2.2 Multinominal Variables

August 20, 2016

0.0.1 Derivation complement of formula 2.35

The normalization coefficient of multinominal distribution is the number of ways to partitioning N objects int oK groups of size $m_1, \cdots, m_k$

We can do this partition sequentially.Firstly, we can choose m_1 objects for the first group.There are $\binom{N}{m_1}$ ways to perform this.Then we should choose anothor m_2 objects from the rest,there are $\binom{N-m_1}{m_2}$ ways to do this. Similarly, we have $\binom{N-(m_1+m_2)}{m_3}$ choices for the third group.

The rest can be done in the same manner, so we have

$$\begin{aligned}
&\binom{N}{m_1}\binom{N-m_1}{m_2}\binom{N-(m_1+m_2)}{m_3}\cdots\binom{N-(m_1+m_2+\cdots+m_{k-1})}{m_k} \\
=&\frac{N!}{m_1!(N-m_1)!}\cdot\frac{(N-m_1)!}{m_2![N-(m_1+m_2)]!}\cdot\frac{[N-(m_1+m_2)]!}{m_3![N-(m_1+m_2+m_3)]!}\cdots\frac{[N-(m_1+m_2+\cdots+m_{k-1})]!}{m_k![N-(m_1+m_2+\cdots m_k)]!}
\end{aligned}$$

Note that $N = m_1 + m_2 + \cdots + m_k$.By simplifying the equation above, we can draw the conclusion that there are $\binom{N}{m_1 m_2 \cdots m_K}$ ways to partitioning N objects into K goups of size $m_1, \cdots, m_k$

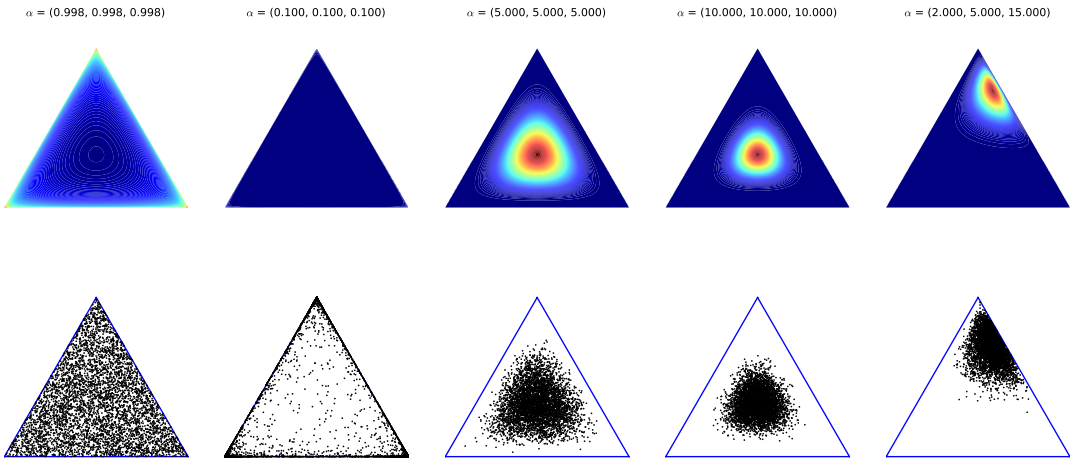
0.0.2 Generalization of Figure 2.5

The plots of the dirichlet distribution in PRML is a 2 dimension version, here we plot a 3 dimension version by drawing contours of dirichlet distribution, thanks to the selfless contribution of Thomas

