Incremental Classification Learning by Using Feature Projections

Tolga Aydın, and Halil Altay Güvenir

Department of Computer Engineering
Bilkent University,
Ankara, 06800, Turkey

Abstract. Classification learning is an important research topic in machine learning and data mining disciplines. In our study, CUFP (Classification by Using Feature Projections), a feature projection-based incremental classification-learning algorithm, was developed and tested on real world data sets, giving promising results. The training phase of CUFP constructs points and determines the counts of the training instances of each class at each point in the case of nominal feature projections. For linear feature projections, gaussian probability density functions are constructed for each class. In the classification phase, each feature projection distributes its vote among possible classes. The vote vectors of features are used according to some vote evaluation types and the query instance's class is predicted.

1 Introduction

Classification learning is an important research topic in machine learning and data mining disciplines. Feature projection-based classification learning algorithms assume feature independence and somehow combine the classification results coming from each individual feature projection. The previous feature projection-based approaches presented in [1, 2, 4] constructed different models on feature projections and achieved successful results. These success stories motivated us to design CUFP (Classification by Using Feature Projections), a new feature projection-based technique.

The previous models constructed point (for nominal features) and range (for linear features) intervals to represent the training data. On a point interval, distribution of the training instances on this interval to classes is determined. On a range interval, the interval may keep the distribution of the training instances on this interval to classes as in the case of point intervals, leading this interval to be a heterogeneous one. However, it is also possible to construct homogeneous range intervals where each such interval consists of some training instances of the same class. If the range intervals are of type homogeneous, it is highly probable that most of those intervals overlap. The work presented in [4] develops overlapping homogeneous range intervals to represent the training data. The heterogeneous range intervals never overlap, but they may have a border between each other. The works presented in [1] and [2] construct heterogeneous

range intervals that do and do not have borders between consecutive intervals, respectively.

The CUFP approach differs from the existing ones by constructing gaussian probability density functions for each class, rather than constructing range intervals, on each linear feature projection. Although the probability of the training data to exhibit a gaussian distribution is low, we make such an assumption and obtain reasonable experimental results.

In the literature, there are incremental but not feature projections based classification-learning algorithms [5, 6]. In the case of feature projections based approaches, incremental learning is possible only in the models using homogeneous range intervals. In CUFP, on any linear feature projection the probability density functions of classes range from $-\infty$ to $+\infty$. Therefore, CUFP can be thought of using always overlapping homogeneous range intervals whose beginning values are all $-\infty$ and ending values are all $-\infty$. The number of such range intervals is obviously the number of classes. CUFP learns the concept description incrementally.

In feature projection-based approaches, each feature projection presents a vote vector holding the vote values given to classes when predicting the class of a query instance. The sum of the vote values is always 1 and the content of the vote vector is highly related to where the query instance falls into on each feature projection. When computing the final vote for some class c, the votes given to class c by all of the features are summed up. In CUFP, we not only use this technique but also employ the below three new techniques while computing the final votes.

a) Select highest "class c" vote between feature projections to be the final vote for class c.
b) Select median "class c" vote between feature projections to be the final vote for class c.
c) Use the number of feature projections that distributed the highest vote to class c as the final vote for class c.

These new techniques were inspired from the work presented in [3]. In their study, the authors state some weak points of Naïve Bayesian Classifier, make some simplifying assumptions to get rid of these weaknesses and obtain similar techniques to 'a', 'b' and 'c'. However, they make use of the posterior probabilities of the classes rather than the votes distributed among the classes as in the case of CUFP. In [3], authors work on combining predictions of different classifiers where each classifier represents a given pattern by a distinct measurement vector. On the other hand, in CUFP, each feature projection is treated as a unique classifier representing a given pattern by the projected value on the associated feature.

The organization of the paper is as follows: Section 2 describes the training phase of CUFP. Section 3 is devoted to the classification in CUFP. Section 4 gives empirical results based on real world data sets and we conclude.

2 Training in CUFP

The training phase of CUFP, given in Figure 1, is achieved incrementally. On a nominal feature, concept description (model) is trained as the set of points and the counts of the training instances of each class at each point. On the other hand, on a linear feature, concept description is trained as the normal (gaussian) probability density functions for all classes.

