In [1]:
```
from datascience import *
import numpy as np
import matplotlib
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
```

Classification

In [2]:
```
ckd = Table.read_table('ckd.csv').relabeled('Blood Glucose Random', 'Glucose')
ckd.show(3)
```

Age	Blood Pressure	Specific Gravity	Albumin	Sugar	Red Blood Cells	Pus Cell	Pus Cell clumps	Bacteria	Glucose	Blood Urea	Cr
48	70	1.005	4	0	normal	abnormal	present	notpresent	117	56	
53	90	1.02	2	0	abnormal	abnormal	present	notpresent	70	107	
63	70	1.01	3	0	abnormal	abnormal	present	notpresent	380	60	

... (155 rows omitted)

In [3]:
```
ckd.select('Glucose','White Blood Cell Count', 'Hemoglobin','Class').show(3)
```

Glucose	White Blood Cell Count	Hemoglobin	Class
117	6700	11.2	1
70	12100	9.5	1
380	4500	10.8	1

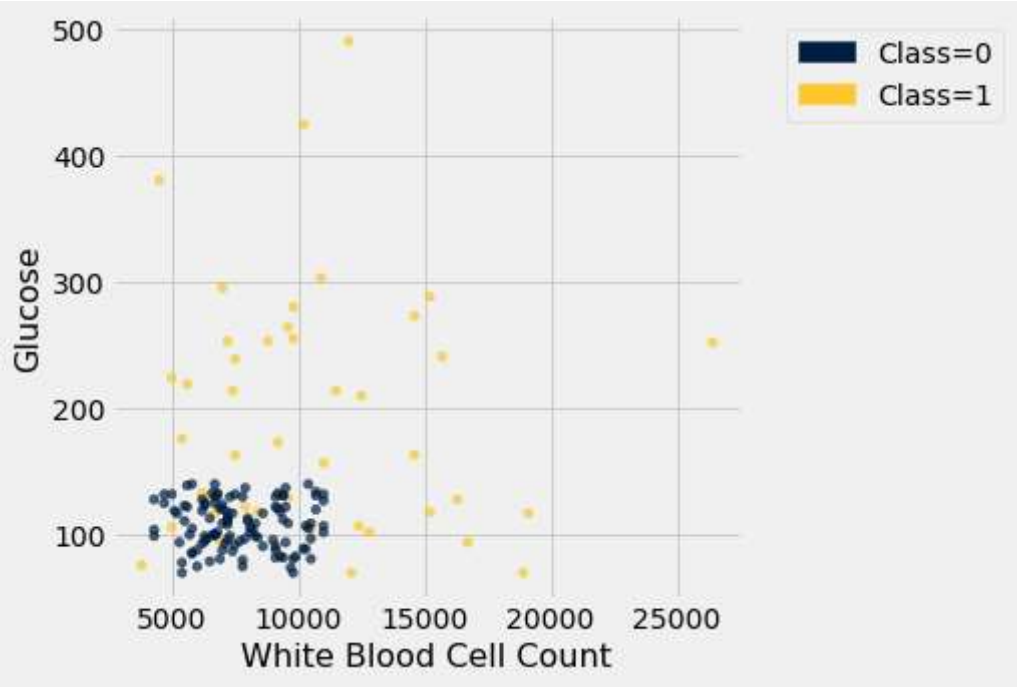
... (155 rows omitted)

