# Methods for Multi-Class and Multi-Target Prediction

Kavish R. Munjal[1]

***Abstract*—Though many problems in the real world are yes/no problems, there are many cases where multiple class values of the target variable are possible. The binary classification algorithms can be transformed to predict multiple classes by two method: algorithm adaptation and multiple uses. The project focuses on the former adapting Naive Bayes, Decision Trees and Online Learning algorithms to predict multiple classes. The methods are applied on two datasets.**

**In a lot of cases, simultaneous prediction of multiple target variables is also necessary. Single-Target Prediction Models can be used for this in two ways: algorithm adaptation and multiple uses. The project focuses on the latter using independent use, forming new targets by random linear combination and forming layers of classifiers in two ways. The methods are applied on three datasets.**

**The report ends with a note on structured prediction.**

## I. INTRODUCTION

The problem of Binary Classification takes in input feature variables and outputs one of two classes. A lot of problems in nature are binary: Spam/not Spam, segmentation by a boundary, etc. Most problems like this deal with saying if a target variable is or is not in a class, effectively positive or negative. However, this makes the problem fairly limited. Multi-class Classification extends this problem to multiple classes, where the target variable can take more than two values. Problems like text classification, clustering, digit classification, etc are multi-class problems. This area deals with most classification problems. Given a training data set of the form (**x**, y), we aim to form a hypothesis $\mathbb{H}(\mathbf{x})$ such that $\mathbb{H}(x_i) = y_i \; \forall i$

Most classifiers defined in Machine Learning are primarily binary: Decision Trees, Naive Bayes, Perceptron, etc. However, this does not mean they cannot be used for multi-class classification. Classifiers can be adapted to multiple classes. The project extends the Naive Bayes classifier, Decision Trees and Online Learning (Perception and Winnow) for multiple classes.

Real world prediction problems often involve the simultaneous prediction of multiple target variables using the same set of predictive variables. For example, annotation of images and video (news clips, movies clips), functional genomics (gene and protein function), music categorization into emotions, text classification (news articles, web pages, patents, e-mails, bookmarks,y), directed marketing and others. When the target variables are discrete, the prediction task is called multi-label classification while when the target variables are real-valued the task is called multi-target regression.

Just like binary classifiers can be used for multi-class classifiers, single target prediction algorithms can be used for predicting multiple targets. The naive way to do this would be to use the single target predictors independently to predict multiple targets one by one. However, any correlations between the targets is lost in this way. Algorithm adaptation is one options that has been looked at in the industry for algorithms like Decision Trees, Neural Networks, etc. However, this project focuses on using independent predictors of multiple targets and combining their predictions in ways that can exploit any possible correlations between target variables.

Section II describes the algorithms I used. Section III describes the datasets these algorithms were used on. Section IV gives a quick explanation of the experimental setup while Section V give the results.

## II. THE ALGORITHMS

### A. *Multi-Class Classification*

Problem Setting: Given various pairs of data points and targets: $(x_i, y_i)$, where $x_i \in \mathbb{R}^n, y_i \in \mathbb{C}$ train a machine learning model and create a hypothesis $\mathbb{H}$ such that given an unseen example, x, H(x) = y.

Here, n is the number of features and $\mathbb{C}$ is the set of classes

*1) **Naive Bayes**:* The Naive Bayes algorithm is a method of supervised learning using Bayes theorem, which says:

$$p(class|data) = \frac{p(data|class)p(class)}{p(data)}$$

$$p(class|data) \propto p(data|class)p(class)$$

Here, p(class|data) is the posterior probability, p(class) is the prior probability of the given class while p(data—class) is the likelihood of the data, given the

[1] `kmunjal2-at-illinois.edu`

class. The data is in the form of feature variables with different values in different classes. In the end, the class with the highest posterior probability is chosen. This is Bayes optimal classifier, which though accurate, is also very hard to compute. To make computations easier, we introduce the Naive Bayes assumption. The Naive Bayes assumption says that feature values are independent of each other given the target variable.

Thus, p(d|c) = p($f_1$|c) p($f_2$|c) .... p($f_n$|c), where $f_i$ are feature values adn c is the class.

The Naive Bayes algorithm is fast and efficient, and despite having an assumption that is not true in most cases, gives satisfactory results. Moreover, we can see that this formula is not limited to two classes and can directly be used for multiple classes making it naturally extensible to multiple classes. After calculating the likelihoods and the priors for each feature and class respectively in the training phase, we can use them to classify new instances.

