Feature Selection and Independence

- Feature Selection is a crucial step in the Data Mining process
- Statistical Independence turns out to be a desirable property of a decent feature set
- Eliminate redundancy in the information that we give to our classifier
- Parsimonious with our computing resources

$$P(B|A) = P(B) \tag{1}$$

$$P(B \wedge A) = P(B)P(A) \tag{2}$$

Random Variables

- We denoted by $P(A)$ the probability that some abstract event "A" is true
- In practical terms what is more useful to associate real numbers to the events in the sample space
- This results in the concept of a *random variable*
- A random variable is *discrete* if it can take countably many values and is *continuous* otherwise

Mass and Density Functions

- Random Variables are completely specified by their probability mass functions (discrete) or density functions (continuous)
- Discrete Case

$$p(x) = P(X = x) \tag{3}$$

- Continuous Case

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx \tag{4}$$

$$p(x, y),\ f_{X_1, X_2, \ldots}(x_1, x_2, \ldots) \tag{5}$$

Statistical Independence from the Joint Density

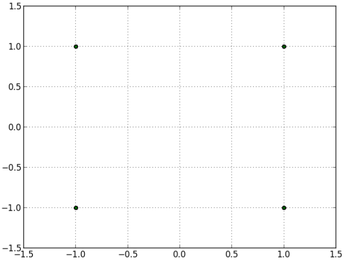
- The joint density is a complete statistical description of a set of random variables
- The condition for statistical independence is that the density or mass function factors

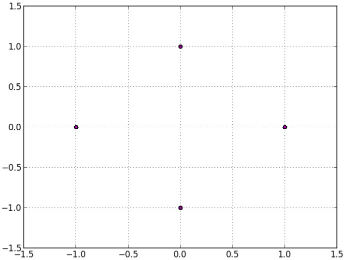
$$p(x, y) = p(x)p(y) \tag{6}$$

$$f_{X_1,X_2,\ldots}(x_1, x_2, \ldots) = f_{X_1}(x_1)f_{X_2}(x_2)\ldots \tag{7}$$

Marginal Distribution

- The marginal distribution of a random variable $f_X(x_i)$ is obtained by integrating or summing with respect to the other variables in some joint density
- Can you use the formulas we've discussed up to this point to determine the dependence of these distributions?





Contingency Table

- From the description in the previous slide we know that our variables are...
- discrete: take on countably many values
- $x, y \in -1, 0, 1$
- This means that our joint probability mass function will assign probabilities to the 9 possible pairs of values from the range
- $p(-1, -1), p(0, -1), (0, 1), \ldots, p(1, 1)$
- If we observe this from actual samples of a random variable then this is called a contingency table

Worked Example

- Take a moment to attempt to work this out...
- Think about what you need to do to test joint density for independence
- What are the values of the probability mass functions

Answer and Conclusion

- Answer: The density is independent in the first case and dependent in the second case
- What conclusions can we draw from this?
- Joint density is a *complete* description of some set of random variables, we can calculate anything we'd want to know from it
- In a practical setting we rarely have such a complete description, consider the complexity of describing the joint density as the number of arguments is increased
- Statistical Independence is an important property but it can be encoded in the data in non-obvious ways

Why is this important for Data-Mining?

- In Classification tasks we are attempting to learn some function $f(x)$ that is defined over the space of possible data instances
- Ideally, $f(x)$ maps the data instance to the correct class
- In a majority situations we don't arrive at a closed form solution for $f(x)$, it is optimized iteratively with respect to some criteria (objective function)
- Because there may be many local minima the algorithms that we use can be affected in sometimes unpredictable ways
- As a rule of thumb it is often desirable to eliminate redundancy in our feature set

Other Concepts of Dependency

- Linear Dependencies in random variables are easier to detect than general statistical independence
- For this we first need to talk about expectation $E[X]$
- $E[X]$ can be thought of in two ways
- The arithmetic mean of a large number of observations of X
- As a quantity defined from the probability distribution

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx \tag{8}$$

Defining Other Quantities

- Many statistics about a random variable are expressed in terms of expectations: moments, covariance, variance, correlation
- We can use the expectation for an arbitrary formula $g(X)$

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx \tag{9}$$

Measures of dispersion

- Dispersion measures how much a r.v. spreads away from its typical or *central* value
- The expectation is also called the *mean* μ and is a measure of central tendency
- The variance describes the width of the distribution around μ

$$\sigma^2 = E[(X - \mu)^2] \tag{10}$$

- In Data-Mining this quantity is useful because we may need to standardize or data or features before applying a learning algorithm, computing the variance allows us to *normalize* features

Measures of Dependence

- Using expectation we can also define quantities that characterize how likely two variables are to change together
- For this we use the *covariance*

$$\sigma_{xy}^2 = E[(X - \mu_x)(Y - \mu_y)] \tag{11}$$

- In the data mining context we will often want to compute the variance or standard deviation because we may want to *normalize* and center the values of all of our features prior to learning
- Cross-tabulation or contingency table is infeasible for a continuous r.v.

