Classification Learning Properties

Notation and terminology: The number of features is $n \geq 1$. **R** is the set of real numbers. A **feature vector** is an element of $\mathbf{R}^n$. The set of classes is denoted **C**. A **training set** is a finite sequence of elements of $\mathbf{R}^n \mathrm{x} \mathbf{C}$.
A **learning algorithm** F is an algorithm that consumes a training set T and returns a function $F(T)\colon \mathbf{R}^n$ -> **C**. F(T) is the **classifier generated by** F **from** T.

General Properties for all Classification Schemes

<u>LearningTerminating</u>

The learning phase terminates on any training set.
I.e., given any finite training set T, F(T) eventually returns.
The following is an idea of how one might prove termination for the specific case of a decision tree learning algorithm: during the generation of the decision tree, the number of instances passed to the recursive function call is always decreasing.

<u>ClassifierTerminating</u>

All classifiers terminate on any input.
Specifically: for any training set T, and any feature vector **v**, F(T) will terminate on input **v** (and return a class in **C**).

<u>LearningDeterministic</u>

Two classifiers generated from the same training data will be identical.
This is another way of saying that F is a function: if given the same input T twice, it will produce the same output function F(T).

Applies to any deterministic approach; some approaches use random, so this would not apply.

<u>ClassifierDeterministic</u>

Each classifier is a deterministic function of its input.

I.e., for any training set T, F(T) is a function: if F(T) is given the same input **v** twice, it will return the same class both times.

Again, there are some approaches (for example, playing a game), where probabilistic techniques are used and this property would not apply.

<u>TrainingOrderOblivious</u>

The classifier generated from T is independent of the order of the elements of T.

I.e., F only depends on the multiset specified by T --- the set of elements with multiplicity.

May not be true in all cases (e.g. reinforcement learning)

<u>FeatureOrderOblivious</u>

The order of the features doesn't matter.

Let s be a permutation of {1,...,n}. For $\mathbf{v}=(v_1,...,v_n)$, define $s(\mathbf{v})=(v_{s(1)},...,v_{s(n)})$. For a training set T, define $s(T)=\{<s(\mathbf{v}),c> \mid <\mathbf{v},c> \text{ in } T\}$. The claim is that for any training set T and feature vector **v**,

$$F(s(T))(s(\mathbf{v})) = F(T)(\mathbf{v}).$$

I.e., if you permute all the feature vectors in a training set, and permute the test feature vector the same way, the resulting class is the same class you would get from the original classifier.

In a decision tree approach, if two features are identical, a different feature may be selected for splitting on, but the end classification result would be the same. This should hold for almost all approaches.

<u>ClassNameOblivious</u>

The names of the classes don't matter.

Let s:**C->C'** be a 1-1 correspondence. For a training set T over **C**, define s(T)={<**v**,s(c)> | <**v**,c> in T}, which is a training set over **C'**. The claim is that for any training set T and feature vector **v**,

$$F(s(T))(\mathbf{v}) = s(F(T)(\mathbf{v})).$$

This should really hold for every approach.

<u>ClassStable</u>

Any class returned by a classifier must occur in the training set used to generate that classifier.

As a special case, if there is only one class that occurs in the training set, then the resulting classifier will say that everything belongs to that class.

Definitely should hold for decision trees.

[not sure for NN if an extra class output node is present??]

<u>ScaleOblivious</u>

For any feature, the algorithm only cares about the relative order of feature values.

Let f:**R**->**R** be any monotonically increasing function. For **v**=$(v_1,...,v_n)$, define f(**v**) = $(f(v_1),v_2,...,v_n)$. Given a training set T, define f(T)={<f(**v**),c> | <**v**,c> in T}. The claim is that for any T and any **v**,

$$F(f(T))(f(\mathbf{v})) = F(T)(\mathbf{v}).$$

We have singled out feature number 1, but if the algorithm is **FeatureOrderOblivious**, a similar statement holds for each component of the feature vector.

This applies to decision tree implementations that don't do any arithmetic on the feature values (e.g., average them). Note entropy is not changed by f. Maybe for other approaches, specific restrictions on f will work, e.g., f is linear. NNs?

<u>TrainingAccurate</u>

If there is no noise in the training set, the classifier will be 100% accurate on the training data used to generate it.

Absence of noise means that if two elements of the training set have the same feature vectors then they have the same class. The claim is that if this holds, then

$$<v,c> \text{ in } T \Rightarrow F(T)(v)=c.$$

Should hold for DTs without pruning.