In the training phase, if a newly added training instance t has a known value for a feature f (t_f is not missing), the model is updated incrementally for the projection on f as follows:

For a nominal feature f, *find_point* (f, t_f) procedure tries to find t_f in the current concept description belonging to f. If t_f is found at a point p, then *point_train_data_count* [f, p, s] is incremented, assuming that the training instance is of class s. If t_f is not found, then a new point p' is constructed and *point_train_data_count* [f, p', *class*] is initialized to 1 for *class* = s, and to 0 for *class* = *others*.

For a linear feature f, if a training instance t of class s is examined, we let the previous training instances that are of class s and that have known f values to construct a set P and let $\mu_{f,s}$ and $\sigma_{f,s}$ to be the mean and the standard deviation of the f values of these instances in P, respectively. Then, $\mu_{f,s}$ and $\sigma_{f,s}$ are updated incrementally and the gaussian probability density function for class s on feature f is redetermined. Updating $\sigma_{f,s}$ incrementally requires $\mu^2_{f,s}$ to be updated incrementally, as well. If t is the first training instance of class s on a feature projection f, $\sigma_{f,s}$ and the probability density function for class s on feature f become undefined. The density function may also become undefined if $\sigma_{f,s}$ is zero.

Training can better be explained by looking at the sample data set in Figure 2. The data set consists of ten training instances and one query instance. It includes one nominal (f_1) and one linear (f_2) feature. Nominal feature takes two values 'A' and 'B', whereas linear feature takes some integer values. Furthermore, there are two possible classes: 'c_1' and 'c_2'. The linear feature is assumed to have gaussian probability density functions for both classes. The data set does not include any missing feature values. In Figure 3, concept description learned from ten training instances on two features are given.

CUFP$_{train}$ (t) /* t: newly added training instance */

begin

let s be the class of t

let *others* be the remaining classes

if *Training Data* = {t}

for each feature f and class c

train_data_count[f][c] = 0

for each feature f

if t_f is not missing

train_data_count[f][s]++

for each feature f

if f is nominal and t_f is not missing

p = *find_point*(f,t_f)

if such a p exists

point_train_data_count [f,p,s]++

else /* add new point for f */

add a new point p'

point_train_data_count [f,p',s] = 1

point_train_data_count [f,p',*others*] = 0

else if f is linear and t_f is not missing

if *train_data_count*[f][s] = 1

$\mu_{f,s} = t_f$, $\mu_{f,others} = 0$

$\mu^2_{f,s} = t_f^2$, $\mu^2_{f,others} = 0$

$\sigma_{f,s}$ = *Undefined*

norm_density_func.$_{f,s}$= *Undefined*

else

n = *train_data_count*[f][s]

$\mu_{f,s} = (\mu_{f,s} * (n-1) + t_f) / n$

$\mu^2_{f,s} = (\mu^2_{f,s} * (n-1) + t^2_f) / n$

$\sigma_{f,s} = \sqrt{\frac{n}{n-1}(\mu^2_{f,s} - (\mu_{f,s})^2)}$

if $\sigma_{f,s} = 0$

norm_density_func.$_{f,s}$= *Undefined*

else

norm_density_func.$_{f,s}$= $\frac{1}{\sigma_{f,s}\sqrt{2\pi}} e^{-\frac{(x-\mu_{f,s})^2}{2\sigma_{f,s}^2}}$

return { normal density functions for linear features; point trainining data counts for nominal features }

end.

