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?.

```
1  algorithm ID3(examples, target_attribute, attributes)
2      root := new Node()
3      if label(e) is + ∀e ∈ examples
4          label(root) = +
5          return root
6      if label(e) is − ∀e ∈ examples
7          label(root) = −
8          return root
9      if attributes is ∅
10         label(root) = most common value of target_attribute in
               ↪ examples
11         return root
12     A := best_decision_attribute(examples)
13     label(root) = A
14     foreach v ∈ A
15         branch := add_branch(root, test(A = v))
16         E_v := {e ∈ examples|e_A = v}
17         if E_v is ∅
```

```
18              leaf := new Node()
19              label(leaf) = most common value of target_attribute in
                    ↪ examples
20              add_node(root, branch, leaf)
21          else
22              add_subtree(root, branch, ID3(E_v, target_attribute, attributes \ {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:

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

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: $y_k(x) = w_k^T * x + w_{k_0}$, $k = 1...K$;

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

$$y(x) = \tilde{W}^T * \tilde{x}$$

With

$$\tilde{W} = \begin{pmatrix} w_{1_0} & w_{2_0} & ... & w_{K_0} \\ w_1 & w_2 & ... & w_k \end{pmatrix} \quad \tilde{x} = \begin{pmatrix} 1 \\ x \end{pmatrix}$$

Consider a dataset $D = \{(x_n, t_n)_{n=1}^N\}$. 1-K coding cheme for t_n that is a vector of K binary values. All values are 0 except from the value at place i when $x_n \in C_i$.

5.1 Least Squares

$$\tilde{X} = \begin{pmatrix} \tilde{x}_1^T \\ ... \\ \tilde{x}_N^T \end{pmatrix} \quad T = \begin{pmatrix} t_1^T \\ ... \\ t_N^T \end{pmatrix}$$

Minimizes sum of squares error function (Tr is the trace operator, the sum on the principal diagonal):

$$E(\tilde{W}) = \frac{1}{2} * Tr\Big\{(\tilde{X} * \tilde{W} - T)^T * (\tilde{X} * \tilde{W} - T)\Big\}$$

$$\tilde{W} = (\tilde{X}^T * \tilde{X})^{-1} * \tilde{X}^T * T$$

This is not robust to outliners.

5.2 Fisher

Two classes C_1, C_2 case. Find the medium $m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n$ and $m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n$.

There is a line $J(w) = w^T * (m2 - m1)$. We want maximizes it with the constraint $||w|| = 1$. We have that $w = -\frac{1}{m_2 - m_1}$ that is the ortogonal line to J.

To be robust to outlines we use the variance.
Fihser criterion:

$$J(w) = \frac{w^T * S_B * w}{w^T * S_W * w}$$

with

$S_B = (m_2 - m_1) * (m_2 - m_1)^T$
$S_W = \sum_{n \in C_1}(x_n - m_1) * (x_n - m_1)^T + \sum_{n \in C_2}(x_n - m_2) * (x_n - m_2)^T$

$w^* = -\frac{1}{S_w^{-1} * (m_2 - m_1)}$ maximizes J.

5.3 Perceptron

$$o(x) = \begin{cases} 1 & \text{if } w^T * x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Weigths $w_i \in w$ must be learned from $D = \{(x_n, t_n)_{n=1}^N\}$ that minimize the square error (loss function):

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(t_n - o_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(t_n - w^T * x)^2$$

With $o = w_0 + w_1 * x_1 + ... + w_d * x_d$ ($x_0 = 1$).
Training rule:

$$\frac{\partial E(w)}{\partial w_i} = \sum_{n=1}^{N}(t_n - w^T * x_n) * (-x_{i,n})$$

Updating w_i: $w_i \leftarrow w_i - \eta * \frac{\partial E(w)}{\partial w_i}$ whee η is small (like 0.05) and called learning rate.
