Summer Research Report

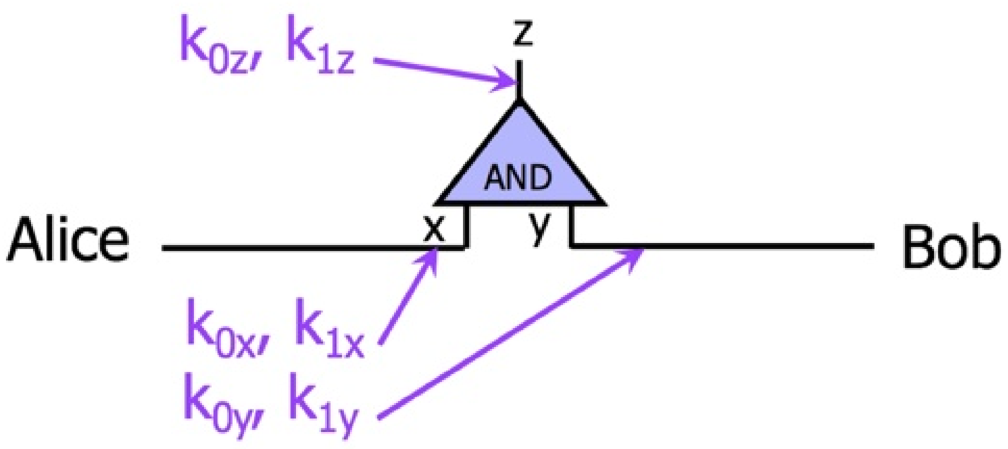
王倩倩 2019.07 - 2019.09

Summer Research Report

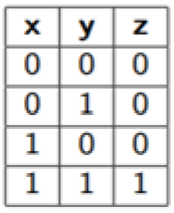
Week 1

Yao's garbled circuit

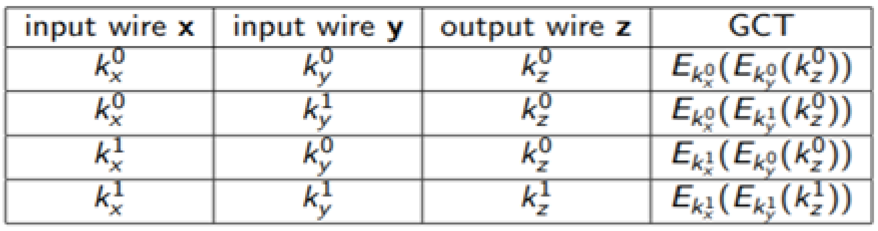
- **Circuit**：计算问题可以转化为电路解决：1. 简单：加法/比较/乘法 2. 复杂：深度学习
- 基本结构：



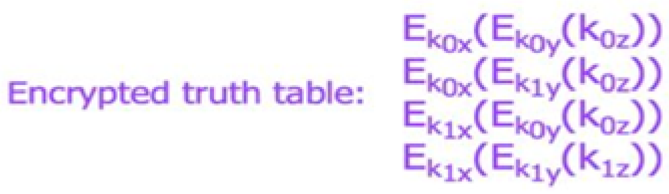
- 具体流程：
  - Alice准备：Alice 给每条线路为每条线路选择两个key，共6个key
    - 随机选择两组密钥k，每组中各选一个K（数值随机，分别对应0和1）
      - 注：她是电路生成者，所以所有的wire的密钥都知道
    - 计算出Garbled Computation Table（GCT）



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

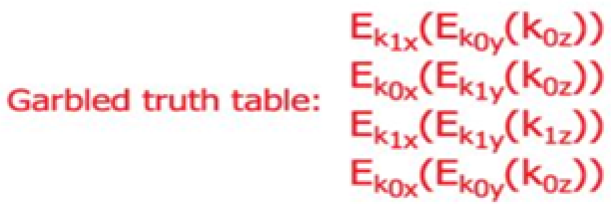


input wire **x**	input wire **y**	output wire **z**	GCT
k_x^0	k_y^0	k_z^0	$E_{k_x^0}(E_{k_y^0}(k_z^0))$
k_x^0	k_y^1	k_z^0	$E_{k_x^0}(E_{k_y^1}(k_z^0))$
k_x^1	k_y^0	k_z^0	$E_{k_x^1}(E_{k_y^0}(k_z^0))$
k_x^1	k_y^1	k_z^1	$E_{k_x^1}(E_{k_y^1}(k_z^1))$



