svm_testing

April 6, 2019

1 SVM Implementations

1.1 Machine Learning, University of Utah

1.1.1 Cade Parkison

```
In [2]: import numpy as np
        import pandas as pd
        import cvxopt
```

1.1.2 Data Import and Preprocessing

```
In [3]: test_data = pd.read_csv('bank-note/test.csv', header=None)
        train_data = pd.read_csv('bank-note/train.csv', header=None)
```

```
In [4]: # first 7 columns are features, last column (Slump) is output
        columns = ['var', 'skew', 'curt', 'ent', 'label']
        features = columns[:-1]
        output = columns[-1]

        test_data.columns = columns
        train_data.columns = columns
```

```
In [5]: train_data.head()
```

```
Out[5]:         var      skew     curt      ent  label
        0  3.848100  10.15390 -3.85610 -4.22280      0
        1  4.004700   0.45937  1.36210  1.61810      0
        2 -0.048008  -1.60370  8.47560  0.75558      0
        3 -1.266700   2.81830 -2.42600 -1.88620      1
        4  2.203400   5.99470  0.53009  0.84998      0
```

```
In [6]: train_X = train_data.iloc[:,:-1].values
        test_X = test_data.iloc[:,:-1].values
```

```
In [7]: train_X.shape, test_X.shape
```

Out[7]: ((872, 4), (500, 4))

In [8]:
```
train_y = train_data.iloc[:,-1].values
test_y = test_data.iloc[:,-1].values
```

In [9]:
```
train_y.shape, test_y.shape
```

Out[9]: ((872,), (500,))

In [10]:
```
# Convert labels to {-1,1}
train_y = np.array([1 if x else -1 for x in train_y])
test_y = np.array([1 if x else -1 for x in test_y])

# reshape to 2D array
train_y = train_y.reshape(-1,1)
test_y = test_y.reshape(-1,1)
```

In [11]:
```
train_y.shape, test_y.shape
```

Out[11]: ((872, 1), (500, 1))

In [216]:
```
class SVM(object):

        def __init__(self, no_of_inputs, epoch, C, rate_schedule):
            self.epoch = epoch
            self.C = C
            self.rate_schedule = rate_schedule
            self.weights = np.zeros(no_of_inputs + 1)  # initialize weights to zero

        def predict(self, X):
            # predicts the label of one training example input with current weights
            return np.sign(np.dot(X, self.weights[:-1]) + self.weights[-1])

        def train(self, X, y):

            N = y.shape[0]

            #labels = np.expand_dims(labels, axis=1)
            data = np.hstack((X,y))

            for e in range(self.epoch):
                #print("Epoch:  "+ str(e))
                #print("Weights:  " + str(self.weights))
                #print('')
                rate = self.rate_schedule(e)
                np.random.shuffle(data)
                for i,row in enumerate(data):
```

```python
                x = row[:-1]
                y = row[-1]
                val = y*(np.dot(x, self.weights[:-1]) + self.weights[-1])
                if val <= 1:
                    self.weights[:-1] = (1-rate)*self.weights[:-1] + rate*self.C*N*y*
                    self.weights[-1] = rate*self.C*N*y
                else:
                    self.weights[:-1] = (1-rate)*self.weights[:-1]

        return self.weights

    def evaluate(self, X, y):
        # calculates average prediction error on testing dataset
        errors = []
        for inputs, label in zip(X, y):
            prediction = self.predict(inputs)
            if np.sign(prediction) != label:
                errors.append(1)
            else:
                errors.append(0)

        return 100*(sum(errors) / float(X.shape[0]))
```

1.2 Evaluation

1.2.1 2.2

Part a:

$$\gamma_t = \frac{\gamma_0}{1 + \frac{\gamma_0}{d} t}$$

In [250]:
```python
gamma_0 = 0.001
d = 0.01

schedule_a = lambda t:gamma_0 / (1 + (gamma_0/d)*t)
```

In [251]:
```python
C_list = [1.0/873, 10.0/873, 50.0/873, 100.0/873, 300.0/873, 500.0/873, 700.0/873]
```