*2) **Decision Trees**:* The decision tree algorithm is a widely known method of machine learning which uses a tree-like model of the data to classify new instances. The idea is to start at the root node, which represents the entire dataset. Then, at each level, we break the nodes into their children, thereby separating the data into two or more parts, each ”purer” than the parent. Here the notion of purity of a node is defined by the number of different classes in the node: the more the number of different classes in the node, the lower the purity (for example, the root is extremely impure). This is done until a node is formed which is pure, i.e, has only one type of class which is known as the leaf node. The splitting of a node into children is done by some feature, which is chosen based on an information gain.

In the binary classification case, the information gain is defined using entropy for data S.

$$\mathtt{Entropy(S)} = -\mathtt{p_{-}log(-p_{-})} - \mathtt{p_{+}log(-p_{+})}$$

where $p_{-}$ is the fraction of negative examples, while $p_{+}$ is the fraction of positive examples. Higher entropy implies higher impurity of node
Information Gain for attribute, a on dataset S, is:

$$\mathtt{I_G(S,a)} = \mathtt{Entropy(S)} - \sum_{\mathtt{v \in values(a)}} \frac{|\mathtt{S_v}|}{\mathtt{S}} \mathtt{Entropy(S_v)}$$

where $S_v$ is the subset of S for which attribute, a has value v

To extend this to multiple classes, we need to change the definition of entropy:

$$\mathtt{Entropy(S)} = -\sum_{\mathtt{c \in classes}} \mathtt{p_c log(-p_c)}$$

where c is the class to be predicted while $p_c$ is the fraction of examples of class c.

Prediction of a new data point involves going down the tree choosing different branches depending on the corresponding feature values until it reaches a leaf node where the prediction is simply the class of the leaf.

*3) **Online Learning**:* Two online learning algorithms that are in common use are: Perceptron and Winnow.

Online learning rules involve the following algorithm in an n-dimensional space: initialize weights and start iterations of prediction, updating if prediction was wrong and repeat until convergence

The difference between perceptron and winnow lies in the update rule for weights: while perceptron uses additive updates, winnow uses multiplicative updates. In both cases, the class label, y, is binary which can be clearly seen as the prediction is given as a sign function. To extend this to multiple classes, the predicting label is given by maintaining a weight vector for each class and taking the argmax over the dot product of the weight vector and the feature vector. If the prediction is correct, we leave things unchanged. If it is incorrect, we promote the weights of the correct class label and demote that of the incorrectly predicted class

Thus, the algorithm becomes:

1) Initialize $w_c = 0 \in \mathbb{R}^n \; \forall \; c \in classes$
2) Predict label: y' = $argmax_{c \in classes}(w_c \cdot$ x)
3) If prediction is wrong, promote weights, $w_y$ and demote weights $w_{y'}$, otherwise, leave weights unchanged
4) Back to step 2, until convergence

## *B. **Multi-Target Prediction***

Problem Setting: Given various pairs of data points and targets: ($x_i$, $y_i$), where $x_i \in \mathbb{R}^n$,$y_i \in \mathbb{C}^c$ train a machine learning model and create a hypothesis $\mathbb{H}$ such that given an unseen example, x, H(x) = y.

Here, n is the number of features and $\mathbb{C}$ is the set of classes and c is the number of target variables

In all the algorithms below, we use single-target predictors. To test accuracies, I have experimented with multiple algorithms, which will be seen in the results.

*1) **Independently using Single-Target Predictors**:* Various single-target algorithms exist for predicting individual target variables, some described above. The simplest way to predict the multiple variables is to use these algorithms to predict each target individually.

Though this method can prove to be fairly accurate, it completely ignores any possible correlations between the targets.

*2) **Random Linear Target Combination**:* This is a way to combine target variables in a way to produce new target variables via random linear combinations.

Consider a vector, y of p target variables. y $\in \mathbb{R}^p$. I define a randomly-generated matrix, A of dimension (r x p) where r >> p. If we then multiply the matrix, A with the vector, y we get a new vector, z = Ay such that z $\in \mathbb{R}^r$. This way, we have created a new vector of target variables which are a combinations of the original target variables.

To get our original vector back, if A is a square matrix (that is, of dimension p x p), we can take the inverse a A to get y = $A^{-1}$z. If A is not a square matrix, which is the case here since we require r >> q, we can use the following operation: y = $int((A^TA)^{-1}A^Tz)$. The only requirement for multi-label classification is that the classes need to be in the form of integers, which can be done using a simple mapping for each class to an integer.

In the training phase, we take the pairs ($x_i$, $y_i$) where $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^p$ and create the vectors $z_i \in \mathbb{R}^r$ by using a randomly generated matrix A. Then we train the model for ($x_i$, $z_i$). For a new data point, we predict the variable $z_i'$, which we then convert to $y_i' = (A^TA)^{-1}A^Tz_i'$ to get the prediction for the target variables.