Encrypted truth table: $E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

  - 例：与门→

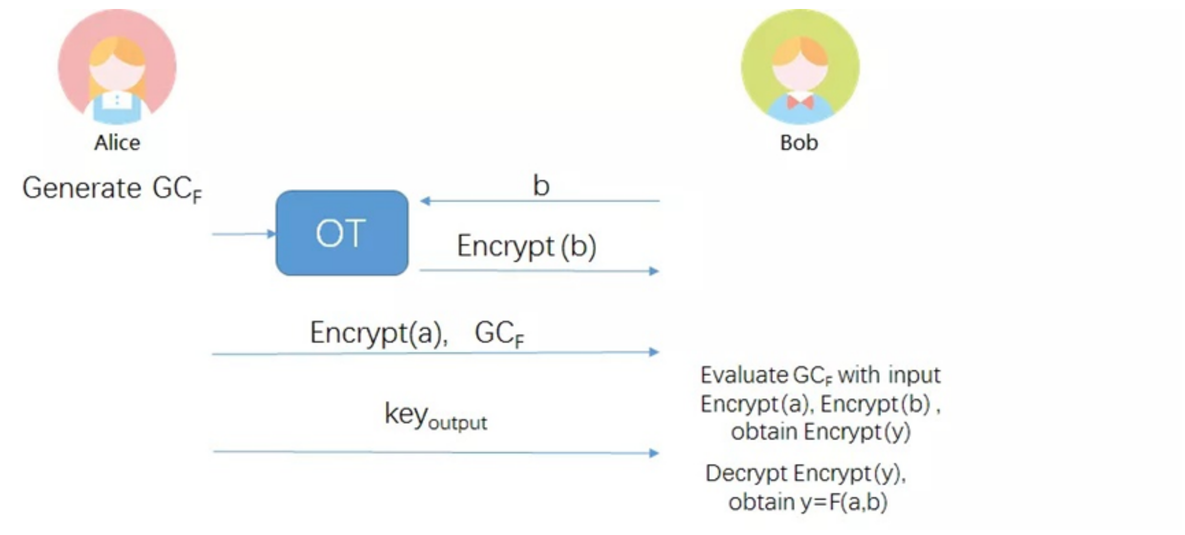


Garbled truth table: $E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

  - Alice发送

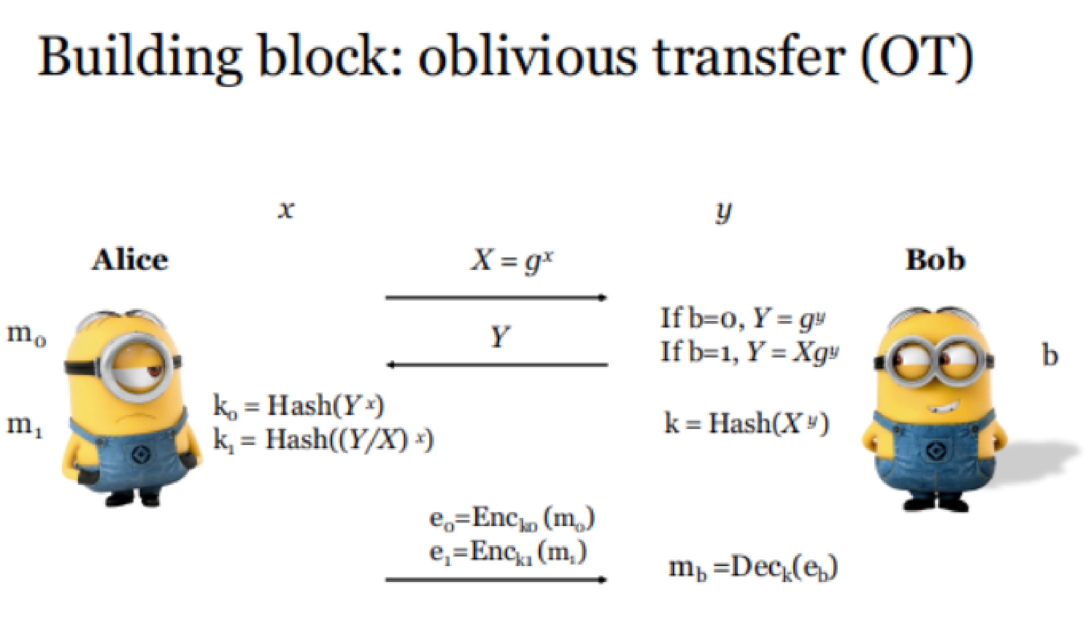
        - 发送自己的输入对应的key（加密后）0对应k_0x，1对应k_1x
        - 发送GCT
        - 发送与Bob有关的key（加密后）0对应k_0y， 1对应k_1y
            - OT protocol
    - Bob接收
        - 收到Alice发送的key
            - 根据k_x和k_y对加密表的每一行尝试解密，最终只有一行能揭秘成功
            - 将k_z发送给Alice，Alice对比得出是k_0z还是k_1z，得知结果是0还是1

Oblivious Transfer(OT)


Alice
Bob
Generate GC_F
OT
b
Encrypt (b)
Encrypt(a), GC_F
key_{output}
Evaluate GC_F with input Encrypt(a), Encrypt(b), obtain Encrypt(y)
Decrypt Encrypt(y), obtain y=F(a,b)

注：指发送方sender传输给接收方receiver n个数据，但是不知道receiver收到了n中的哪一个，而receiver也只能解码其中一个。

Week 2 补充OT protocol细节推理



- Alice生成秘钥：

    - Alice通过生成元g和阶q定义一个乘法循环群G
    - Alice在集合R={0, 1, 2, ..., q-1}中随机选择一个整数x
    - Alice根据群G的生成元和阶生成群中的一个元素 $X = g^x$
    - Alice将{G, q, g, X}作为公钥发布，x作为私钥妥善保存
- Bob根据$\{G, q, g, X\}$生成群中的一个元素

$$Y = \begin{cases} g^y, & \text{b=0} \\ Xg^y, & \text{b=1} \end{cases}$$

- 发送给Alice
- Alice根据自己拥有的资源m_0和m_1计算k_0和k_1

$$k_0 = Hash(Y^x)$$
$$k_1 = Hash((Y/X)^x)$$

- Bob计算k

$$k = Hash(X^y)$$
$$If\ b = 0,\ k_0 = Hash(g^{xy}), k_1 = Hash(g^{(y-x)x})$$
$$If\ b = 1,\ k_0 = Hash(g^{x(x+y)}), k_1 = Hash(g^{xy})$$
$$k = Hash(g^{xy})$$

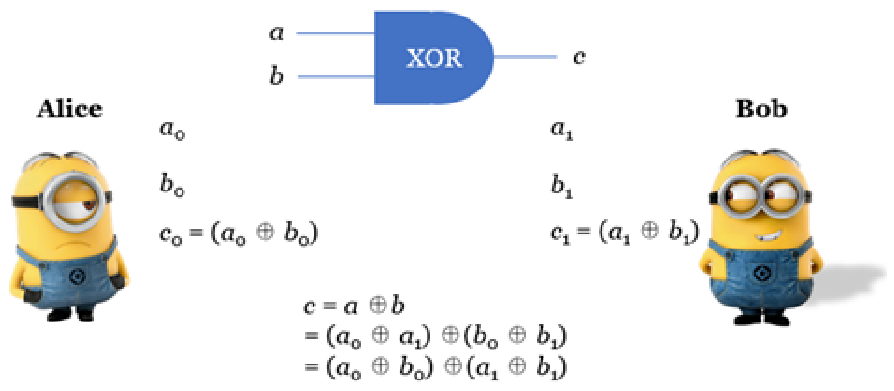
- Alice将加密后的e_0和e_1发送给Bob，Bob匹配k计算出自己需要的信息