Algorithm:

1. Initialize w' with small random values;
2. Repeat the udating w_i' procedure until termination condition (e.g. finite number of iteration or treshold on $E(w)$);
3. Output w';

5.4 Support Vector Machine

5.5 Linear Models for Non Linear Dataset

If a dataset is not linearly separable we can consider to apply a non linear transformation to the input and then apply a linear model.

6 Linear Regression

Learning $X \subseteq R^d$ and $Y = R$ from datatset $D = \{(x_n, t_n)_{n=1}^N\}$.

Linear model for linear function: $y(x, w) = w_0 + w_1 * x_1 + ... + w_d * x_d = w^T * x$.

$$x = \begin{pmatrix} 1 \\ x_1 \\ ... \\ x_d \end{pmatrix} \quad w = \begin{pmatrix} w_0 \\ w_1 \\ ... \\ w_d \end{pmatrix}$$

Linear basis functions: $y(x, w) = w^T * \phi(x)$ with ϕ is a not linear transformation of the input. This is still linear in functions of w.

Least squares error minimization:

$$E_D(w) = \frac{1}{2} \sum_{n=1}^{N} \Big(t_n - w^T * \phi(x_n)) \Big)^2 = \frac{1}{2} * (t - \Phi * w)^T * (t - \Phi * w)$$

Optimal condition: $\nabla E_D = 0$, $w_{ML} = (\Phi^T * \Phi)^{-1} * \Phi^T * t$.

Learning with stochastic gradient descent:

$\hat{w} \leftarrow \hat{w} - \eta \nabla E_n$.

So $\hat{w} \leftarrow \hat{w} - \eta * \Big(t_n + \hat{w}^T * \phi(x_n) \Big) * \phi(x_n)$.

7 Markow Decision Process

Notation:

- X set of states;
- A set of actions;
- δ transition function;
- Z set of observations;

The properties are that once a state is know it contains all the information to predict the future that does not depends from previous states.

Let $r : X \to R$ the reward function. In MDP state are fully observable. The objective of MDP is to find an optial policy $\pi : X \to A$ that for each state gives the optimal action to take.

Optimality is maximizing the comulative reward $V^{\pi}(x_1) = E\big[r_1 + \gamma * r_2 + \gamma^2 * r_3 + ...\big]$ where $r_t = r(x_t, a_t, x_{t+1})$, $a_t = \pi(x_t)$ and $\gamma \in [0, 1]$ called discount factor for future rewards.

So the optimal policy is $\pi^* = \arg\max_{\pi} V^{\pi}(x) \forall x \in X$.

7.1 One State MDP

Assume $X = \{x_0\}$, $\delta(x_0, a_i) = x_0 \forall a_i \in A$, $r(x_0, a_i, x_0) = r(a_i)$. The optimal policy is one actions in A.

If r is deterministic and known $\pi^*(x_0) = \arg\max_{a_i \in A} r(a_i)$.

If r is deterministic and unknown then execute each a_i ($|A|$ executions), collect the rewards and return the best a_i.

If r is not deterministic and known $\pi^*(x_0) = \arg\max_{a_i \in A} E[r(a_i)]$.

If r is not deterministic and unknown collect many times T the rewards for actions and update a data structure at entry a_i with the average of all rewards of a_i.

8 Q Learning

If functions r and δ are not known we have training examples in form $\{(x_n, a_n, r_n)_{n=1}^N\}$. The agent cannot predict the effect of an action. So the learning now is:

1. choose an action;
2. execute such action;
3. observe the resulting state;
4. collect reward;

First type is value iteration. The agent could learn $V^{\pi^*}(x) = V^*(x)$ and use it to determinate the optimal policy $\pi^*(x) = \arg\max_{a \in A} \left[r(x,a) + \gamma * V^*(\delta(x,a)) \right.$ but this is impossible since r and δ are unknown.

Let $Q^\pi(x,a) = r(x,a) + \gamma * V^\pi(\delta(x,a))$ the expected value when executing a in x and then act accordingly to π. We have that:

$$Q^\pi(x,a) = \sum_{x'} P(x'|x,a) * (r(x,a,x') + \gamma * V^\pi(x'))$$

If the agent learns Q the π^* can be computed without knowing r and δ.