In [4]:
```
ckd.group('Class')
```

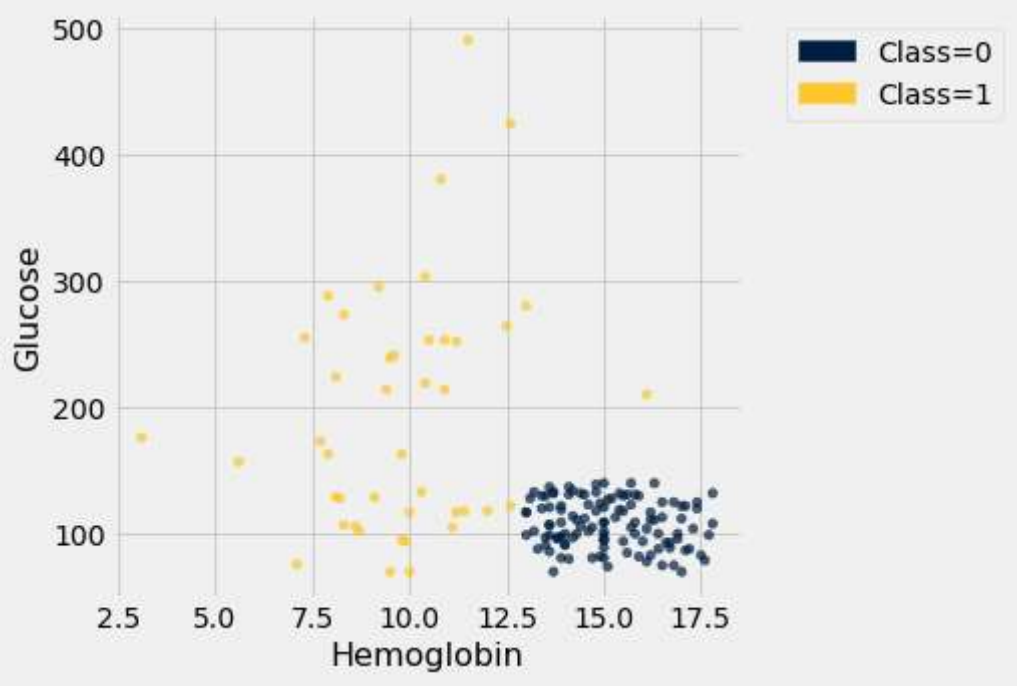
Out[4]:

Class	count
0	115
1	43

In [5]:
```
ckd.scatter('White Blood Cell Count', 'Glucose', group = 'Class')
```



In [6]:
```
ckd.scatter('Hemoglobin', 'Glucose', group = 'Class')
```



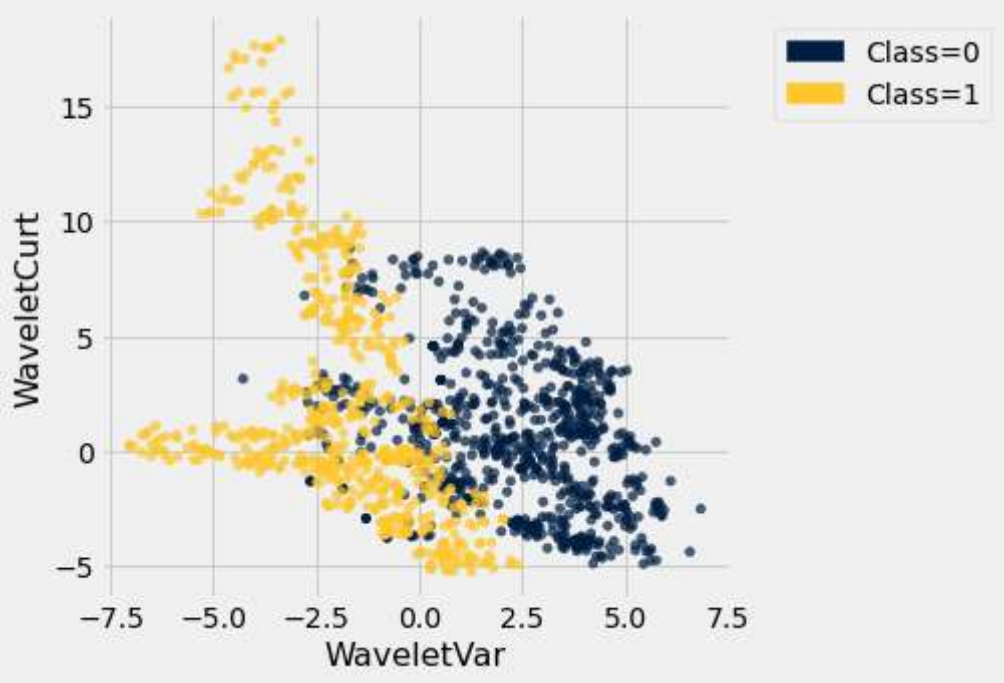
In [7]:
```
banknotes = Table.read_table('banknote.csv')
banknotes
```