Goldreich-Micali-Wigderson(GMW) Protocol

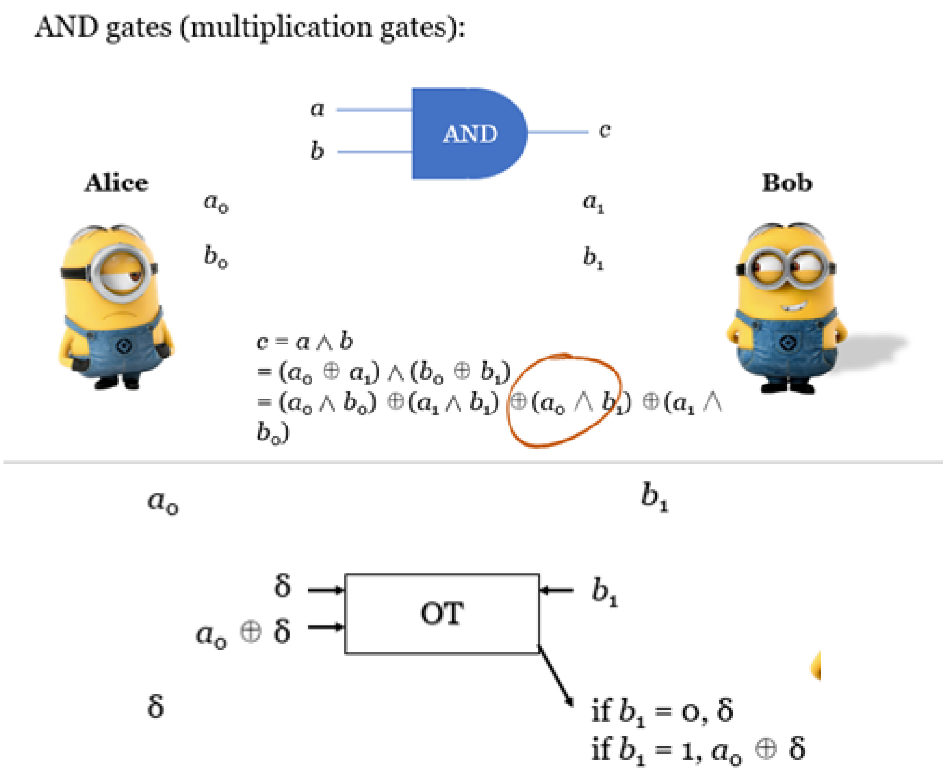
- 使用情况
    - 恶意敌手模型：中途可能退出，GMW可以解决一部分（略）。
    - 半诚实模型： GMW可以解决 中途不会退出也不篡改运行结果。
- 流程
    - 所有输入数据共享
    - 扫描电路
        - 输入数据共享
        - 每一步的输出结果由协议共享给各方（GMW）

        ·XOR：XOR （addition gates）

XOR gates (addition gates):



·AND：OT （multiplication gates）


AND gates (multiplication gates):
a
b
AND
c
Alice
Bob
a_0
b_0
a_1
b_1
$c = a \wedge b$
$= (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$
$= (a_0 \wedge b_0) \oplus (a_1 \wedge b_1) \oplus (a_0 \wedge b_1) \oplus (a_1 \wedge b_0)$
a_0
b_1
δ
$a_0 \oplus \delta$
OT
b_1
δ
if $b_1 = 0, \delta$
if $b_1 = 1, a_0 \oplus \delta$

- 重复向前

- 输出

Week 2 - Week 3

补充密码学知识

- Differences between boolean circuit and arithmetic circuit

  Boolean circuit:

  - Input: bit inputs and the gates of the circuit correspond to boolean operations (such as **XOR, AND**).
  - Transform: one can always use some representation of the field elements, and translate the field operations to boolean operations.
  - Application: many applications of, say, secure computation involve **non-arithmetic operations** (for example, integer comparison, which is a basic procedure in many computations). In this case, implementing this non-arithmetic operation as an arithmetic one would be very **inefficient**, and boolean circuits are a more natural representations.
  - boolean circuits correspond to arithmetic circuits over the field F_2

  Arithmetic circuit:

  - Input: elements of some field F, and the gates of the circuit correspond to arithmetic operations, i e, field operations (such as **additions and multiplications**).
  - Transform: Conversely, one **can always translate a boolean circuit to an arithmetic one**, by interpreting 0 as the neutral for addition over F,

1 as the neutral for multiplication over F, and translating the boolean gates to arithmetic operations (e.g. a OR b becomes a+b−a∗b).
    - Application: In cryptography, it is quite common to rely on computation on field elements, and we **have cryptographic tools that allow us to manipulate field elements in an atomic way**, *id est*, treating a given field element as a single input (and not as a string of bits). In this case, it makes sense to represent the computation with an arithmetic circuit.
    - arithmetic circuits are a generalization of boolean circuits to arbitrary fields

为什么Garbled Circuit使用的是Boolean Circuit?

- boolean circuit 理论上可以将任何函数表达出来
- 因为是bit输入，所以GCT表可以较为简单的写出来，并传达给对方，若是arithmetic circuit则可能表过大。

- Differences between Garbled Circuit and GMW protocol

	GC	**GMW**
算法深度	constant	O(depth), interactive
电路结构	Boolean	can be generalized to arithmetic
容易扩展	×	√

补充群论知识（Group Theory）

- n是一个正整数，对于任意一个正整数，被n除后余数会有几种可能，0、1....n-1。所有的这些剩余类所构成的集合叫**模n剩余类群**。
    - 其中的乘法运算定义为：相加后再和n求模
    - i.e. Z_5的生成元是1、2、3、4
    - Z_n中的全部生成元：r是元的充分必要条件是r与n互素。
- 一个群若只有一个生成元就叫**循环群(cyclic group)**。
    - 生成循环群的单个特殊元素***g***称为生成元(**generator**)，群中元素的个数称为阶(**order**)。
- 设n为素数，定义集合{g^i|i = 0, 1, ..., n -1}上的乘法运算：$g_i \neq g_j \in G$, $g_i \cdot g_j = g^i \cdot g^j = g^{(i+j) mod n}$,则乘法对集合成群，称作由生成元g生成的n阶乘法循环群，记作$MG^{(n)}$.