In [139]:
```python
svm_1 = SVM(4, 100, 100/873, schedule_a)
svm_1.train(train_X, train_y)
svm_1.evaluate(train_X, train_y)
```

Out[139]: 0.0389908256880734

In [252]:
```python
train_errors = []
test_errors = []
for c in C_list:
```

```
            svm = SVM(4, 100, c, schedule_a)
            svm.train(train_X, train_y)
            print('C {}: weights = {}'.format(c,svm.weights))
            train_errors.append(svm.evaluate(train_X, train_y))
            test_errors.append(svm.evaluate(test_X, test_y))
        print('Training Errors: {}'.format(train_errors))
        print('Testing Errors: {}'.format(test_errors))
```

```
C 0.001145475372279496: weights = [-3.74999012e-01 -1.71782612e-01 -1.46115309e-01 -7.80712244e
  9.16380298e-05]
C 0.011454753722794959: weights = [-0.74050692 -0.37308407 -0.37429121 -0.19790761 -0.00091638]
C 0.0572737686139748: weights = [-1.13026463 -0.55535113 -0.69693188 -0.31042097  0.0045819 ]
C 0.1145475372279496: weights = [-1.44720154 -0.71534303 -0.80515422 -0.32493326  0.0091638 ]
C 0.3436426116838488: weights = [-2.00728194 -1.1476715  -1.04464131 -0.68690303  0.02749141]
C 0.572737686139748: weights = [-2.62307216 -1.73517131 -1.29206735 -1.02332767  0.04581901]
C 0.8018327605956472: weights = [-3.42848911 -2.24249739 -1.97210138 -0.83556891 -0.06414662]
Training Errors: [5.045871559633028, 4.128440366972478, 5.045871559633028, 4.013761467889909, 4
Testing Errors: [7.199999999999999, 4.8, 6.4, 4.8, 5.6000000000000005, 6.800000000000001, 7.000
```

```
In [253]: schedule_b = lambda t:gamma_0 / (1 + t)
```

```
In [254]: train_errors = []
          test_errors = []
          for c in C_list:
              svm = SVM(4, 100, c, schedule_b)
              svm.train(train_X, train_y)
              print('C {}: weights = {}'.format(c,svm.weights))
              train_errors.append(svm.evaluate(train_X, train_y))
              test_errors.append(svm.evaluate(test_X, test_y))
          print('Training Errors: {}'.format(train_errors))
          print('Testing Errors: {}'.format(test_errors))
```

```
C 0.001145475372279496: weights = [-3.73505563e-01 -1.70655237e-01 -1.45437691e-01 -7.89171176e
  9.98854525e-06]
C 0.011454753722794959: weights = [-7.28630240e-01 -3.68194467e-01 -3.69761376e-01 -1.93643088e
  9.98854525e-05]
C 0.0572737686139748: weights = [-1.10445012e+00 -5.89441091e-01 -6.33769389e-01 -2.47590179e-0
  4.99427262e-04]
C 0.1145475372279496: weights = [-1.26426257e+00 -6.76874543e-01 -7.32126995e-01 -2.82249291e-0
  9.98854525e-04]
C 0.3436426116838488: weights = [-1.58527834 -0.85788808 -0.90598141 -0.33869477  0.00299656]
C 0.572737686139748: weights = [-1.72428192 -0.96253544 -1.05129922 -0.35398384  0.00499427]
C 0.8018327605956472: weights = [-1.88370336 -1.07649151 -1.07785222 -0.39328958  0.00699198]
Training Errors: [5.045871559633028, 4.128440366972478, 4.013761467889909, 4.013761467889909, 3
Testing Errors: [7.199999999999999, 4.8, 4.6, 4.6, 4.8, 4.8, 5.2]
```

2 Dual SVM

Dual Form SVM:

$$\min_{\{0\leq\alpha_i\leq C\},\sum_i \alpha_i y_i=0} \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j {x_i}^T x_j - \sum_i \alpha_i$$