NNs?

Reinforcement learning? SVMs.

<u>ClassMonotonic</u>

Adding a training input with class c to a training set can only make it more likely that feature vectors classify to c.

Let T be a training set. Let v_0 be a feature vector and c_0 a class. Let T' = T U $\{<v_0,c_0>\}$.

For any feature vector v, if $F(T)(v)=c_0$, then $F(T')(v)=c_0$.

True for Matt's DT, maybe for any DT?

Any example where this fails?

<u>RedundantFeatureOblivious</u>

If there exists an injective function between two features for all instances, then the removal of one of those features will not matter.

Let T be a training set with a feature vector v that has n≥2 features, and for $v=(v_1, \ldots, v_n)$ let f:**R**->**R** be an injective function that exists between v_1 and v_n for all feature vectors in T; v_n is completely determined by v_1 and is essentially a redundant feature.

Let T' be a training set that is identical to T, except with v_n removed from all feature vectors; $v'=(v_1, \ldots, v_{n-1})$. If this is the case, then

$$F(T)(v_1,...,v_{n-1},f(v_1)) = F(T')(v_1,...,v_{n-1})$$

<u>IdenticalFeatureOblivious</u>

If there exists two identical features in the training set, then the removal of one of them will not matter.

Let T be a training set with a feature vector v that has $n \geq 2$ features, and for $v = (v_1, v_2, \ldots, v_n)$, let f:**R**->**R** be the identity function that exists between v_1 and v_2 for all v in T. Features v_1 and v_2 can be said to be identical, that is within each feature vector in T, $v_1 = v_2$. If T' be a training set that is identical to T, except with v_1 removed from all feature vectors, then

$$F(T)(v_1, v_2, ...v_n) = F(T')(v_2, \ldots, v_n)$$

Further Implications:

This property can be combined with **FeatureOrderOblivious** to apply to any two features in v, not just v_1 and v_2.

This property can also be combined with **ScaleOblivious** to extend that f can be any monotonically increasing function, not just the identity function.

Specific Properties for Decision Tree Schemes

Note:

Given a training set T, and k classifications, the information in bits of T is

$$\text{info}(T) = -\sum_{j=1}^{k}\left(\frac{freq(Cj, T)}{|T|}\ x\ \log_2\left(\frac{freq(Cj, T)}{|T|}\right)\right) \text{ bits}$$

where C_j is class in the class set C = (C_1, C_2, …) and freq(C_j, T) is the number of times that class C_j appears in T.

Given a feature v_o with n values (in a binary tree n=2; either ≤ or > a specified split value s_0), the information in bits remaining after splitting on v_o is

$$\text{info}_x(T) = \sum_{i=1}^{n}\frac{|Ti|}{|T|}x\,\text{info}(T_i)$$

In the case of a binary decision tree, T_1 represents all elements of T that have a value for v_0 that is ≤ s_0, and T_2 represents all elements of T that have a value for v_0 that is > s_0.

Entropy of T at v_0 or **E(T)(v_o)(s_o)** will refer to the result of this info_x(T) function on a specified feature v_0 (and split value s_0), which is how much information can be gotten out of a set of training data after splitting on v_o (at s_0). The lower the entropy, the better; it means the decision tree is getting closer to classifying the data.

<u>LowestEntropyFirst</u>

If in the training set, feature v_0 is most discriminatory (i.e. results in the lowest entropy), v_0 will be split on first in the learning of the tree.

Let T be a training set, and let v = (v_1, v_2, …, v_n) represent the feature vectors in T. Applying the entropy function for each feature for T will give us E(T)(v) = (E(T)(v_1), E(T)(v_2), …, E(T)(v_n)). Let the minimum value in E(T)(v) be represented with E(T)(v_o).

Splitting on this feature v_0 will result in the lowest possible entropy, thus feature v_0 best classifies the data and will be split on first.

<u>LowerEntropy</u>

In the generation of the decision tree, the resultant entropy of splitting on instances at a parent node at the best feature and split value will be less than the entropy of the instances at the parent.

Let T_p be a training set at a node in the decision tree where for at least two classes $c \neq c'$ and for at least two feature vectors $v \neq v'$ i.e. the instances are able to be split upon. Before splitting on a feature, $E(T_p) = 1 \times \text{info}(T_P)$. Let v_0 and s_o represent the best feature and value to split on for T_p, then

$$E(T_p)(v_o)(s_0) < E(T_p)$$