The learning rule in the deterministic case is: $Q(x_t, a_t) = r(x_t, a_t) + \gamma * \max_{a' \in A} Q(x_{t+1}, a')$

Let $\hat{Q}$ the current learner approximation and $\bar{r}$ the immediate reward.

$$\hat{Q}(x,a) = \bar{r} + \gamma * \max_{a' \in A} \hat{Q}(x', a')$$

The algorithm has a table entry for each couple x, a and tiem instant.

1. Set all $\hat{Q}_{(0)}(x,a) = 0$;
2. Observe x;
3. For each time until the termination condition do
   (a) Choose an action a;
   (b) Execute it;
   (c) Observe the new state x';
   (d) Collect the immediate reward $\bar{r}$;

(e) Set $\hat{Q}_{(t)}(x,a) = \bar{r} + \gamma * \max_{a' \in A} \hat{Q}_{(t-1)}(x',a')$;

(f) $x = x'$;

Optimal policy $\pi^*(x) = \arg\max_{a \in A} Q_{(\text{last})}(x,a)$.

How to choose an action? The agent can select an cation that maximize $\hat{Q}(x,a)$ (*exploitation*) or one with a low value of $\hat{Q}(x,a)$ (*exploration*).

Tipically do exploration with probability ϵ that decrease with time.

A strategy is teh soft-max strategy. Higher $\hat{Q}$ have higher probabilities but all actions have probability more than 0.

$$P(a_i|x) = \frac{k^{\hat{Q}(x,a_i)}}{\sum_j k^{\hat{Q}(x,a_j)}}$$

k higher selects an higher $\hat{Q}$ with more probability, so tipically k increase with time (first exploration then exploitation).

9 Not Deterministic Reinforcement Learning

In the not deterministic case we have that:

$$Q^{\pi}(x,a) = E\big[r(x,a)+\gamma*V^{\pi}(\delta(x,a))\big] = E[r(x,a)]+\gamma*\sum_{x'}P(x'|x,a)*\max_{a'}Q(x',a')$$

The training rule is

$$\hat{Q}_n(x,a) = \hat{Q}_{n-1}(x,a) + \alpha * \big(r + \gamma * \max_{a'}\hat{Q}(x',a') + \hat{Q}(x,a)\big)$$

With $\alpha = \alpha_{n-1}(x,a) = \frac{1}{1+visits_{n-1}(x,a)}$ and visits the number of visits of the pairs x, a until time n.

In not deterministic $\hat{Q}_{n+1}(x,a) \geq \hat{Q}_n(x,a)$ is not valid, use Temporal Difference or SARSA.

10 Hidden Markov Models

States are discrete and not observable. Observations (discrete or continuos) z_t are avaiable.

- Transition model: $P(x_t|x_{t-1})$;
- Observation Model: $P(z_t|x_t)$ $(b_k(z_t) = P(z_t|x_t = k))$;
- Initial distribution: $\pi_0 = P(x_0)$;
- State transition matrix A where $A_{i,j} = P(x_t = j|x_{t-1} = i)$;

11 Kernels

Kernel function: similarity measure $k(x, x') \geq 0$. Typically is also simmetric.

Linear kernel: $k(x, x') = x^T * x'$.

Considering a linear model $y(x, w) = w^T * x$ we want to minimize $J(w) = (t - X * w)^T * (t - X * w) + \lambda * ||w||^2$.

... TODO

Any linear model with kernel k: $y(x, \hat{w}) = \sum_{i=1}^{N} \alpha_i * k(x, x')$.

The solution is $\alpha = (K + \lambda * I_N)^{-1} * t$.

The Gram matrix is:

$$K = \begin{pmatrix} k(x_1, x_1) & ... & k(x_1, x_N) \\ ... & ... & ... \\ k(x_N, x_1) & ... & k(x_N, x_N) \end{pmatrix}$$

Kerne trick: substitute an inner product with a kernel to extend a linear model.

12 Instance Based