  乘法循环群形如{g^0,g^1,g^2,....,g^{n-1}}，其中g^0 称作单位元，记作e。

- 有限阶的循环群和模n剩余类群同构，所以只需分析模n剩余类群就可以获得所有有限阶群得结构

Paper Review: SecureML： A System for Scalable Privacy-Preserving

Preliminaries

1.1 Machine Learning

Linear regression

1. Regression is a statistical process **to learn a function g** such that g(x_i) ≈ y_i
2. The **function g** is assumed to be linear $g(\mathbf{x}_i) = \sum_{j=1}^{d} x_{ij} w_j = \mathbf{x}_i \cdot \mathbf{w}$
3. To learn the coefficient vector **w**, a cost function **C(w)** is defined and w is calculated by the optimization argmin$_w$ C(**w**). In linear regression, a commonly used cost function is $C(w) = \frac{1}{n} \sum C_i(\mathbf{w}), where C_i(\mathbf{w}) = \frac{1}{2}(\mathbf{x}_i \cdot \mathbf{w} - y_i)^2$
4. The solution for this optimization problem can be computed by solving the linear system$(X^T \times X) \times w = X^T \times Y$.

Stochastic gradient descent (SGD)

1. SGD is an effective approximation algorithm for approaching a local minimum of a function, step by step.
2. $\boldsymbol{w}$ is initialized as a vector of random values or all 0s. In each iteration, a sample $(\boldsymbol{x}_i, y_i)$is selected randomly and a coefficient w j is updated as $w_j := w_j - \alpha \frac{\partial C_i(\boldsymbol{w})}{\partial w_j}$
3. Substituting the cost function of linear regression, the formula becomes $w_j := w_j - \alpha(x_i \cdot w - y_i)x_{ij}$
4. The phase to calculate the predicted output forward propagation:$y_i^* = \boldsymbol{x}_i \cdot \boldsymbol{w}$
5. The phase to calculate the change $\alpha(y_i^* - y_i)x_{ij}$is called backward propagation.

Mini-batch

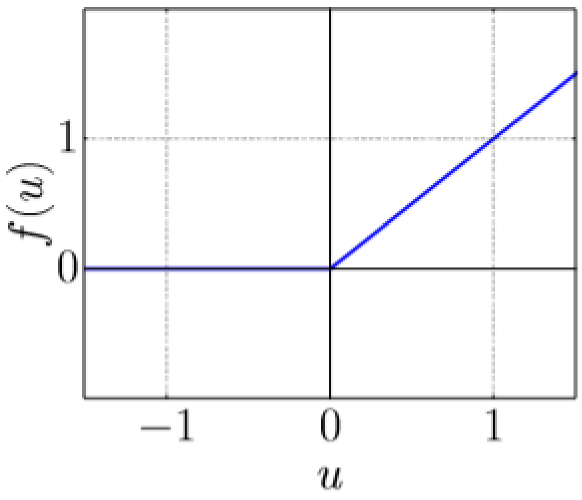
1. A small batch of samples are selected randomly and w is updated by averaging the partial derivatives of all samples on the current **w**.
2. With mini-batch, the update function can be expressed in a vectorized form: $\mathbf{w} := \mathbf{w} - \frac{1}{|B|} \alpha \mathbf{X}_B^T \times \mathbf{w} - \mathbf{Y}_B$.

Logistic Regression

1. $g(x_i) = f(x_i \cdot \mathbf{w})$
2. In logistic regression, the activation function is defined as the logistic function $f(u) = \frac{1}{1+e^{-u}}$.
3. The cost function is changed to the cross entropy function $C_i(\mathbf{w}) = -y_i log y_i^* - (1 - y_i) log(1 - y_i^*)$
4. The mini-batch SGD algorithm for logistic regression updates the coefficients in each iteration as follows: $\mathbf{w} := \mathbf{w} - \frac{1}{|B|} \alpha \mathbf{X}_B^T \times (f(\mathbf{X}_B \times \mathbf{w}) - \mathbf{Y}_B)$

Neural Networks

1. Neural networks are a generalization of regression to learn more complicated relationships between high dimensional input and output data.
2. Each node in the hidden layer and the output layer is an instance of regression and is associated with an activation function and a coefficient vector. Nodes are also called neurons. Popular activation functions include the logistic and the **RELU** function (f (u) = max(0, u)).



3. For classification problems with multiple classes, usually a **softmax** function .
4. The matrix X_i of the i_{th} layer is computed as $\mathbf{X}_i = f(\mathbf{X}_{i-1} \times \mathbf{W}_i)$.
5. To calculated it, we compute the vectors $\mathbf{Y}_i = \frac{\partial C(\mathbf{W})}{\partial \mathbf{U}_i}$ iteratively, where $\mathbf{U}_i = \mathbf{X}_{i-1} \times \mathbf{W}_i$.
6. $\mathbf{Y}_m$ is initialized to $\frac{\partial C}{\partial \mathbf{X}_m} \odot \frac{\partial f(\mathbf{U}_m)}{\partial \mathbf{U}_m}$.
7. By the chain rule, $\mathbf{Y}_i = (\mathbf{Y}_{i+1} \times \mathbf{W}_i^T) \odot \frac{\partial f(\mathbf{U}_i)}{\partial U_i}$.
8. Finally, the coefficients are updated by letting $\mathbf{W}_i := \mathbf{W}_i - \frac{\alpha}{|B|} \cdot \mathbf{X}_i \cdot \mathbf{Y}_i$.

1.2 Secure Computation

OT Protocol

Garbled Circuit 2PC

Secret Sharing and Multiplication Triplets

Additive sharing

- Additively share ***(Shr^A(·))***: P_0 generates $a_0 \in Z_{2^l}$ uniformly at random and sends $a_1 = a - a_0 \bmod 2^l$ to P_1 .
- Reconstruct ***(Rec^A(·,·))***: an additively shared value $<a>$, P_i sends $<a>$ to P_{1-i} who computes $<a>_0 + <a>_1$.

- Multiply ***$Mul^A(;;)$***: Use Beaver's precomputed multiplication triplet technique. Lets assume that the two parties already share ***<u>, <v>, <z>*** where u,v are uniformly random values in Z_{2^l} and *z = uv mod* 2^l. Then P_i locally computes ***$<e>_i = <a>_i - <u>_i$*** and ***$<f>_i = <b>_i - <v>_i$***. Both parties run ***$Rec(<e>_0, <e>_1)$*** and ***$Rec(<f>_0, <f>_1)$***, and P_i lets ***$<c>_i = -i \cdot e \cdot f + f \cdot <a>_i + e \cdot <b>_i + <z>_i$***.

Boolean sharing

- Like additively sharing, but the addition operation is replaced by the XOR operation (⊕) and multiplication is replaced by the AND operation ***(AND(;;))***.

