naive-bayes

November 22, 2017

In [19]:
```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from scipy.spatial import distance
from scipy.stats import mode
import numpy as np
import time
import itertools
import matplotlib.pyplot as plt
```

In [104]:
```
def use_subset(condition, x, y):
    sub_ix = np.where((y == condition[0]) | (y == condition[1]))
    y_sub = (y[sub_ix]).squeeze()
    # rename labels
    y_sub[y_sub == condition[0]] = 0
    y_sub[y_sub == condition[1]] = 1
    x_sub = (x[sub_ix, :]).squeeze()
    return x_sub, y_sub


def red_dim(x, features=[10, 60]):
    x_red = np.zeros((x.shape[0], len(features)))
    x_red[:, 0] = x[:, features[0]]
    x_red[:, 1] = x[:, features[1]]

    return x_red


def fit_naive_bayes(features, labels, bincount=0):
    def filter_data(features, labels, c):
        assert c in labels, 'Class is not in labels'
        sub_ix = c == labels
        return features[sub_ix], labels[sub_ix]

    def histogram(xx, num_bins):
        h, b = np.histogram(xx, bins=num_bins)
        return h, b
```

```
    def freedman_diaconis(X):
        iqr = np.percentile(X, 75) - np.percentile(X, 25)  # X_3/4 quartile - X_1/4
        if iqr == 0:
            delta_x = float('nan')
        else:
            delta_x = (2 * iqr) / (np.power(len(X), 1 / 3))
        return delta_x

    if bincount == 0:
        # loop over features and use reasonable freedman diaconis
        bincount_helper = np.zeros(features.shape[1])
        for i in range(features.shape[1]):
            delta_x = freedman_diaconis(features[:, i])
            bincount_helper[i] = np.ceil((np.max(features[:, i]) - np.min(features[:
        # use median value neglecting nan
        bincount = int(np.ceil(np.median(bincount_helper[np.invert(np.isnan(bincount_

    hist = np.zeros((len(labels), features.shape[1], bincount))
    binning = np.zeros((len(labels), features.shape[1], bincount + 1))
    for i, c in enumerate(labels):
        features_sub, _ = filter_data(features, labels, c)
        hist[i,:,:], binning[i,:,:] = histogram(features_sub, bincount)
    return hist, binning


# this function is way to loopy, however it works ...
def predict_naive_bayes(features, hist, binning):
    l = np.nan * np.ones((features.shape[0], hist.shape[0], features.shape[1]), dtype
    # assign instance i to correct bin
    for i in range(features.shape[0]):
        for j in range(features.shape[1]):
            for k in range(hist.shape[0]):
                l[i, k, j] = np.floor((features[i,j] - binning[k,j,0])/binning[k,j,1]
                if l[i, k, j] >= (binning.shape[2] - 1):
                    l[i, k, j] += -1

    # get N_l
    p_h = np.zeros_like(l)
    p = np.zeros_like(l[:,:,0])
    for i in range(features.shape[0]):
        for j in range(features.shape[1]):
            for k in range(hist.shape[0]):
                p_h[i, k, j] = hist[k, j, int(l[i, k, j])] / (np.sum(hist[k,j,:]) * l
    for i in range(features.shape[0]):
        for k in range(hist.shape[0]):
            p[i, k] = np.prod([p_h[i, k, j] for j in range(features.shape[1])])
    y = np.nan * np.ones(features.shape[0], dtype=np.int)
    y = np.argmax(p, 1)
```

```
            return p, y


        def pred_quality(pred, truth):
            is_eq = (pred == truth)
            pass_rate = np.sum(is_eq) / is_eq.__len__()
            err_rate = 1-pass_rate
            return pass_rate, err_rate
```

In [105]:
```
digits = load_digits()

print(digits.keys())

data = digits["data"]
images = digits["images"]
target = digits["target"]
target_names = digits["target_names"]
```

```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

In [106]:
```
x, y = use_subset([1, 7], data, target)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state

hist, binning = fit_naive_bayes(X_train, y_train)
_, y_pred = predict_naive_bayes(X_test, hist, binning)
passed, errored = pred_quality(y_pred, y_test)
print('Prediction pass rate: ', passed, ' ---- error rate: ', errored)
```

```
Prediction pass rate:  0.825688073394  ---- error rate:  0.174311926606
```

In [107]:
```
data = red_dim(data)
x, y = use_subset([1, 7], data, target)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state

hist, binning = fit_naive_bayes(X_train, y_train)
_, y_pred = predict_naive_bayes(X_test, hist, binning)
passed, errored = pred_quality(y_pred, y_test)
print('Prediction pass rate: ', passed, ' ---- error rate: ', errored)
```

```
Prediction pass rate:  0.614678899083  ---- error rate:  0.385321100917
```