Out[7]:

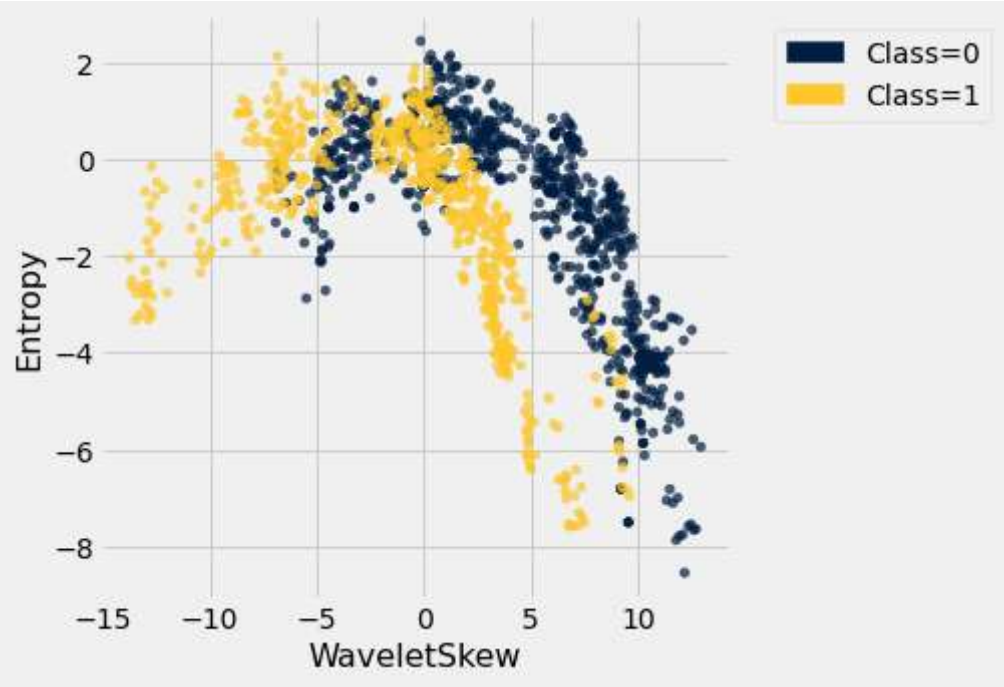
WaveletVar	WaveletSkew	WaveletCurt	Entropy	Class
3.6216	8.6661	-2.8073	-0.44699	0
4.5459	8.1674	-2.4586	-1.4621	0
3.866	-2.6383	1.9242	0.10645	0
3.4566	9.5228	-4.0112	-3.5944	0
0.32924	-4.4552	4.5718	-0.9888	0
4.3684	9.6718	-3.9606	-3.1625	0
3.5912	3.0129	0.72888	0.56421	0
2.0922	-6.81	8.4636	-0.60216	0
3.2032	5.7588	-0.75345	-0.61251	0
1.5356	9.1772	-2.2718	-0.73535	0

... (1362 rows omitted)