This method is not limited to multi-label classification, but can be used for multi-target regression as well. In fact, this method is expected to work better for regression problems since we don't have to round the result to an integer class.

*3) **Two-Layered Classifiers**:* This is a two-layered prediction algorithm.

We start with pairs ($x_i$, $y_i$) where $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^p$. In the training phase, we use independent single-target prediction models to predict the target variables and get predictions $y_i'$. However, instead of directly outputting these prediction, we do a second training phase by using these predicted values as additional input features and create a new learning model. The inputs for this, second learning model will be of the form ($x_i'$, $y_i$), where $x_i \in \mathbb{R}^{n+p}$ This way, any correlation between classes can be fully exploited since they are being used to predict each other.

For the testing phase, we take the data point and predict the targets. Then we combine these predictions and the original features and input them to the new model. Finally, we get the outputs.

*4) **Chained Classifiers**:* This is a multi-layered prediction algorithm and similar to the above, two-layered classifiers.

We start with pairs ($x_i$, $y_i$) where $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^p$. In the training phase, we start by predicting the first target variable. Then, we use this prediction and make it part of the input features and use the n+1 features to predict the second target. And so on. This way, we form a chain of classifiers predicting one target variable at a time and making it part of the input features and forming a new model. The input to the jth model is ($x_i'$, $y_i$), where $x_i \in \mathbb{R}^{n+j}$, j $\in [1, p-1]$

One problem with this classifier is that it is highly dependent on the order we form the chains, i.e, the order in which we choose the target variables to be predicted. One way to counter this would be to build multiple chains with different, random ordering and combine the results, thus forming an ensemble of chained classifiers.

## III. DATASETS

### *A. Digit Classification*

Input is in the form of a 28x28 array, where each spot in the array is filled by either a symbol or a blank space which together form a digit. An example of such a digit is given below:

```
      +++#######+
    ++##########+
    +############+
    +###+++++####+
     +++     ###+
            +###+
           ++###
         +######
    ++########++
   +#########+
   +##########+
    ++++++++##+
            ##+
           +##+
  +       +###+
++#+     +####+
###+++++####++
###########+
++#######++
  +##+++
```

The feature vectors extracted from this data contain 784 elements each representing a position in the above array. A value of 1 means there is a symbol at the position while a value of zero means there is a black space.

The label is simply given by the digit the figure represents. The label for the above data point is: 3

This data was used for the multi-class prediction algorithms

Total number of training points: 5000

Total number of testing points: 1000

### *B. Letter Recognition*

Input is in directly the form of vectors containing 17 elements where the first one is the letter category (using images based on 20 different fonts) and the other 16 are numeric attributes (statistical moments and edge counts)

For example: X,3,9,5,7,4,8,7,3,8,5,6,8,2,8,6,7

This data was used for the multi-class prediction algorithms by making the letter (first attribute) the target and for multi-target prediction algorithms, by making the last 4 attributes the targets (in which case the letter is mapped to it's corresponding integer).

Total number of training points: 15000

Total number of testing points: 5000

### C. Emotions

The data represents human emotions after listening to music. The feature vector contains various numeric attributes to quantify listening to music, while the target vector contains emotions: Amazed-Suprised, Happy-pleased, relaxing-clam, quiet-still, sad-lonely, angry-aggresive. Since humans can feel multiple emotions, this is clearly a multi-target learning problem

Total number of training points: 500

Total number of testing points: 93

### D. Yeast

The data represents yeast functional genomics. The feature vector contains various numeric attributes while there are 14 binary target variables each representing a class of yeast. A yeast can belong to multiple classes, making this a multi-target problem.

Total number of training points: 2000

Total number of testing points: 417

## IV. EXPERIMENTAL SETUP

I programmed multi-class algorithms for Naive Bayes, Perceptron and Winnow using C++ myself. For the multi-class decision tree algorithm, I used the data mining software, Weka made by the Machine Learning Group at the University of Waikato.

The Naive Bayes algorithm included the use of Laplace smoothing. The Decision Tree algorithm was tried with various depths of the tree and the best value is given below, which was achieved for depth 5. The perceptron and winnow algorithms are set such that they cycle through the training examples until weight vectors are achieved such that the error on the training dataset is zero. The perceptron learning rate was set such that it gets lower as we see more and more examples. The winnow learning rate was constant.