In [1]:
```
'''Functions for drawing contours of Dirichlet distributions.'''

# Author: Thomas Boggs

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.tri as tri
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf')

_corners = np.array([[0, 0], [1, 0], [0.5, 0.75**0.5]])
_triangle = tri.Triangulation(_corners[:, 0], _corners[:, 1])
_midpoints = [(_corners[(i + 1) % 3] + _corners[(i + 2) % 3]) / 2.0 \
              for i in range(3)]
```

```
def xy2bc(xy, tol=1.e-3):
    '''Converts 2D Cartesian coordinates to barycentric.
    Arguments:
        `xy`: A length-2 sequence containing the x and y value.
    '''
    s = [(_corners[i] - _midpoints[i]).dot(xy - _midpoints[i]) / 0.75 \
         for i in range(3)]
    return np.clip(s, tol, 1.0 - tol)

class Dirichlet(object):
    def __init__(self, alpha):
        '''Creates Dirichlet distribution with parameter `alpha`.'''
        from math import gamma
        from operator import mul
        self._alpha = np.array(alpha)
        self._coef = gamma(np.sum(self._alpha)) / \
                     reduce(mul, [gamma(a) for a in self._alpha])
    def pdf(self, x):
        '''Returns pdf value for `x`.'''
        from operator import mul
        return self._coef * reduce(mul, [xx ** (aa - 1)
                                         for (xx, aa)in zip(x, self._alpha)
    def sample(self, N):
        '''Generates a random sample of size `N`.'''
        return np.random.dirichlet(self._alpha, N)

def draw_pdf_contours(dist, border=False, nlevels=200, subdiv=8, **kwargs):
    '''Draws pdf contours over an equilateral triangle (2-simplex).
    Arguments:
        `dist`: A distribution instance with a `pdf` method.
        `border` (bool): If True, the simplex border is drawn.
        `nlevels` (int): Number of contours to draw.
        `subdiv` (int): Number of recursive mesh subdivisions to create.
        kwargs: Keyword args passed on to `plt.triplot`.
    '''
    from matplotlib import ticker, cm
    import math

    refiner = tri.UniformTriRefiner(_triangle)
    trimesh = refiner.refine_triangulation(subdiv=subdiv)
    pvals = [dist.pdf(xy2bc(xy)) for xy in zip(trimesh.x, trimesh.y)]

    plt.tricontourf(trimesh, pvals, nlevels, **kwargs)
    plt.axis('equal')
    plt.xlim(0, 1)
    plt.ylim(0, 0.75**0.5)
    plt.axis('off')
    if border is True:
```

```
            plt.hold(1)
            plt.triplot(_triangle, linewidth=1)

    def plot_points(X, barycentric=True, border=True, **kwargs):
        '''Plots a set of points in the simplex.
        Arguments:
            `X` (ndarray): A 2xN array (if in Cartesian coords) or 3xN array
                           (if in barycentric coords) of points to plot.
            `barycentric` (bool): Indicates if `X` is in barycentric coords.
            `border` (bool): If True, the simplex border is drawn.
            kwargs: Keyword args passed on to `plt.plot`.
        '''
        if barycentric is True:
            X = X.dot(_corners)
        plt.plot(X[:, 0], X[:, 1], 'k.', ms=1, **kwargs)
        plt.axis('equal')
        plt.xlim(0, 1)
        plt.ylim(0, 0.75**0.5)
        plt.axis('off')
        if border is True:
            plt.hold(1)
            plt.triplot(_triangle, linewidth=1)

    if __name__ == '__main__':
        f = plt.figure(figsize=(14, 6))
        alphas = [[0.998] * 3,
                  [0.1] * 3,
                  [5] * 3,
                  [10] * 3,
                  [2, 5, 15]]
        for (i, alpha) in enumerate(alphas):
            plt.subplot(2, len(alphas), i + 1)
            dist = Dirichlet(alpha)
            draw_pdf_contours(dist)
            title = r'$\alpha$ = (%.3f, %.3f, %.3f)' % tuple(alpha)
            plt.title(title, fontdict={'fontsize': 8})
            plt.subplot(2, len(alphas), i + 1 + len(alphas))
            plot_points(dist.sample(5000))
        plt.show()
```



The color scale runs from dark blue (lowest values) to red (highest values). From the figure above, we can see that when $\{\alpha_k\}$ are small values,say 0.1, most of the sample points locate at the margin while at the centre for large $\{\alpha_k\}$. As for the case of the binomial distribution with its beta prior, we can interpret the parameters α_k of the Dirichlet prior as an effective number of observation of $x_k = 1$.