In [8]:
```
banknotes.scatter('WaveletVar', 'WaveletCurt', group = 'Class')
```

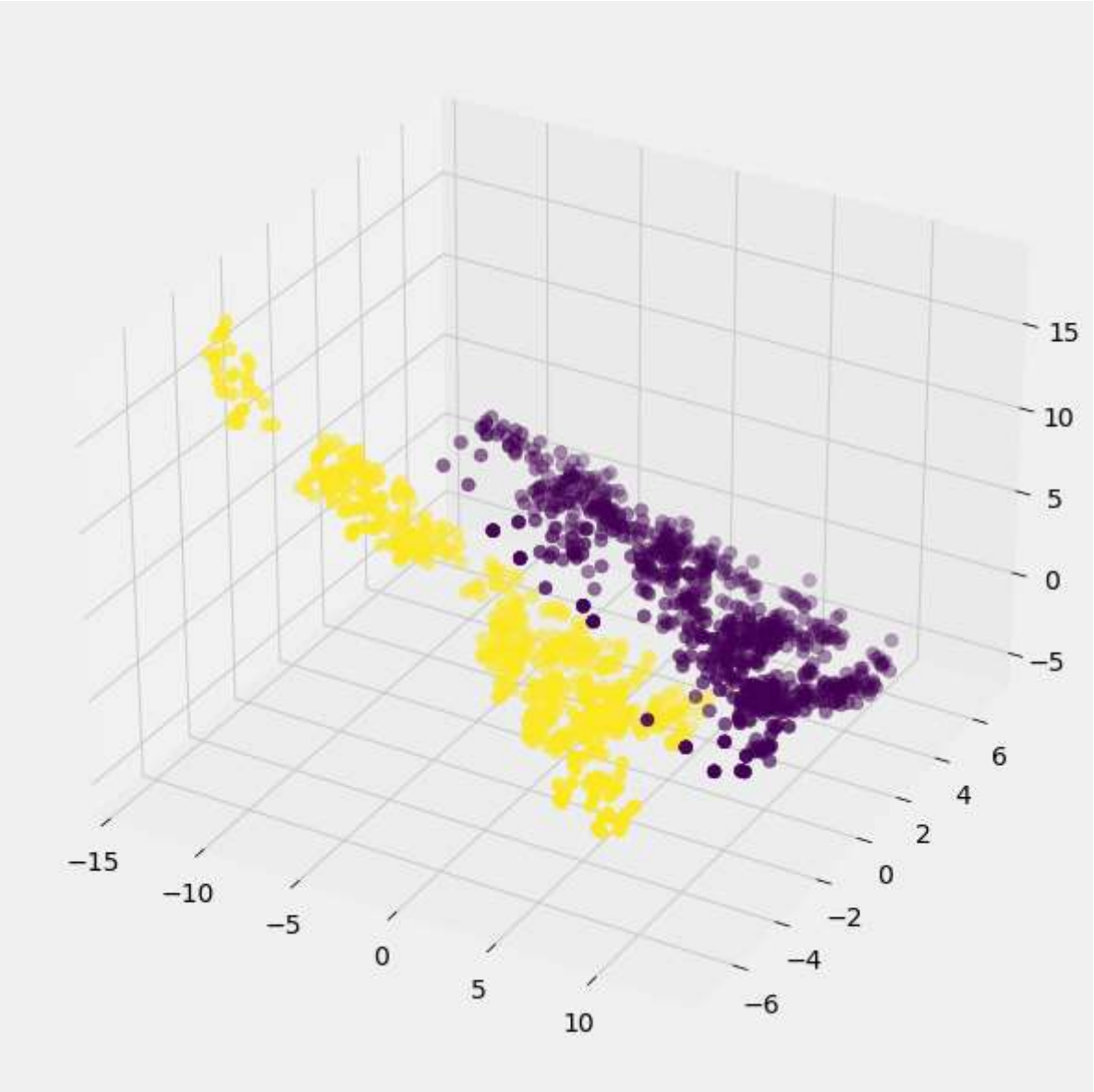


In [9]:
```
banknotes.scatter('WaveletSkew', 'Entropy', group = 'Class')
```



In [10]:
```
fig = plots.figure(figsize=(8,8))
ax = Axes3D(fig)
ax.scatter(banknotes.column('WaveletSkew'),
           banknotes.column('WaveletVar'),
           banknotes.column('WaveletCurt'),
           c=banknotes.column('Class'),
           cmap='viridis',
          s=50);
```

```
C:\Users\schoend\AppData\Local\Temp\ipykernel_19576\4103308493.py:2: MatplotlibDeprec
ationWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass t
he keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress thi
s warning. The default value of auto_add_to_figure will change to False in mpl3.5 and
True values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(fig)
```