Figure 1. Incremental train in CUFP

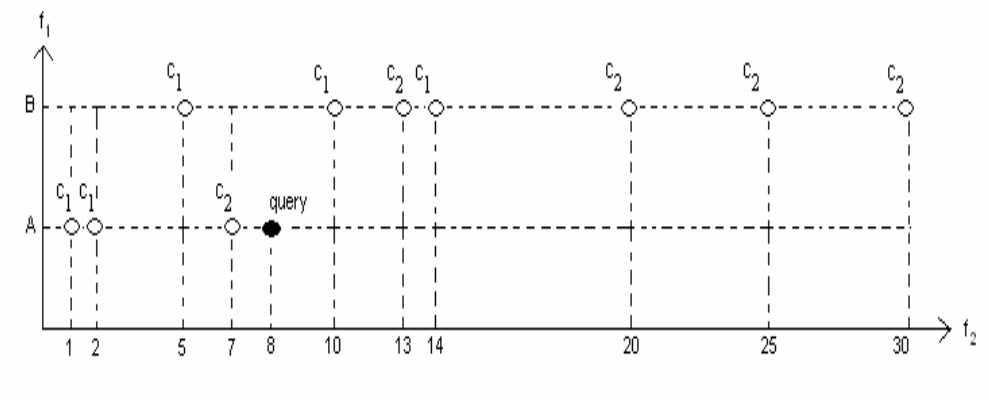

Figure 2. Sample data set

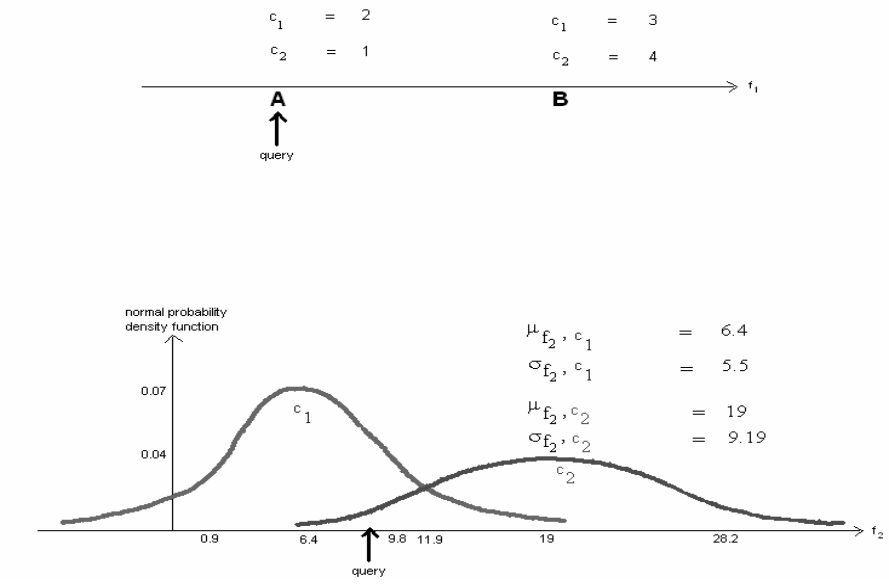

Figure 3. Concept description learned for the sample data set

3 Classification in CUFP

In the classification phase, each feature projection, independent of each other, issues a vote vector for the query instance q. Different vote evaluation types are used to obtain the

final votes for classes. The class with the highest vote is predicted to be the class of the query instance.

Classification phase of CUFP is shown in Figure 4. The query instance is projected on all features and each feature issues a vote vector. On all feature projections, the classification starts by giving a zero vote for each class. If q_f is missing, the classification process on f terminates, otherwise it proceeds according to the type of the features.

For a nominal feature f, *find_point* (f, q_f) procedure is used to search whether q_f exists in the f projection of the training instances. If q_f is found at a point p then for classes c such that there exists at least one training instance of class c and having a known value on feature f, this feature gives votes for class c as shown in the below equation and then these votes are normalized to ensure equal voting power among features. If q_f is not found, the classification process on f terminates.

$$feature\_vote\ [f, c] = \frac{point\_train\_data\_count\ [f, p, c]}{train\_data\_count\ [f][c]} \qquad (1)$$

It is important to note that '100 * *feature_vote* [f, c]' gives the percentage of the class c training instances with known f values that fall into the point p on feature projection f.

For a linear feature f, feature vote for class c is computed if $\sigma_{f,c}$ is defined and different than zero. If there exists at least one class c where $\sigma_{f,c}$ is defined and different than zero, the votes are again normalized to ensure equal voting power among features. The feature votes, if possible, are computed as given in equation 2.

$$feature\_vote\ [f, c] = \lim_{\Delta x \to 0} \int_{q_f}^{q_f + \Delta x} \frac{1}{\sigma_{f,c}\sqrt{2\pi}} e^{-\frac{(q_f - \mu_{f,c})^2}{2\sigma_{f,c}^2}}\, dx \qquad (2)$$

In the above equation, $\frac{1}{\sigma_{f,c}\sqrt{2\pi}} e^{-\frac{(q_f - \mu_{f,c})^2}{2\sigma_{f,c}^2}}$, gives the domain value of the gaussian probability density function of class c training instances on feature projection f for $x = q_f$. The equation, itself, gives the area between the x-axis and the probability density function between $x = q_f$ and $x = q_f + \Delta x$ when Δx goes to zero. It is apparent that '100 * *feature_vote* [f, c]' gives the percentage of the class c training instances with

known f values that fall into the point p on feature projection f. It is also apparent that feature votes are all zero. However, normalization process handles this seemingly problematic situation.