For the multi-target algorithms, I used scikit-learn: the machine learning package for python and wrote a script in python for each of the methods that I implemented. The Random Linear Combinations algorithm was tested with various number of targets for multiple iterations. To create an ensemble of chained classifiers, randomized order of labels was chosen for prediction and the code was also run for multiple iterations. In all cases, as my single target algorithm, I used a Decision Tree Classifier.

The code is available upon request

## V. RUNNING THE ALGORITHMS

### A. *Multi-Class Algorithms*

Table 1 shows the comparison comparison between the accuracies of the various algorithms for the digit classification dataset and the letter classification dataset.

Some key things to note:

- Perceptron took 85 iterations on all examples for the digit classification training dataset to have zero error
- Winnow went through the maximum of 200 iterations on all examples for the digit classification training dataset without having zero error (final error was 0.08%)
- Full length decision tree was 6 deep on the digit classification dataset, but a stump of depth 5 had a higher accuracy.

TABLE I
COMPARING ACCURACIES OF MULTI-CLASS ALGORITHMS

| Accuracies | | |
|---|---|---|
| Datasets –> Algorithm | Digits Classification | Letter Classification |
| Decision Trees | 64% | 86.6 |
| Naive Bayes | 77.2% | 63.3 |
| Perceptron | 80.5% | 97.24 |
| Winnow | 79.4% | 95.11 |

### B. *Multi-Target Algorithms*

The Random Linear Combinations algorithm was run for 5 iterations for a 1000 new target variables. Table III shows a comparison of various number of target variables, just for comparison. The Chained Classifier algorithm was run for 100 iterations, thus giving an ensemble of 100 chained classifiers.

*1) Letter Recognition:* Table II shows the accuracies of predicting the targets using the four algorithms.

*2) Yeast Classification:* Table IV shows the accuracies of predicting the targets using the four algorithms.

*3) Emotion Classification:* Table V shows the accuracies of predicting the targets using the four algorithms.

TABLE II
COMPARING ACCURACIES OF MULTI-TARGET ALGORITHMS

| Accuracies for Letter Classification | | | | |
|---|---|---|---|---|
| Alg. –> | Independent | Combinations | Layered | Chains |
| Target1 | 67.38% | 70.92% | 70.06% | 67.38% |
| Target2 | 65.60% | 71.02% | 69.40% | 67.38% |
| Target3 | 65.72% | 68.72% | 67.00% | 67.38% |
| Target4 | 59.40% | 66.28% | 63.20% | 67.38% |
| Average | 64.525% | 69.24% | 67.42% | 67.38% |

TABLE III
COMPARING ACCURACIES OF DIFFERENT NUMBER OF TARGETS

| Accuracies for Letter Recognition | | | | |
|---|---|---|---|---|
| No. of New Targets –> | 4 | 10 | 100 | 1000 |
| Target1 | 36.90% | 56.38% | 68.16% | 70.92% |
| Target2 | 43.88% | 55.48% | 68.20% | 71.02% |
| Target3 | 47.26% | 53.84% | 66.80% | 68.72% |
| Target4 | 35.82% | 52.55% | 64.30% | 66.28% |
| Average | 40.96% | 54.56% | 66.86% | 69.24% |

TABLE IV
COMPARING ACCURACIES OF MULTI-TARGET ALGORITHMS

| Accuracies for Yeast Classification | | | | |
|---|---|---|---|---|
| Alg. –> | Independent | Combinations | Layered | Chains |
| Target1 | 68.03% | 77.55% | 86.39% | 71.48% |
| Target2 | 62.58% | 65.30% | 80.95% | 74.10% |
| Target3 | 59.86% | 76.87% | 95.24% | 70.34% |
| Target4 | 65.99% | 72.79% | 89.80% | 71.55% |
| Target5 | 68.03% | 80.95% | 95.24% | 72.36% |
| Target6 | 64.63% | 81.63% | 96.60% | 69.95% |
| Target7 | 75.57% | 83.67% | 97.28% | 73.73% |
| Target8 | 75.51% | 82.31% | 97.28% | 72.97% |
| Target9 | 87.07% | 92.51% | 95.23% | 70.10% |
| Target10 | 80.27% | 89.12% | 94.55% | 69.60% |
| Target11 | 76.87% | 87.75% | 94.55% | 71.05% |
| Target12 | 61.90% | 72.10% | 100 % | 69.33% |
| Target13 | 59.86% | 69.38% | 99.32% | 71.04% |
| Target14 | 97.96% | 98.64% | 98.64% | 67.01% |
| Average | 71.72% | 80.76% | 94.36% | 71.04% |