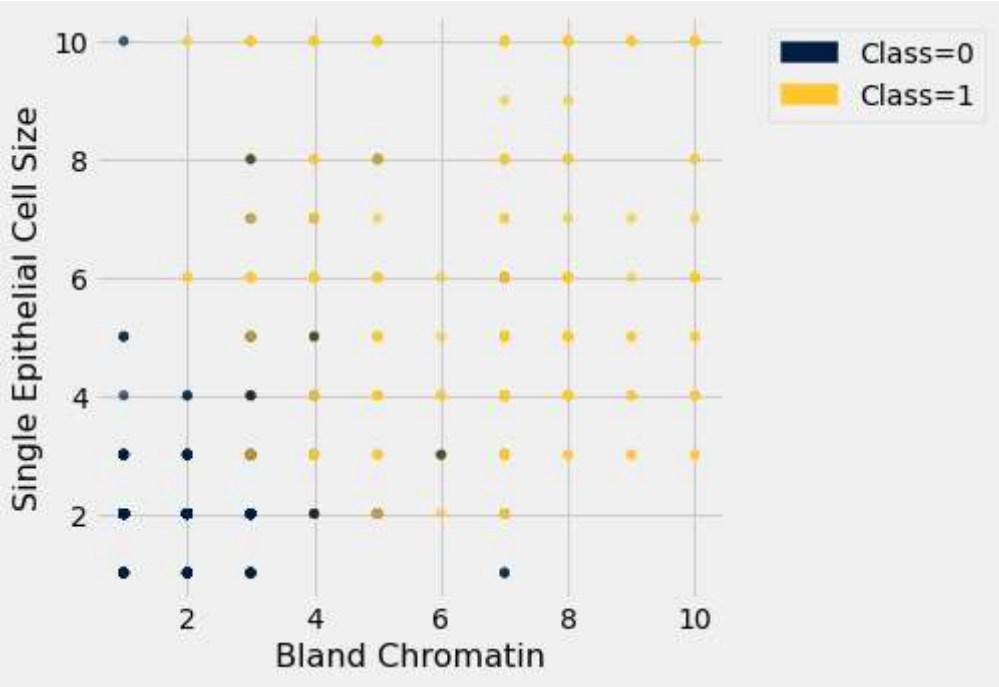


In [11]:
```
patients = Table.read_table('breast-cancer.csv').drop('ID')
patients.show(5)
```

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
5	1	1	1	2	1	3	1	1	0
5	4	4	5	7	10	3	2	1	0
3	1	1	1	2	2	3	1	1	0
6	8	8	1	3	4	3	7	1	0
4	1	1	3	2	1	3	1	1	0

... (678 rows omitted)

In [12]:
```
patients.scatter('Bland Chromatin', 'Single Epithelial Cell Size', group = 'Class')
```



In [13]:
```
def randomize_column(a):
    return a + np.random.normal(0.0, 0.09, size=len(a))

jittered = Table().with_columns([
        'Bland Chromatin (jittered)',
        randomize_column(patients.column('Bland Chromatin')),
        'Single Epithelial Cell Size (jittered)',
        randomize_column(patients.column('Single Epithelial Cell Size')),
        'Class',
        patients.column('Class')
    ])
```