Upon each feature issues a vote vector, the final votes are determined according to the vote evaluation types. Figure 5 explains these types and the computation of the final votes briefly. Finally, if there exists exactly one class c that received the highest vote, that class is predicted to be the class of the query instance q. Otherwise, no prediction is made.

The querying phase can be better explained by using the sample data set and the corresponding model shown in Figure 2 and 3. The query instance is shown as <A, 8>.

$$feature\_vote[f_1, c_1] = \frac{2}{5} = 0{,}4$$

$$feature\_vote[f_1, c_2] = \frac{1}{5} = 0{,}2$$

$$normalized\_feature\_vote[f_1, c_1] = \frac{0{,}4}{0{,}4+0{,}2} = 0{,}67$$

$$normalized\_feature\_vote[f_1, c_2] = \frac{0{,}2}{0{,}4+0{,}2} = 0{,}33$$

$$feature\_vote[f_2, c_1] = \lim_{\Delta x \to 0} \int_{8}^{8+\Delta x} \frac{1}{5{,}5\sqrt{2\pi}} e^{-\frac{(8-6{,}4)^2}{2*5{,}5^2}} dx = \lim_{\Delta x \to 0} 0{,}07\Delta x$$

$$\text{feature}\_vote[f_2, c_2] = \lim_{\Delta x \to 0} \int_{8}^{8+\Delta x} \frac{1}{9{,}19\sqrt{2\pi}} e^{-\frac{(8-19)^2}{2*9{,}19^2}} dx = \lim_{\Delta x \to 0} 0{,}02\Delta x$$

$$normalized\_feature\_vote\ [f_2, c_1] = \lim_{\Delta x \to 0} \frac{0{,}07\Delta x}{0{,}07\Delta x + 0{,}02\Delta x} = 0{,}78$$

$$normalized\_feature\_vote\ [f_2, c_2] = \lim_{\Delta x \to 0} \frac{0{,}02\Delta x}{0{,}07\Delta x + 0{,}02\Delta x} = 0{,}22$$

Set of votes given to c_1 = {0,67, 0,78}
Set of votes given to c_2 = {0,33, 0,22}

If the vote evaluation type is chosen to be "*Sum Votes of Feature Projections*" :

$final\_vote[c_1] = 0.67 + 0.78 = 1.45$
$final\_vote[c_2] = 0.33 + 0.22 = 0.55$

Query instance is predicted to be of class c_1.

CUFP$_{query}$(*q*, *Vote_Eval_Type*) /* *q*: query instance*/
begin

for each feature *f*

for each class *c*
feature_vote[*f*,*c*] = 0

if *f* is nominal and q_f is not missing

p = *find_point*(*f*,q_f)
if such a *p* exists
for each class *c*
if (*train_data_count*[*f*][*c*] ≠ 0)
feature_vote[*f*,*c*] = $\frac{point\_train\_data\_count[f,p,c]}{train\_data\_count[f][c]}$
normalize_feature_votes(*f*)
/* such that $\sum_c feature\_vote[f,c] = 1$ */

else if *f* is linear and q_f is not missing

for each class *c* satisfying ($\sigma_{f,c}$ ≠ 0) and
($\sigma_{f,c}$ ≠ *Undefined*)

feature_vote[*f*,*c*] = $\lim_{\Delta x \to 0} = \int_{q_f}^{q_f+\Delta x} \frac{1}{\sigma_{f,c}\sqrt{2\pi}} e^{-\frac{(q_f-\mu_{f,c})^2}{2\sigma_{f,c}^2}} dx$

if ∃*c* such that ($\sigma_{f,c}$ ≠ 0) and ($\sigma_{f,c}$ ≠ *Undefined*)
normalize_feature_votes(*f*)

for each class *c*
Determine Final Vote(*Vote_Eval_Type*, *c*)

if there exists exactly one class *c* such that
final_vote[*c*] = $\max_{i=1}^{\#Classes} final\_vote[i]$
classify *q* as "class *c*"
else
do not classify *q*
end.