Yao sharing

- in all garbling schemes, for each wire *w* the garbler (P_0) generates two random strings $\mathbf{k_0^w, k_1^w}$, a random permutation bit r_W and lets $\mathbf{K_0^w = k_0^w \| r_W}$ and $\mathbf{K_1^w = k_1^w \| (1 - r_W)}$.
- A Yao sharing of a is $\mathbf{<a>_0^Y = K_0^w}$ and $\mathbf{<a>_1^Y = K_a^w}$. To reconstruct the shared value, parties exchange their shares. XOR and AND operations can be performed by garbling/evaluating the corresponding gates.
- Switch from a Yao sharing to a Boolean sharing, P_0 lets $\mathbf{<a>_0^B = K_0^w[0]}$ and P_1 lets $\mathbf{<a>_1^B = <a>_1^Y[0]}$.

2.1 Privacy Preserving Linear Regression

Notation: Two servers S_0 and S_1, Shares $<X>_0$, $<Y>_0$ and $<X>_1$, $<Y>_1$

- Secretly share the coefficients ***w***
  - Setting $<w>_0$, $<w>_1$ to be random. It's updated and remains secret after each iteration of SGD, until the end when it is reconstructed.
  - The update function for linear regression is $w_j := w_j - \alpha(x_i \cdot w - y_i)x_{ij}$ Only consisting of additions and multiplications. → $< w_j >:=< w_j > -\alpha Mul^A(< x_{ik} >, < w_k >) - < y_i >, < x_{ij} >)$
- Online phase: train the models
- Offline phase: multiplication triplets generation

Online Phase

Vectorization in the Shared Setting

Addition: Letting *$<C>_i = <A>_i + <B>_i$*.

Multiplication:

- we take shared matrices ***<U>,<V>,<Z>***, where each element in U and V is uniformly random in $Z_{2^\wedge l^\wedge}$, U has the same dimension as A, V has the same dimension as B and Z = U × V mod 2^l. S_i computes ***<E>$_i$ = <A>$_i$ −<U>$_i$, <F>$_i$ = <B>$_i$ −<V>$_i$*** and sends it to the other server. Both servers reconstruct E and F and set ***<C>$_i$ = −i·E × F + <A>$_i$ × F + E × <B>$_i$ + <Z>$_i$.***

- Applying the technique to linear regression, in each iteration, we assume the set of mini-batch indices B is public, and perform the update $< w >:=< w > -\frac{1}{|B|}\alpha Mul^A(< \mathbf{X}_B^T >, Mul^A(< \mathbf{X}_B >, < \mathbf{w} >)) - < \mathbf{Y}_B >)$
- At the beginning of the online phase, <E>$_i$ =<X>$_i$ −<U>$_i$ is computed and exchanged to reconstruct E through one interaction. After that, in each iteration, E_B is selected and used in the multiplication protocol, without any further computation and communication.
- Then the multiplication triplets are precomputed with the following property.
- let $\mathbf{Z}[i] = \mathbf{U}_{B_i} \times \mathbf{V}[i]$ and $\mathbf{Z}^{'}[i] = \mathbf{U}_{B_i}^T \times \mathbf{V}^{'}[i]$for $i = 1, \ldots, t$.

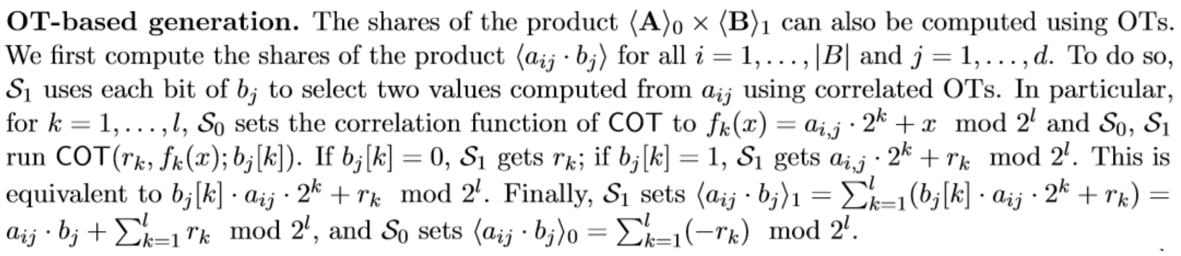
Arithmetic Operations on Shared Decimal Numbers

We first transform the numbers to integers by letting $x^{'} = 2^{l_D}x$ and $y^{'} = 2^{l_D}y$ and then multiply them to obtain the product z = x'y'. Note that z has at most 2^{l_D} bits representing the fractional part of the product, so we simply **truncate** the last l^D bits of z such that it has at most l^D bits representing the fractional part.

Offline Phase

LHE-based(同态加密) generation

OT-based generation



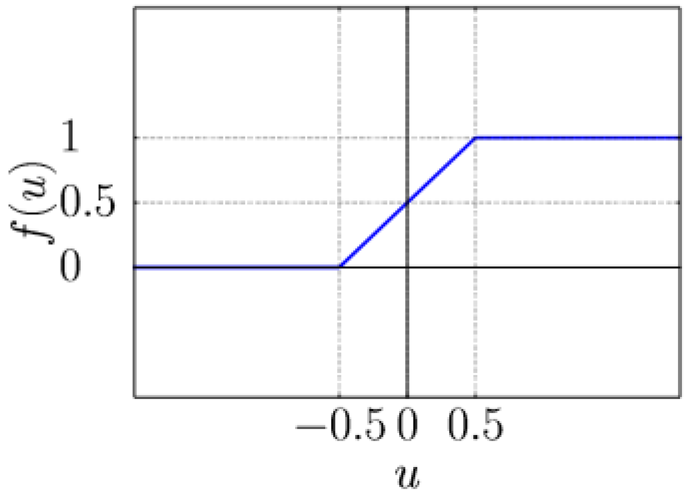
OT-based generation. The shares of the product $\langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1$ can also be computed using OTs. We first compute the shares of the product $\langle a_{ij} \cdot b_j \rangle$ for all $i = 1, \ldots, |B|$ and $j = 1, \ldots, d$. To do so, $\mathcal{S}_1$ uses each bit of b_j to select two values computed from a_{ij} using correlated OTs. In particular, for $k = 1, \ldots, l$, $\mathcal{S}_0$ sets the correlation function of COT to $f_k(x) = a_{i,j} \cdot 2^k + x \mod 2^l$ and $\mathcal{S}_0, \mathcal{S}_1$ run $\mathsf{COT}(r_k, f_k(x); b_j[k])$. If $b_j[k] = 0$, $\mathcal{S}_1$ gets r_k; if $b_j[k] = 1$, $\mathcal{S}_1$ gets $a_{i,j} \cdot 2^k + r_k \mod 2^l$. This is equivalent to $b_j[k] \cdot a_{ij} \cdot 2^k + r_k \mod 2^l$. Finally, $\mathcal{S}_1$ sets $\langle a_{ij} \cdot b_j \rangle_1 = \sum_{k=1}^{l}(b_j[k] \cdot a_{ij} \cdot 2^k + r_k) = a_{ij} \cdot b_j + \sum_{k=1}^{l} r_k \mod 2^l$, and $\mathcal{S}_0$ sets $\langle a_{ij} \cdot b_j \rangle_0 = \sum_{k=1}^{l}(-r_k) \mod 2^l$.

