to each case - this is also known as **Laplacian correction** (or Laplacian estimator)

Results Analysis

Naive Bayes in comparison to K-nn gives us a better accuracy and precision for a mixture of continuous and categorical data as seen in dataset 2 results. Although, for continuous only data K-nn did marginally better in terms of accuracy and precision as seen in dataset 1. Decision Tree and Random Forests consistently performed better than Naive Bayes in classification for given datasets.

Results for toy dataset

For the toy dataset we get the following probability values for the test case:

```
'sunny cool high weak 1'
p(H0|X) = 0.000624739691795        p(H1|X) = 0.00281132861308
```

Pros

1. It is fairly **easy to implement** Naive Bayes classifier.
2. The technique requires a **small amount of training data** to estimate the parameters.
3. The implementation is **highly scalable** and scales linearly with the number of predictors and data points.
4. The classifier works for both the binary and multiclass features.

Cons

1. A very strong assumption on the data distribution i.e. **features** being completely **independent** given their label class can lead to less accuracy. Hence "naive" classifier.

2. Continuous features need a special handling, usually **Gaussian based distribution functions** works well to adapt the classifier to continuous data. Though techniques like binning can them discrete but also lose useful information if not careful.

Decision Tree Algorithm and Implementation

1. Questioning the attributes carefully can conclude in predicting the class label of a test record. This question and answers can be arranged in a hierarchical structure consisting of nodes and directed edges.
2. A decision tree has 3 types of nodes -
    a. Root - has 0 incoming branch, 0 or more outgoing branch(es).
    b. Internal - has 1 incoming branch, 2 or more outgoing branches.
    c. Leaf (terminal) - has 1 incoming branch, 0 outgoing branch. Assigned a class label indicating a classification has been reached.
3. Exponentially many decision trees are possible from a given set of attributes. For computational feasibility we follow greedy strategy in algorithms - by asking locally optimum decisions about which attribute to use for partitioning the data.
4. Hunt’s algorithm forms the basis of many existing decision tree induction algorithms, including ID3, C4.5 and CART. The basic principle of growing a decision tree is by recursively partitioning the training records into purer subsets that form leaf/terminal nodes.
5. Gini Index is calculated as a measure of the impurity on every node to help us better understand how much purer each subset of data is at that respective node. A value 0 indicating the purest subset and 0.5 indicating that records are equally distributed among all classes making it tough to classify them on that node.

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

6. Information Gain measures the impurity reduction which helps us choose the attribute to split the records on that node. The most reduction i.e. maximum gain is favored.

$$GAIN_{split} = Measure(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Measure(i) \right)$$

Algorithm

```
TreeGrowth (E, F)
 1. if stopping_cond(E, F) = true then
```

```
2.    leaf = createNode()
3.    Leaf.label = Classify(E)
4.    return leaf
5. else
6.    root =  createNode()
7.    root.test.cond = find_best_split(E, F)
8.    Let V = {v | v is a possible outcome of root.test_cond}
9.    for each v ε V do
10.        Ev = {e | root.test_cond(e)=v and e ε E}
11.        child = TreeGrowth(Ev, F)
12.        add child as descendent of root and label of the edge (root → child) as v
13.     end for
14.   end if
15.   return root.
```

Implementation

Categorical-feature - We’re using a list of lists implementation to hold the categorical feature values and use it ahead for processing and calculations too.

Continuous-feature -We’re extracting the continuous valued feature data from the list of lists and converting it to a numpy array, since they are numerical it helps to use numpy array for processing and calculating the implementation metrics and variables.

Best-feature - With the help of Gini index and Information Gain we get to choose attribute that gets us larger and purer subsets that help us decide the best question to ask locally and split the records there accordingly.

Post-process - Tree pruning is a post-process operation done after inducing the full-grown tree. Pruning is useful to restrict the overfitting of the model.

Results Analysis

1. Decision tree is an excellent approach for classifying data, as is evident when we are using our toy dataset.
2. But, often times when we do not provide a proper stopping criteria, it is prone to overfitting which provides high accuracy on train set, essentially memorizing the data and provides poor generalization. This is evident on our

Pros

1. They are **inexpensive to construct** and **extremely fast** at classifying unknown records.
2. Decision trees when at a smaller-depth size are **very easy and quick to interpret.**
3. Despite being an old algorithm in general, the performance in terms of **accuracy** of classification is still **at par** with many modern techniques.
4. The decision **tree induction is a non-parametric** approach for building classification models meaning **not requiring any assumptions** regarding the type of probability distributions satisfied by the class and other attributes.
5. They are **expressive** representation for learning **discrete-valued functions**.
6. They are quite **robust to the presence of noise**, especially when with methods that avoids overfitting.
7. They do **not get affected** by presence of **redundant attributes** too.

Cons

1. **Overfitting** is a real issue here, with too much of training data reducing training error but increase in testing error after a threshold.
2. **Not expressive enough** for modelling continuous variables especially when test condition is involving only a single attribute at-a-time or a Boolean problem like parity function.
3. This algorithm makes decision trees susceptible to **high variance** if they are not pruned.
4. Due to top-down approach, the number of records at leaf node may be too small to make a statistically significant decision about the class representation of the node, also known as **data fragmentation.** The solution is setting a maximum depth limit.
5. **Subtree replication** can cause the decision tree to become more complex than necessary also adding to difficulty in interpretation.
6. The boundaries are **rectilinear**.