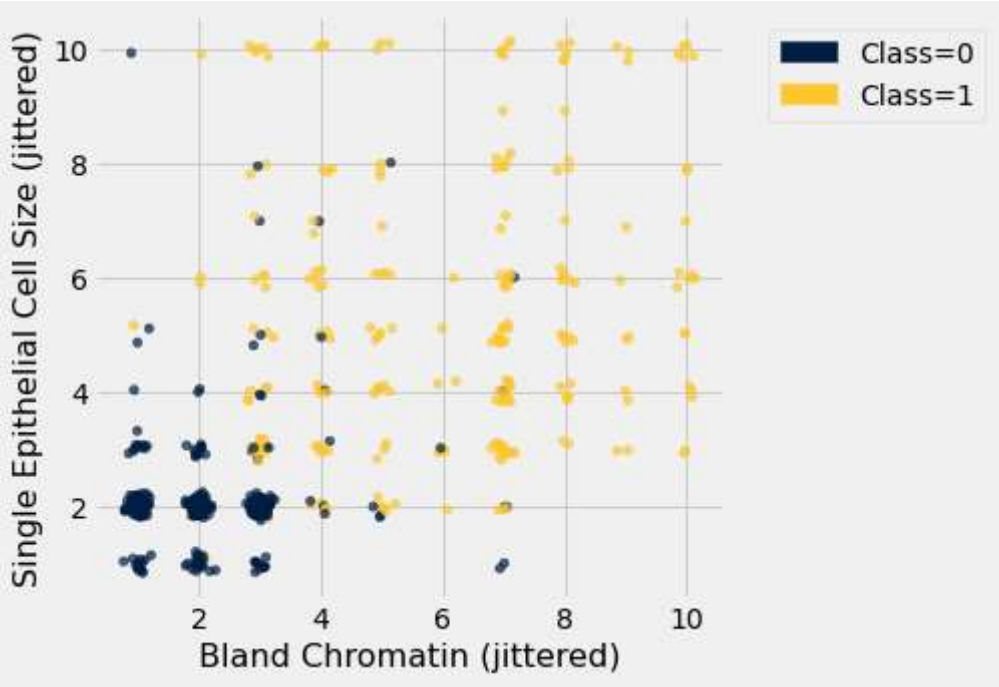
In [14]:
```
jittered
```

Out[14]:

Bland Chromatin (jittered)	Single Epithelial Cell Size (jittered)	Class
3.11984	1.88949	0
2.99751	7.00208	0
3.01831	2.09159	0
2.86836	2.97898	0
3.04583	2.00337	0
9.00477	6.89209	1
3.06453	1.9773	0
2.87751	2.07706	0
0.995618	2.15012	0
1.92147	1.98328	0

... (673 rows omitted)

In [15]:
```
jittered.scatter(0, 1, group = 'Class')
```



Distance

In [16]:
```
def distance(pt1, pt2):
    """Return the distance between two points, represented as arrays"""
    return np.sqrt(sum((pt1 - pt2)**2))

def row_distance(row1, row2):
    """Return the distance between two numerical rows of a table"""
    return distance(np.array(row1), np.array(row2))
```

In [17]:
```
attributes = patients.drop('Class')
attributes.show(3)
```

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
5	1	1	1	2	1	3	1	1
5	4	4	5	7	10	3	2	1
3	1	1	1	2	2	3	1	1

... (680 rows omitted)

In [18]:
```
row_distance(attributes.row(0), attributes.row(1))
```

Out[18]:
```
11.874342087037917
```

In [19]:
```
row_distance(attributes.row(0), attributes.row(2))
```

Out[19]:
```
2.2360679774997898
```

In [20]:
```
row_distance(attributes.row(0), attributes.row(0))
```

Out[20]: 0.0

Classification Procedure

In [21]:
```
def distances(training, example):
    """Compute distance between example and every row in training.
    Return training augmented with Distance column"""
    distances = make_array()
    attributes = training.drop('Class')
    for row in attributes.rows:
        distances = np.append(distances, row_distance(row, example))
    return training.with_column('Distance', distances)
```