2.2 Privacy Preserving Logistic Regression

Secure computation friendly activation functions

the main additional challenge is to compute the logistic function $f(u) = \frac{1}{1+e^{-u}}$ on shared numbers.

注：Polynomials多项式法: 在函数上取n个点（n阶）逼近原函数。



The privacy preserving protocol

The only difference between the SGD for logistic regression and linear regression is the application of an extra activation function in each forward propagation.

Therefore, following the same protocol for privacy preserving linear regression, after computing the inner product of the input data and the coefficient vector, we switch the arithmetic sharing to a Yao sharing and evaluate the activation function using a garbled circuit. Then, we switch back to arithmetic sharing and continue the backward propagation.

2.3 Privacy Preserving Neural Network Training

We can use the **RELU** function as the activation function in each neuron and the **cross entropy** function as the cost function.

All the functions in both forward and backward propagation, other than evaluating the activation function and its partial derivative, **involve only simple additions and multiplications**, and are implemented using the same techniques discussed for linear regression.

- **A secure computation friendly alternative to the softmax function** $f(u_i) = \frac{e^{-u_i}}{\sum_{i=1}^{d_m} e^{-u_i}}$
- We first replace the exponentiations in the numerator with RELU functions such that the results remain non-negative as intended by e^{-u_i}.
- we compute the total sum by adding the outputs of all RELU functions, and divide each output by the total sum using a division garbled circuit.

3 Contributions

1. 线性回归，逻辑回归，神经网络 for **privacy-preserving**

- 综合使用secret sharing + arithmetic with precomputed triplets(三元组代数运算)+ garbled circuit

2. 性能提升 System

- Faster
- Scale to larger datasets

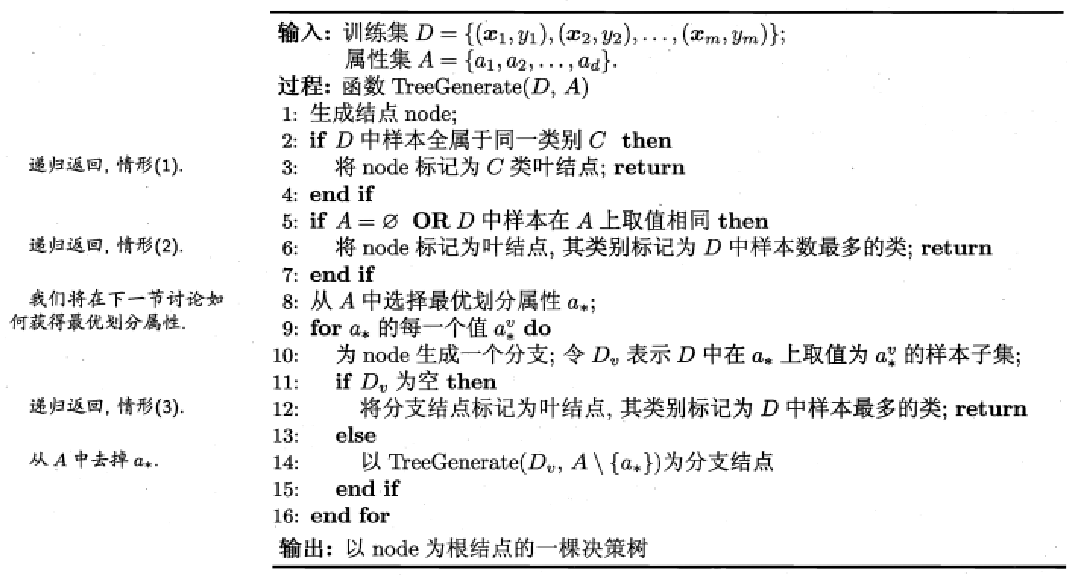
Specificly:

- Decimal multiplication in integer fields
- Secure-computation-friendly activation function
- Vectorization (mini-batch SGD)
- Privacy-preserving neutral networks

Week 4

DecisionTree

- 基本算法步骤



```
输入: 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
      属性集 $A = \{a_1, a_2, \ldots, a_d\}$.
过程: 函数 TreeGenerate($D$, $A$)
1:  生成结点 node;
2:  if $D$ 中样本全属于同一类别 $C$ then
3:      将 node 标记为 $C$ 类叶结点; return                         递归返回, 情形(1).
4:  end if
5:  if $A = \varnothing$ OR $D$ 中样本在 $A$ 上取值相同 then
6:      将 node 标记为叶结点, 其类别标记为 $D$ 中样本数最多的类; return   递归返回, 情形(2).
7:  end if
8:  从 $A$ 中选择最优划分属性 $a_*$;                                  我们将在下一节讨论如何获得最优划分属性.
9:  for $a_*$ 的每一个值 $a_*^v$ do
10:     为 node 生成一个分支; 令 $D_v$ 表示 $D$ 中在 $a_*$ 上取值为 $a_*^v$ 的样本子集;
11:     if $D_v$ 为空 then
12:         将分支结点标记为叶结点, 其类别标记为 $D$ 中样本最多的类; return   递归返回, 情形(3).
13:     else
14:         以 TreeGenerate($D_v$, $A \setminus \{a_*\}$)为分支结点            从 $A$ 中去掉 $a_*$.
15:     end if
16: end for
输出: 以 node 为根结点的一棵决策树
```

图 4.2 决策树学习基本算法

- 划分准则

  1 信息增益 (ID3：分类)