AdaBoosting Algorithm and Implementation

Algorithm

```
1. W = {wj = 1/n | j=1,2,...,N} (initialize weights)
2. Let k be the number of boosting rounds.
3. For i-1 to k do
4.    Create training set Di by sampling(with replacement)
   from D according to w.
5.    Train a based on classifier Ci on Di.
6.    Apply Ci to all examples in the original training set D.
7.    $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \; \delta(C_i(x_j) \neq y_j)\right]$  {weighted error}
8.    If εi > 0.5 then
9.       w = {wj = 1/N | j=1,2,...,N} {reset the weights}
10.       Go back to Step 4.
11.     End if
12.    $\alpha_i = \frac{1}{2}\ln\frac{1-\epsilon_i}{\epsilon_i}$
13.     Update the weight of each example
14.   End for
15.   $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$
```

Pros

1. Powerful algorithm as it is. Minimum adjustments required to be mentioned for running this classifier in comparison to others like SVM.
2. The algorithm adaptively chooses the weak classifier from GRT that works best at that round of boosting.

Cons

1. Can be sensitive to noisy data and outliers.
2. Less susceptible to the overfitting problem than most learning algorithms.
3. Does not currently support null rejection.

Random Forests Algorithms and Implementation

1. The technique is an ensemble learning method for decision tree classifiers.
2. The basic concept is to construct multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction depending on the nature of the data being worked on.
3. They are also a primary way to correct the decision tree's habit of overfitting the generated model to trained data.
4. They differ from AdaBoost, where random vectors used to generate trees are from a fixed probability distribution, and not adaptive approach.
5. Bagging using decision trees is a special case of Random Forests, where randomness is injected into the model-building process by randomly choosing N samples with replacements from the original training dataset.

Algorithm

```
1. Choose T-number of trees to grow.
2. Choose m<M (M is the number of total features) -number of
   features used to calculate the best split at each node
   (typically 20%)
3. For each tree
     a. Choose a training set by choosing N times (N is the
        number of training examples) with replacement from the
        training set.
     b. For each node, randomly choose m features and
        calculate the best splot.
     c. Fully grown and not pruned.
4. Use majority voting among all the trees.
```

Implementation

The implementation of random forest is similar like decision tree. It is an extension of bagging and building trees based on multiple samples of your training data, it also constrains the features that can be used to build the trees, forcing trees to be different. This, in turn, can give a lift in performance.

Decision trees involve the greedy selection of the best split point from the dataset at each step and therefore, susceptible to high variance if they are not pruned. This high variance can be reduced by creating multiple trees with different samples of the training dataset and combining their predictions. This approach is called bootstrap aggregation or bagging for short.

Like bagging, multiple samples of the training dataset are taken and a different tree trained on each of dataset. The difference is that at each point a split is made in the data and added to the tree, only a fixed subset of attributes/features can be considered.

The result of creating separate trees trained on different samples is that the trees that are more diverse will have different predictions and a combined prediction that often has better performance that single tree or bagging alone.

Results Analysis

1. Random forest gives better result for the 10 fold cross validation as compared to Decision tree, since it creates multiple trees for different set of training data. This helped in combining precisions of different trees which have high variance.
2. Classification of data becomes faster due to reduced tree size and iterative training of different trees on different data sets.

Pros

1. Currently one of the most accurate learning algorithms available and the classifier given a highly accurate results and estimations on what variables are important to classification.
2. Running efficiency on larger datasets is also commendable, since it can handle thousands of input variables without variable deletion.
3. Can effectively estimate missing data and maintains accuracy when a large proportion of the data are missing.
4. It can compute balancing error for unbalanced data sets that can further help in finding relation between the variables and classification with prototype computation.
5. All the above benefits can also be achieved for unlabeled data i.e. unsupervised clustering is possible along with outlier detection.

Cons

1. They are known to overfit some datasets with noisy classification.
2. A very large number of trees may actually make the algorithm slow for real-time prediction and can also cause memory limitations depending on the computing setup being used.
3. Can end up making very complex forests and operate more like a black box that are very hard to interpret.

Cross Validation

For all of our algorithms we are using 10 fold cross validation, on our training datasets to split it into training and validation sets. We are essentially randomizing and splitting our data into 10 parts, then putting aside one part for validation and using the rest for training on each of our 10 iterations. This allows us to get average metric values on the performance of our models. This is better, so that we get a more reliable average accuracy, precision, recall and F1 measure from our models.

External References

1. A very simplified introduction to K-nn was referred to from Analytics Vidhya blog - https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/
2. Naive Bayes basics - https://www.youtube.com/watch?v=iA2nEXanP_o
3. Decision tree required understanding the various metrics involved and this blog was useful - https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python
4. Analytics Vidya has another good practical way to explaining the Decision Tree - https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/
5. Comparing the popular algorithms - http://www.cs.uvm.edu/~icdm/algorithms/10Algorithms-08.pdf
6. Google Developers YouTube channel has quick learning video about Decision Trees - https://www.youtube.com/watch?v=LDRbO9a6XPU&t=252s
7. Random Forests - https://en.wikipedia.org/wiki/Random_forest
8. Random Forests understanding better - https://amateurdatascientist.blogspot.com/2012/01/random-forest-algorithm.html
9. AdaBoost -http://www.nickgillian.com/wiki/pmwiki.php/GRT/AdaBoost#Advantages