TABLE V
COMPARING ACCURACIES OF MULTI-TARGET ALGORITHMS

| Accuracies for Emotion Classification | | | | |
|---|---|---|---|---|
| Alg. –> | Independent | Combinations | Layered | Chains |
| Target1 | 69.89% | 81.72% | 75.27% | 72.24% |
| Target2 | 58.06% | 78.49% | 76.34% | 71.75% |
| Target3 | 62.36% | 80.65% | 83.87% | 72.82% |
| Target4 | 84.94% | 89.24% | 84.95% | 72.10% |
| Target5 | 72.04% | 77.42% | 77.41% | 73.96% |
| Target6 | 76.34% | 86.02% | 86.02% | 72.38% |
| Average | 70.60% | 82.26% | 80.65% | 72.54% |

## VI. RESULTS

Among the various multi-class algorithms, the perceptron algorithm has the highest accuracy. However, both Perceptron and Winnow are computationally very expensive since the data is cycled through many times. If computational power is to be taken into account, the Naive Bayes algorithm will emerge as the winner since it has high accuracy and it is efficient.

The Multi-target algorithms gave out interesting results. Firstly, independently using single-target predictors gives a satisfactory accuracy. However, the multi-target uses of single-target predictors was successful since each algorithm gave a higher accuracy in most cases, though sometimes not that much higher.

In case of the letter recognition and the emotions classification dataset, the highest accuracy is achieved by Random Linear Combinations, while in the yeast classification Dataset, the highest accuracy is achieved by the Layered Classifier. This shows that there is much more correlation between the targets of the latter dataset than the other two. This correlations was successfully exploited to give out better results.

An interesting things to notice about the Ensemble of Chained Classifiers is that they not just increase the overall accuracy, but also tends to smooth out the accuracies of the target variables. This is most prominent in the yeast classification, where accuracies ranging from 59 to 97 % in the independent case is turned to a range of 67 to 74 %. In this particular case, however, there was also a net decrease of accuracy.

Also, though it consistently increases accuracy, the random linear combinations algorithm is computationally expensive due to multiple iterations and increase of the number of target variables by multiple orders of magnitude. The Ensemble of Chained Classifiers also give computational inefficiency due to multiple iterations.

## VII. Conclusions and Future Work

In conclusion, the algorithms described able for multi-target prediction work better than using independent single-target predictors.

Using algorithm adaptation methods for multi-class classification has very high accuracy and is computationally comparable with the binary classification algorithms. In case of multiple targets, I did not test any algorithm adaptation methods. Using single target classifiers in different ways to exploit possible correlations between targets has high accuracy but is computationally inefficient. Except the layered classifier, the other classifiers use multiple iterations, where each iteration uses a single target classifier multiple time. This can use a lot of computational power.

The single target classifier used was a Decision Tree Classifier. We could also try other methods, like Naive Bayes, k-Nearest Neighbors or Support Vector Machines and see how the results are affected.

Moreover, the datasets I chose did not have a significant correlation between targets. Better datasets might be able to tell better how the algorithms work.

***Structured Prediction**:* The vast majority of prediction algorithms, such as those described in the previous section, are built to solve prediction problems whose outputs are simple. Here, simple is intended to include binary classification, multiclass classification and regression. In contrast, the problems I am interested in solving are complex. The family of generic techniques for solving such complex problems are generally known as structured prediction algorithms. Structured Prediction deals with predicted complex structures which can range from arrays of integers to images.

The inputs and output of such algorithms are structures. An example of a structured learning library is shown in http://cogcomp.cs.illinois.edu/page/software_view/Illinois-SL

In the future, I would like to explore this field more and apply to my data

## References

[1] G. Madjarov, D. Kocev, D. Gjorgjevikj, S. Dzeroski, An extensive experimental comparison of methods for multi-label learning, in Pattern Recognition 45, 2012, p30843104.

[2] Breiman L., Friedman J. H., Olshen R. A., Stone C. J, Classification and Regression Trees (Book style), 1984, Wadsworth Publishing Company.

[3] E. Spyromitros-Xioufis G. Tsoumakas, W. Groves, I. Vlahavas, Multi-Label Classification Methods for Multi-Target Regression.2014

[4] M. Aly, Survey on Multiclass Classification Methods, 2005

[5] M. Carrasco Kind, R.J. Brunner, TPZ : Photometric redshift PDFs and ancillary information by using prediction trees and random forests, 2013

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R. and Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., Scikit-learn: Machine Learning in Python, 2011 Journal of Machine Learning Research 12 p2825–2830.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[8] A. Elisseeff, J. Weston, A Kernel Method for Multi-Labelled Classification, 2001, In Advances in Neural Information Processing Systems 14, p681–687, MIT Press