  信息熵 information entropy：Ent(D)越小,D的纯度越高

$$\operatorname{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k \ .$$

  信息增益 information gain： Gain（D,a）

  属性a有V个可能的取值$\{a^1,\ldots,a^V\}$,使用a来对样本集D进行划分，则会产生V个分支结点，其中第v个分支节点包含了D中所有在属性a上取值为a^V的样本，记为D^V.

$$\operatorname{Gain}(D, a) = \operatorname{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \operatorname{Ent}(D^v) \ .$$

  样本数越多分支结点的影响越大（权重）

  Gain越大，使用属性a来进行划分所获得的纯度提升越大。

第八步中选择属性：$a_* = \underset{a \in A}{\arg\max}\, \mathrm{Gain}(D, a)$.（ID3决策树）

例：D中所有样例：正例占p_1=8/17,反例p_2=9/17,计算得出信息熵

$$\mathrm{Ent}(D) = -\sum_{k=1}^{2} p_k \log_2 p_k = -\left(\frac{8}{17}\log_2\frac{8}{17} + \frac{9}{17}\log_2\frac{9}{17}\right) = 0.998\ .$$

设用属性a进行划分，则得到3个子集，分别记为D^1,D^2,D^3,

子集 D^1 包含编号为 {1, 4, 6, 10, 13, 17} 的 6 个样例, 其中正例占 $p_1 = \frac{3}{6}$, 反例占 $p_2 = \frac{3}{6}$; D^2 包含编号为 {2, 3, 7, 8, 9, 15} 的 6 个样例, 其中正、反例分别占 $p_1 = \frac{4}{6}$, $p_2 = \frac{2}{6}$; D^3 包含编号为 {5, 11, 12, 14, 16} 的 5 个样例, 其中正、反例分别占 $p_1 = \frac{1}{5}$, $p_2 = \frac{4}{5}$. 根据式(4.1)可计算出用“色泽”划分之后所获得的 3 个分支结点的信息熵为

$$\mathrm{Ent}(D^1) = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1.000\ ,$$

$$\mathrm{Ent}(D^2) = -\left(\frac{4}{6}\log_2\frac{4}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) = 0.918\ ,$$

$$\mathrm{Ent}(D^3) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right) = 0.722\ ,$$

$$\begin{aligned}\mathrm{Gain}(D, \text{色泽}) &= \mathrm{Ent}(D) - \sum_{v=1}^{3}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v) \\ &= 0.998 - \left(\frac{6}{17}\times 1.000 + \frac{6}{17}\times 0.918 + \frac{5}{17}\times 0.722\right) \\ &= 0.109\ .\end{aligned}$$

依次计算出其他属性的Gain，取最大值。

2 增益率（C4.5：分类）

因为信息增益准则对可取值数目较多（例如编号）的属性有所偏好，为减少不利影响（决策树不具有泛化能力），采用增益率（gain ratio）来选择最优划分属性。（C4.5决策树算法）

- 定义式

$$\mathrm{Gain\_ratio}(D, a) = \frac{\mathrm{Gain}(D, a)}{\mathrm{IV}(a)}\ ,$$

其中

$$\mathrm{IV}(a) = -\sum_{v=1}^{V}\frac{|D^v|}{|D|}\log_2\frac{|D^v|}{|D|}$$

IV(a)为属性a的固有值，a的可能取值数目越多（V越大），IV(a)越大。

- 增益率准则对可取值数目较少的属性有所偏好，C4.5中使用启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

3 基尼指数（CART：分类和回归都能用）

基尼值：

$$\mathrm{Gini}(D)=\sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k}p_k p_{k'}$$
$$=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2 .$$

- Gini(D)反映从数据集D中随机抽取两个样本其类别标记不一致的概率。
- Gini(D)越小，D的纯度越高
- $$\mathrm{Gini\_index}(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(D^v) .$$
- 第八步中的选择最优属性可改为：$a_*=\underset{a\in A}{\arg\min}\ \mathrm{Gini\_index}(D,a).$

4 算法比较

1. ID3
   是最早提出的一种决策树方法，使用上述信息增益的方式建立。 缺点是只能处理离散型属性，并且对倾向于选择取值较多的属性
2. C4.5
   使用增益率对信息增益进行扩充，以解决偏向取值较多的属性的问题。另外它可以处理连续型属性。
3. CART
   CART中用于选择变量的不纯性度量是Gini指数

CART与ID3和C4.5的区别

1. CART树是二叉树，而ID3和C4.5可以是多叉树
2. CART在生成子树时，是选择一个特征一个取值作为切分点，生成两个子树
3. 选择特征和切分点的依据是基尼指数，选择基尼指数最小的特征及切分点生成子树

- 剪枝处理

  防止过拟合

  prepruning　postpruning

  - 判断依据：该划分是否可以使决策树泛化能力提升
  - 如何判断泛化性能是否提升？：划分验证集，
    - pre：若增加该枝节在验证集上正确率上升，则选择该划分。（因为基于“贪心”本质禁止某些在后续分支可能带来泛化能力提升的分支展开，带来欠拟合风险）
    - post：若减去该枝节正确率上升，则剪去。（欠拟合风险减低，开销上升）
- 应用

  · Evaluation of brand expansion opportunities for a business using historical sales data

· Determination of likely buyers of a product using demographic data to enable targeting of limited advertisement budget

· Prediction of likelihood of default for applicant borrowers using predictive models generated from historical data

· Help with prioritization of emergency room patient treatment using a predictive model based on factors such as age, blood pressure, gender, location and severity of pain, and other measurements

· Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

单棵决策树最大的优点在于，它可以被很轻松的可视化甚至是提取规则分类规则。而集成学习很难做到。

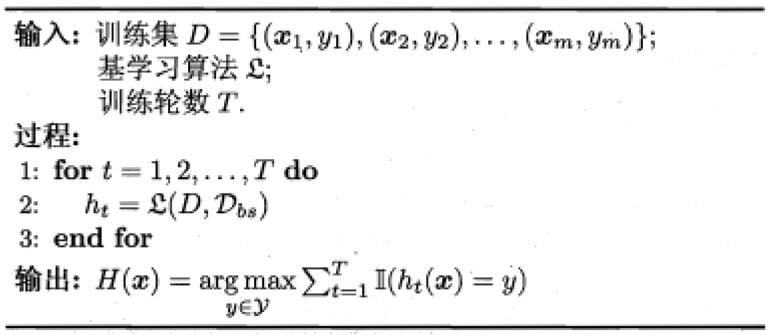
Random Forest

Bagging + 决策树 = 随机森林

随机森林顾名思义，使用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的死后，就让森林的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类能被选择最多，就预测这个样本为那一类。