In [22]:
```
patients.take(15)
```

Out[22]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
7	4	6	4	6	1	4	3	1	1

In [23]:
```
example = attributes.row(15)
example
```

Out[23]:
```
Row(Clump Thickness=7, Uniformity of Cell Size=4, Uniformity of Cell Shape=6, Margina
l Adhesion=4, Single Epithelial Cell Size=6, Bare Nuclei=1, Bland Chromatin=4, Normal
Nucleoli=3, Mitoses=1)
```

In [24]:
```
distances(patients.exclude(15), example).sort('Distance')
```

Out[24]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
8	4	6	3	3	1	4	3	1	0
5	3	4	1	4	1	3	1	1	0
5	4	5	1	8	1	3	6	1	0
8	2	4	1	5	1	5	4	4	1
9	5	5	4	4	5	4	3	3	1
6	3	3	3	3	2	6	1	1	0
9	5	5	2	2	2	5	1	1	1
3	4	5	3	7	3	4	6	1	0
5	3	3	2	3	1	3	1	1	0
5	6	5	6	10	1	3	1	1	1

... (672 rows omitted)

In [25]:
```
def closest(training, example, k):
    """Return a table of the k closest neighbors to example"""
    return distances(training, example).sort('Distance').take(np.arange(k))
```

In [26]:
```
closest(patients.exclude(15), example, 5)
```

Out[26]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
8	4	6	3	3	1	4	3	1	0
5	3	4	1	4	1	3	1	1	0
5	4	5	1	8	1	3	6	1	0
8	2	4	1	5	1	5	4	4	1
9	5	5	4	4	5	4	3	3	1

In [27]:
```
def majority_class(topk):
    """Return the class with the highest count"""
    return topk.group('Class').sort('count', descending=True).column(0).item(0)

def classify(training, example, k):
    "Return the majority class among the k nearest neighbors of example"
    return majority_class(closest(training, example, k))
```

In [28]:
```
classify(patients.exclude(15), example, 5)
```

Out[28]:
```
0
```

In [29]:
```
patients.take(15)
```

Out[29]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
7	4	6	4	6	1	4	3	1	1

In [30]:
```
new_example = attributes.row(10)
closest(patients.exclude(10), example, 5)
```

Out[30]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
7	4	6	4	6	1	4	3	1	1
8	4	6	3	3	1	4	3	1	0
5	3	4	1	4	1	3	1	1	0
5	4	5	1	8	1	3	6	1	0
8	2	4	1	5	1	5	4	4	1

In [31]:
```
classify(patients.exclude(10), new_example, 5)
```

Out[31]:
```
0
```

In [32]:
```
patients.take(10)
```

Out[32]:

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
1	1	1	1	1	1	3	1	1	0

Evaluation

In [33]:
```
patients.num_rows
```

Out[33]:
```
683
```

In [34]:
```
shuffled = patients.sample(with_replacement=False) # Randomly permute the rows
training_set = shuffled.take(np.arange(342))
test_set  = shuffled.take(np.arange(342, 683))
```

In [35]:
```
def evaluate_accuracy(training, test, k):
    """Return the proportion of correctly classified examples
    in the test set"""
    test_attributes = test.drop('Class')
    num_correct = 0
    for i in np.arange(test.num_rows):
        c = classify(training, test_attributes.row(i), k)
        num_correct = num_correct + (c == test.column('Class').item(i))
    return num_correct / test.num_rows
```

In [36]:
```
evaluate_accuracy(training_set, test_set, 5)
```

Out[36]:
```
0.967741935483871
```

In [37]:
```
evaluate_accuracy(training_set, test_set, 3)
```

Out[37]:
```
0.9589442815249267
```

In [38]:
```
evaluate_accuracy(training_set, test_set, 11)
```

Out[38]:
```
0.967741935483871
```

In [39]:
```
evaluate_accuracy(training_set, training_set, 1)
```

Out[39]:
```
1.0
```

In [ ]:
```

```