Figure 4. Classification in CUFP

Determine Final Vote (*Vote_Eval_Type, c*)

begin

for each class *c*

final_vote [*c*] = 0

if *Vote_Eval_Type* is "*Sum Votes of Feature Projections*"

$$final\_vote[c] = \sum_{f=1}^{\#Features} feature\_vote[f, c]$$

else if *Vote_Eval_Type* is "*Select Highest Vote Between Feature Projections*"

$$final\_vote[c] = \max_{f=1}^{\#Features} feature\_vote[f, c]$$

else if *Vote_Eval_Type* is "*Select Median Vote Between Feature Projections*"

$$final\_vote[c] = \underset{f=1}{\overset{\#Features}{median}}\ feature\_vote[f, c]$$

else if *Vote_Eval_Type* is "*Use Number of Feature Projections on Which Highest Vote is Obtained*"

for each feature *f*

if *feature_vote* [*f* , *c*] = $\max_{c=1}^{\#Classes} feature\_vote[f, c]$

final_vote [*c*]++

end.

Figure 5. Final Vote Determination in CUFP

4 Experimental Results

CUFP and the Naïve Bayesian Classifier were tested on eleven real world classification data sets, using leave-one-out cross validation. In Naïve Bayesian Classifier, the overall posterior probability is computed by multiplying the individual posterior probabilities. So, although Naïve Bayesian seems to be a bit better than the CUFP, there is always a risk of having a zero overall posterior probability just because of a single zero posterior probability on some feature *f*. In CUFP, feature vote concept is used instead of the posterior probabilities of classes. The elements of the set of votes given by the features to a class are never multiplied by each other. So, CUFP is a non-risky classifier.

Table 1. Accuracy of Classifications with Leave-One-Out Cross Validation

Data Set	Sum Votes of Feature Projections	Select Highest Vote Between Feature Projections	Select Median Vote Between Feature Projections	Use Number of Feature Projections on Which Highest Vote is Obtained	Naive Bayesian Classier
bcancerw	**96,14**	93,28	94,13	94,13	95,99
cleveland	82,84	69,64	82,18	82,45	**83,5**
diabetes	75	71,48	71,48	**75,61**	75,26
echocardio	71,62	75,68	75,68	75	**78,38**
horse	76,09	**82,34**	70,71	65	77,99
hungarian	84,35	75,51	81,29	83,39	**84,98**
iris	95,33	**96**	92	94,78	95,33
sonar	68,75	58,65	70,67	**70,94**	67,31
thyroid	97,21	92,09	98,14	**98,52**	96,74
vote	88,67	**93,33**	87,33	89,55	89
wine	94,38	90,45	90,45	86,39	**97,19**
Average Accuracy	84,58	81,68	83,1	83,25	**85,61**

5 Conclusion

In this work CUFP, a new feature projection-based, incremental classification-learning algorithm was developed giving promising experimental results. It differs from the existing feature projection based approaches by using gaussian probability density functions rather than range intervals for linear features. It also differs by using three new vote evaluation types in addition to the classical type of summing up the corresponding votes coming from the features for some class *c*.

References

1. Güvenir, H.A., and Demiröz, G., "**Classification by voting feature intervals**" *Proceedings of 9th European Conference on Machine Learning*, 1997, 85-92.
2. Güvenir, H.A., "**Benefit Maximization in Classification on Feature Projections**" *Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications* (AIA'03), Malaga, Spain (Sept. 8-10, 2003), 424-429.
3. Kittler, J., Hatef, M., Duin, R.P.W., and Matas, J., "**On Combining Classifiers**" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, 1998, 226-239.
4. Güvenir, H.A., and Koç, H.G., "**Concept Representation with Overlapping Feature Intervals**" Cybernetics and Systems, vol.29, 1998.
5. Mandziuk, J., and Shastri, L., "**Incremental class learning – an approach to longlife and scalable learning**" IEEE International Conference on Neural Networks, vol. 2, 1999.
6. Diehl, C.P., and Cauwenberghs, G., "**SVM Incremental Learning, Adaptation and Optimization**" International Joint Conference on Neural Networks, Portland OR, July 2003.