- 集成学习方法
  - 为了保证结合后获得好的集成，个体学习器应“好而不同”（准确性和多样性，然而这两者从根源上来说是矛盾的，本来这些模型就是为了解决同一问题而生成，不是独立的）
  - 集成学习方法
    - 个体学习器之间存在强依赖关系，必须串行生成的序列化方法（Boosting）
    - 个体学习器之间不存在强依赖关系，可同时生成的序列化方法（Bagging，Random Forest）
- 构建过程

  Bagging：（基于bootstrap sampling）给定m个样本的数据集，先随机取出一个样本放入采样集中，再放回初始数据集（使得下一次采样时该样本仍有可能被选中），m次后得到含m个样本的采样集。然后采样出T个含m个训练样本的采样集，然后基于每个采样集训练出一个基学习器。



```
输入: 训练集 $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$;
      基学习算法 $\mathfrak{L}$;
      训练轮数 $T$.
过程:
1: for $t = 1, 2, \ldots, T$ do
2:     $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: end for
输出: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y)$
```

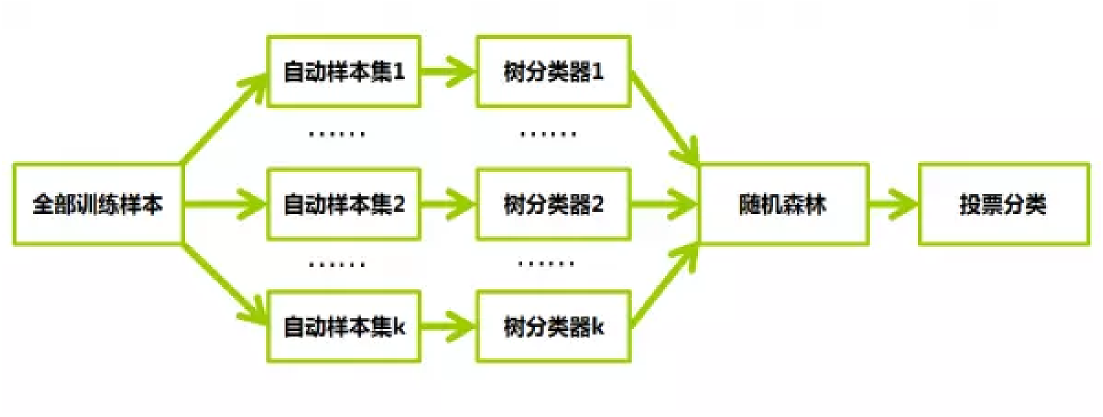
图 8.5 Bagging 算法

以上称为自助采样过程。

注意到：每个采样集中实际上只使用了63.2%的样本，剩下36.8%(bootstrapping: 样本在m次采样中始终不被采样到的概率是0.368)可以用于验证泛化能力（out-of-bag estimate包外估计）

- 预测时，分类任务：简单投票法；回归任务：简单平均法。

RF在以决策树为基学习器构建Bagging的集成的基础上，进一步在决策树的训练过程中引入了**随机属性选择**。



对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含k个属性的子集，然后再从这个子集中选择一个最优属性用于划分。

- Bagging使用的是确定性决策树，而RF使用的是随机性决策树。

RF的构建过程

- 从原始训练集中使用Bootstrapping方法随机有放回采样选出m个样本，共进行T次采样，生成T个训练集
- 对于T个训练集，我们分别训练T个决策树模型
- 对于单个决策树模型，假设训练样本特征的个数为n，那么每次分裂时根据信息增益/信息增益比/基尼指数选择最好的特征进行分裂
- 每棵树都一直这样分裂下去，直到该节点的所有训练样例都属于同一类。在决策树的分裂过程中不需要剪枝
- 将生成的多棵决策树组成随机森林。对于分类问题，按多棵树分类器投票决定最终分类结果；对于回归问题，由多棵树预测值的均值决定最终预测结果

- Decision Tree, RF,和神经网络的比较

  https://www.zhihu.com/question/68130282

Week 5

DecisionTree Datasets

- **Iris Plants:**
  - 下载：https://archive.ics.uci.edu/ml/datasets/iris

    或可直接从Sklearn中下载

    ```
    from sklearn.datasets import load_iris
    ```

    https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

    各种格式iris数据集

    https://gist.github.com/curran/a08a1080b88344b0c8a7

  - 数据格式：4个特征变量(SepalLength, SepalWidth, PetalLength, PetalWidth)，1个类别变量(0,1,2)，共有150个样本，无缺失值。iris属性：iris.data 150*4 iris.target 3 (0,1,2各50个).

- **Wine**
  - 下载：https://archive.ics.uci.edu/ml/datasets/Wine
  - 数据格式：12个特征变量，一个类别变量(1,2,3)，共178个样本，无缺失值。
  - 数据分析：https://github.com/ajeenkkya/Decision-tree-wine/blob/master/wine.ipynb

- **Adult Income**
  - 数据格式：14个属性，48842个样本，有缺失值。
  - 数据分析：https://github.com/saravrajavelu/Adult-Income-Analysis/blob/master/Adult_Income_Analysis.ipynb

    内含decision tree和random forest 的实现和数据分析

- **Heart Disease**（补）
  - 下载：http://archive.ics.uci.edu/ml/datasets/Heart+Disease
  - 数据格式：75个特征变量，303个样本，有缺失值。
  - 数据分析：https://blog.csdn.net/iizhuzhu/article/details/89067386

Week 6

Decision Tree C++实现Python中的DecisionTreeClassifier()

- Reference: https://github.com/bonzo/Decision-Tree
- 代码基本结构：header.h functions.cpp DecisionTree.cpp
- 输入数据格式：

属性1	属性2	属性n
属性值1.1	属性值2.1	属性值n.1
属性值1.2	属性值2.2	属性值n.2
属性值1.3	属性值2.3	属性值n.3

txt, csv, dat....均可（用“，”隔开）

- 建树逻辑同上伪代码

主要思想就是建立一个二维数组把表格中的每一行看作一个vector存入，但是内存容易爆炸，后续改进。（见注释）