Two Dimensional Cases

- When we have more than one random variable we can define an entire *covariance matrix*
- Often denoted as R or Σ
- Properties: symmetric $i \leftrightarrow j$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_M \end{bmatrix} \tag{12}$$

$$[\mathbf{\Sigma}] = \sigma_{ij}^2 = E[(X_i - \mu_{x_i})(X_j - \mu_{x_j})] \tag{13}$$

Verifying our Results with Code

- Without the ability to program Data Science is moot
- We need some environment where we can compute statistics, explore the data with simple plots or visualizations, apply different learning algorithms
- Typical solution for this would be either a scripting language or a domain specific language such as Matlab, Octave, Mathematica, R, or SPSS
- As mentioned in the previous class we will use Python for mainline work

Python for Data Analysis

- Python with one of its shell environments and some libraries basically replicate what is available from Matlab
- Advantages: general purpose language, not tied to proprietary environment, code will be more portable, numerous bindings API for other frameworks
- `numpy`: basic array data type
- `scipy`: scientific computing
- `matplotlib`: reproduces the style of plotting from Matlab, can be run interactively
- `pandas`: importing data

Code for the Example

- Importing packages and calling into them

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats

g = np.random.random_integers(0,3,100)
```

Working with arrays

- We also have more convenient Matlab style array indexing

```
s1 = np.array([[-1,1],[1,-1],[1,1],[-1,-1]])
s2 = np.array([[0,1],[0,-1],[1,0],[-1,0]])

x1 = s1[g,0]
y1 = s1[g,1]

H1,xedges,yedges = np.histogram2d(x1,y1,bins=[-1.5, 0., 1.5])

x2 = s2[g,0]
y2 = s2[g,1]

H2,xedges,yedges = np.histogram2d(x2,y2,bins=[-1.5,-0.5,0.5,1.5])
```

Accessing Library Routines

- These lines perform a χ^2 test on our contingency table data
- A χ^2 test is a goodness-of-fit test of data to probability distributions

```
chi2, p, dof, ex = scipy.stats.chi2_contingency(H1)
print p

chi2, p, dof, ex = scipy.stats.chi2_contingency(H2)
print p
```

Simple Visualizations

- `matplotlib.pyplot` gives us a state machine interface for creating graphics that allows us to manipulate markers, colors, add grids, ticks, label axes, and add titles
- also enable this to run interactively through the shell

```
plt.figure(1)
plt.scatter(x1,y1,20,'g')
plt.grid()

plt.figure(2)
plt.scatter(x2,y2,20,'m')
# plt.plot(g)
# plt.plot(x1,'k')
plt.grid()

plt.show()
```

Useful Python Linear Algebra Commands

```
// generate numbers according to normal dist
np.random.normal(size=(N))
// generate rand num with uniform dist
np.random.uniform(size=(N))
// matrix of zeros
np.zeros((N,2))
// element wise mult.
np.multiply
// form numpy array
S = np.array([[1/6.0, 0],[0, 1/3.0]])
// matrix multiplication
B = np.dot(A, np.dot(S,G))
// least squares
h = np.linalg.lstsq(X,y)[0]
// generate grids
xx,yy = np.meshgrid(np.linspace(-6,6,100),np.linspace(-6,6,100))
// concatenation
np.hstack((a,b))
np.vstack((a,b))
np.newaxis
// x[:,1,np.newaxis]
```

- Some useful commands for performing linear algebra calculations

Linear Algebra

- Matrix Algebra: multiply matrices together with the "row dot column" rule
- Is defined as long as the dimensions match up
- *transposition*: Exchange rows and columns

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_M \end{bmatrix} = [x_1, x_2, x_3, \ldots, x_M]^T \tag{14}$$

$$\mathbf{x}^T\mathbf{x} = [x_1, x_2, x_3, \ldots, x_M] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_M \end{bmatrix} = x_1^2 + x_2^2 + \ldots \tag{15}$$

Inner Products

- An expression such as $\mathbf{x}^T\mathbf{y}$ is what we call the *inner product* of the vector space
- A comparison operation with substantial geometric meaning
- Often written with angle brackets $\langle x, y \rangle = \mathbf{x}^T\mathbf{y}$
- Square root of the inner product of a vector with itself is called the *norm* and is the length of that particular vector $||\mathbf{x}|| = \sqrt{\mathbf{x}^T\mathbf{x}}$
- *Orthogonality*: Two vectors are orthogonal when their inner product $\mathbf{x}^T\mathbf{y} = 0$

Matrices

- Matrix: 2 Dimensional array of numbers $A = [a_{i,j}]$
- A matrix multiplies a vector and returns a vector $\mathbf{y} = A\mathbf{x}$

$$y_i = \sum_{j=1}^{M} a_{ij} x_j \tag{16}$$

- Other properties: determinant $|A|$, trace (sum of the diagonal elements), and inverse $AA^{-1} = I$

More Linear Algebra Review cont.

- If we are given an entire data matrix $\mathbf{X}$ then we simultaneously express projection of each of the data instances onto a vector v with a simple matrix multiplication

$$\mathbf{X}v = \begin{bmatrix} x_1 v \\ x_2 v \\ x_3 v \\ \vdots \\ x_n v \end{bmatrix} \tag{17}$$