1 Introduction

A generic Machine Learning (ML) problem is learning a function $c : X \to Y$ given a dataset D containing sampled information about c. The learned approximated function h must be as close as possible to c also on elements $x \notin D$.

Types of ML problems:

- Supervised Learning: $D \subset \{(x, y) \ |x \in X, y \in Y\}$;
  - X finite sets: Discrete;
  - $X \subset R^n$: Continuos;
  - Y finite sets: Classification;
  - $Y \subset R^k$: Regression;
- Unsupervised Learning: $D \subset \{x|x \in X\}$ and Y must be created grouping similar instances in $X \cap D$;
- Reinforcement Learning: set of states S with associated actions and rewards, $D = \{(a_i^1...a_i^n, r_i)|i \in 1...|S|\}$ and the learned function is a transition policy;

Hyphotheses space H of all the approximation h of c, Learning can be a searching problem in H to find the best h. An hypothesis h is consistent with the train set D if $c(x) = h(x) \forall x \in D$.

We define $VS_{H,D} := \{h \in H|h \text{ is consistent with } D\}$.

If any subsets of teh instances can be represented in $VS_{H,D}$ and it is complete then it is not able to classify new instances $x' \notin D$. In fact different hypotheses in $VS_{H,D}$ may return different values for x'.

In case of noise in D then $VS_{H,D} = \emptyset$ and statistical methods are needed to remove the noise.

2 Classification Evaluation

Let Z a probability distribution over X and S n instances of X sampled with Z.

Performance evaluation are based on accuracy and error rate:

The true error of h repsect c and distribution Z is

$$error_Z(h) = Pr_{x \in Z}[c(x) \neq h(x)]$$

The sample error of h repsect c and S is

$$error_S(h) = \frac{|\{x \in S | c(x) \neq h(x)\}|}{|S|}$$

Let $accuracy(h) = 1 - error(h)$. $accuracy_S(h)$ estimates $accuracy_Z(h)$ but if $accuracy_S(h)$ is very high but $accuracy_Z(h)$ low then h is not useful. h must be accurate also in $x \notin S$.

IN order to bet and unbiased $accuracy_S(h)$ S must be chosen independently from D. For example we can split $D = T \cup S$ and train on T.

AN hypotheses h overfits the training data if exists an h' such that

$$error_S(h) < error_S(h') \wedge error_Z(h) > error_Z(h')$$

We can evaluate an hypotheses h estimating the error on different S, for example using the K-Fold method:

```
1 $D = S_1 \cup S_2 \cup ... \cup S_k$
2 for $i = 1$ to $k$
3     train $h_i$ on $D \setminus S_i$
4     $\delta = \delta + error_{S_i}(h_i)$
5 $error_{k,D} = \frac{\delta}{k}$
```

Accuracy can be not a valid metric in some cases. Consider binary classification problem with a D with 98% of ok labels and 2% of not ok labels. A dumb hypothesis that returns always ok has an high accuracy.

True class / Predicted class	ok	not ok
ok	True Positive (TP)	False Negative (FN)
not ok	False Positive (FP)	True Negative (TN)

From this values derives other measures:

- Accuracy: $(TP + TN)/(TP + FN + TN + FP)$

- Precision: $TP/(TP + FP)$
- Recall: $TP/(TP + FN)$
- False positives rate: $FP/(FP + TN)$
- False negatives rate: $FN/(FN + TP)$
- F-Measure: 2$*$Preision$*$Recall/(Precision + Recall)

A confusion matrix represents in each entry how many an element of class C_i is misclassified as an element of class C_j.

3 Decision Trees

Given X formed by values coming from a set of attributes, a *decision tree* is a tree defined as follows:

- each internel node test an attribute;
- each branch represents a value of an attribute;
- each eaf assign a classification value;

Such tree represents a disjunction of conjuctions of constraints on the attribute value $v \in A$ of the instance x.

An example dataset:

outlook	**temperature**	**humidity**	**wind**	**play?**
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes

To create the decision tree, we have to choose a target attribute. For example, considering the table above, we choose play?.

```
algorithm $ID3(examples, target\_attribute, attributes)$
    $root :=$ new $Node()$
    if $label(e)$ is $+ \ \forall e \in examples$
        $label(root) = +$
        return $root$
    if $label(e)$ is $- \ \forall e \in examples$
        $label(root) = -$
        return $root$
    if $attributes$ is $\emptyset$
        $label(root) =$ most common value of $target\_attribute$ in
            ↪ $examples$
        return $root$
    $A := best\_decision\_attribute(examples)$
    $label(root) = A$
    foreach $v \in A$
        $branch := add\_branch(root, test(A = v))$
        $E_v := \{e \in examples | e_A = v\}$
        if $E_v$ is $\emptyset$
```

```
            $leaf := \textbf{new } Node()$
            $label(leaf)$ = most common value of $target\_attribute$ **in**
                ↪ $examples$
            $add\_node(root, branch, leaf)$
        **else**
            $add\_subtree(root, branch, ID3(E_v, target\_attribute, attributes \setminus \{A\}))$
                ↪
```

With ID3 you can seach in the hypothesis space in a complete way and create an hypothesis tree.

