

---

# **CSci 5541: Natural Language Processing**

## ***Random Variable Template Library (Session 8)***

William Schuler

# RVTL

---

RVTL includes for random variables, models:

```
#include "nl-modelfile.h"  
#include "nl-randvar.h"  
#include "nl-prob.h"  
#include "nl-cpt.h"
```

No additional includes for joint RVs

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

```
// sample use...
```

```
Joint2DRV<X,Y> xy;
```

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

```
// sample use...
```

```
Joint2DRV<X,Y> xy;
```

```
// member functions to access component variables...
```

```
X& Joint2DRV<X,Y>::setSub1();
```

```
Y& Joint2DRV<X,Y>::setSub2();
```

```
const X& Joint2DRV<X,Y>::getSub1() const;
```

```
const Y& Joint2DRV<X,Y>::getSub2() const;
```

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

- Delimited joint RV – read/write segments components:

```
class DelimitedJoint2DRV<ps1,X,ps2,Y,ps3>  
: public Joint2DRV<X,Y>;
```

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

- Delimited joint RV – read/write segments components:

```
class DelimitedJoint2DRV<ps1,X,ps2,Y,ps3>  
    : public Joint2DRV<X,Y>;
```

```
// sample use...
```

```
char* psN = ""; char* psUsc = "_";
```

```
typedef DelimitedJoint2DRV<psN,X,psUsc,Y,psN> XY;
```

# Joint RV Class

---

Programs may use the following RVTL classes/templates (in red):

- Joint random variable template:

```
class Joint2DRV<X,Y>;
```

- Delimited joint RV – read/write segments components:

```
class DelimitedJoint2DRV<ps1,X,ps2,Y,ps3>  
    : public Joint2DRV<X,Y>;
```

```
// sample use...
```

```
char* psN = ""; char* psUsc = "_";
```

```
typedef DelimitedJoint2DRV<psN,X,psUsc,Y,psN> XY;
```

```
// nested (read: outer delim not substr of inner)...
```

```
char* psUscUsc = "__";
```

```
class DelimitedJoint2DRV<psN,XY,psUscUsc,Z,psN> xyz;
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
CPT2DModel<X,W,LogProb> modXgivW;
CPT2DModel<Y,X,LogProb> modYgivX;
typedef Modeled2DRV<typeof(modXgivW),modXgivW> ModeledX;
typedef Modeled2DRV<typeof(modYgivX),modYgivX> ModeledY;
typedef Joint2DRV<X,Y> XY
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
}
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
    bool setNext ( XY& xy, const W& w ) const
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
    bool setNext ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( ( y.setNext(x) ) ||
                 ( x.setNext(w) && y.setFirst(x) ) ); }
}
```

(iterate RVs like bits in counter: iterate low order bit, then high order bit)

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
    bool setNext ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( ( y.setNext(x) ) ||
                 ( x.setNext(w) && y.setFirst(x) ) ); }
    bool getProb ( const XY& xy, const W& w ) const
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
    bool setNext ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( ( y.setNext(x) ) ||
                 ( x.setNext(w) && y.setFirst(x) ) ); }
    bool getProb ( const XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.getProb(w) * y.getProb(x) ); }
```

# Joint Model Class (define it yourself)

---

Joint model using joint RVs:

```
// Define XY model w. components ModeledX, ModeledY...
XYModel : public Generic2DModel<XY,W> {
    bool setFirst ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.setFirst(w) && y.setFirst(x) ); }
    bool setNext ( XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( ( y.setNext(x) ) ||
                 ( x.setNext(w) && y.setFirst(x) ) ); }
    bool getProb ( const XY& xy, const W& w ) const {
        ModeledX x=xy.getSub1(); ModeledY y=xy.getSub2();
        return ( x.getProb(w) * y.getProb(x) ); }
};
```