How is computed the best decision attribute? ID3 select the attribute with the highest information gain that is how well it separates the training examples. We can measure the information gain using the entropy.

Let p_+ the proportion of positive examples ($|positive\ examples|/|examples|$) and $p_- = 1 - p_+$ the proprotion of negative examples.

$$entropy(examples) = -p_+ * log_2(p_+) - p_- * log_2(p_-)$$

Then we define the information gain:

$$gain(examples) = entropy(examples) - \sum_{v \in A} \frac{|E_v|}{|examples|} * entropy(examples)$$

To avoid overfitting we can stop growing the tree when the data is not splitting in a significant way or build the complete tree and then perform post pruning replacing subtrees with the more common value in order to improve accuracy on validation set.

If attributes have contonious values, the tree must be build using intervals (e.g. $A < 10$ and $A > 10$).

Random Forest is an esemble method that builds many trees using random criteria (e.g. random picking attributes). The reuslt is the most common classification of the trees. Random Forest is less prone to overfitting.

4 Bayes Networks

Let's recall some definition from the thery of probability.

- Conditional probability: $P(a|b) = P(a \wedge b)/P(b)$ if $P(b) \neq 0$;
- Product rule: $P(a \wedge b) = P(a|b)/P(b) = P(b|a)/P(a)$;
- Sum rule: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$;
- Total probability: $P(a) = P(a|b) * P(b) + P(a|\neg b) * P(\neg b)$ and in general $P(X) = \sum_{y \in values(Y)} P(X|Y = y) * P(Y = y)$ when all values y are mutually exclusive.
- Idependence: X is independent from Y given Z if $P(X, Y|Z) = P(X|Z)$;
- Bayes rule: $P(a|b) = \frac{P(b|a)*P(a)}{P(b)}$;

A classification problem can be interpreted as a probabilistic estimation. Given $x' \notin D$ the best prediction is $h^*(x') = y^*$

$$v^* = \arg\max_{y \in Y} P(y|x', D)$$

Also learning is probabilistic. From the Bayes rule we have that

$$P(h|D) = \frac{P(D|h) * P(h)}{P(D)}$$

In general we want the most probable hypothesis given D, we call it *maximum a posteriori hypothesis*:

$$h_{MAP} = \arg\max_{h \in H} P(h|D) = \arg\max_{h \in H} \frac{P(D|h) * P(h)}{P(D)} = \arg\max_{h \in H} P(D|h)*P(h)$$

If we assume that all the hypothesis have the same probability we can simplify an choose the *maximum likehood hypothesis*:

$$h_{ML} = \arg\max_{h \in H} P(D|h)$$

Note: $h_{ML} = \arg\max_{h \in H} P(D|h) = \arg\max_{h \in H} log(P(D|h))$ and you can derivate to get the max.

A learning process can be the brute force of all $h \in H$ returning h_{MAP} after computing all posteriori probabilities.

Given $x' \notin D$ $h_{MAP}(x')$ may not be the best classification!

Here we must introduce the *Bayes Optimal Classifier*:

$$P(y|x', D) = \sum_{h \in H} P(y|x', h) * P(h|D)$$

So we have that:

$$y_{opt} = \arg\max_{y \in Y} \sum_{h \in H} P(y|x', h) * P(h|D)$$

This is very powerful and returns the classification with the highest probability but it is not practicable when H is large.

Naive Bayes Classifier uses condition idependence to estimate the solution. Consider each x composed by attributes $(a_1, a_2, ..., a_n)$.

$$\arg\max_{y \in Y} P(y|x, D) = \arg\max_{y \in Y} P(y|a_1, a_2, ..., a_n, D)$$

For $x' \notin D$ the most probable classification thanks to Bayes is

$$y_{MAP} = \arg\max_{y \in Y} P(y|a_1, a_2, ..., a_n, D) = \arg\max_{y \in Y} P(a_1, a_2, ..., a_n|y, D) * P(y|D)$$

With the assuption of independece between the attributes we have that

$$y_{NB} = \arg\max_{y \in Y} P(y|D) * \Pi_{i=1}^{n} P(a_i|y, D)$$

5 Linear Classification

Let $X \subset R^d$ and $Y = \{C_1, ..., C_k\}$ with linearly separable data.

Linear discriminant function:

- 2 classes: $y(x) = w^T * x + w_0$;
- k classes: $y_k(x) = w_k^T * x + w_{k0}$;

When having k classes we cannot use combinations of models with two classes (class C_i and $\neg C_i$).