Dual form with Kernel:

$$\min_{\alpha} \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_i \alpha_i$$

$$s.t.\ 0 \leq \alpha_i \leq C$$

$$\sum_i \alpha_i y_i = 0$$

Converting to Matrix notation:
H is a matrix such that $H_{i,j} = y_i y_j K(x_i, x_j)$
We now convert the sums into vectors:

$$\min_{\alpha} \frac{1}{2}\alpha^T \mathbf{Hff} - \mathbf{1^T ff}$$

$$s.t.\ 0 \leq \alpha_i \leq C$$

$$y^T \alpha = 0$$

CVXOPT QP Solver:

$$\min_{x} \frac{1}{2}x^T P x - q^T x$$

$$s.t. Gx \leq h$$

$$and Ax = b$$

Converting to cvxopt format:
$P = H$ matrix (mxm)
$q = -1$ vector (mx1)
G = (2mxm) matrix, first 3 rows are 0 <= alpha constraint, last 3 are alpha <= C constraint
h = vector (2mx1) , first 3 elements are 0, last three elements are C
$A = y$ labels vector (mx1)
$b = 0$ scalar

```
In [13]: def linear_kernel(x1, x2):
             return np.dot(x1, x2)
```

```
In [247]: def gaussian_kernel(gamma=0.5):
              return lambda x,y: np.exp(-np.linalg.norm(x-y)**2 / gamma)
```

```
In [261]: class DualSVM(object):

              def __init__(self, C, kernel=linear_kernel):
                  self.C = C
                  self.kernel = kernel

              def predict(self, inputs):
                  # predicts the label of one training example input with current weights
                  if self.kernel == linear_kernel:
                      return  np.sign(np.dot(inputs, self.weights[:-1]) + self.weights[-1])
                  else:
                      result = 0
                      for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
                          result += a * sv_y * self.kernel(inputs, sv)

                      return np.sign(result).item()

              def train(self, X, y):
                  n_samples, n_features = X.shape

                  # Kernel Matrix
                  K = np.zeros((n_samples, n_samples))
                  for i in range(n_samples):
                      for j in range(n_samples):
                          K[i,j] = self.kernel(X[i], X[j])

                  P = cvxopt.matrix(np.outer(y,y)*K)
                  q = cvxopt.matrix(-1*np.ones(n_samples))
                  G = cvxopt.matrix(np.vstack((np.diag(-1*np.ones(n_samples)), np.identity(n_sa
                  h = cvxopt.matrix(np.hstack((np.zeros(n_samples),self.C*np.ones(n_samples)))
                  A = cvxopt.matrix(y, (1,n_samples), 'd')
                  b = cvxopt.matrix(0.0)

                  cvxopt.solvers.options['show_progress'] = False
                  cvxopt.solvers.options['abstol'] = 1e-10
                  cvxopt.solvers.options['reltol'] = 1e-10
                  cvxopt.solvers.options['feastol'] = 1e-10

                  # Quadratic Programming solution from cvxopt
                  sol = cvxopt.solvers.qp(P,q,G,h,A,b)

                  # Lagrange Multipliers
                  alphas = np.array(sol['x'])

                  # weights
                  w = ((y * alphas).T @ X).reshape(-1,1)
                  # non-zero alphas
                  S = (alphas > 1e-4).flatten()
```

```
            self.S = S
            self.n_supports = np.sum(S)
            # intercept
            b = y[S] - np.dot(X[S], w)

            ind = np.arange(len(alphas))[S]
            self.a = alphas[S]
            self.sv = X[S]

            self.sv_y = y[S]

            self.b = 0
            for n in range(len(self.a)):
                self.b += float(self.sv_y[n])
                self.b -= np.sum(self.a * self.sv_y * K[ind[n],S])
            self.b /= len(self.a)

            self.weights = np.zeros(n_features + 1)
            self.weights[:-1] = w.flatten()
            self.weights[-1] = b[0]

        def evaluate(self, X, y):
            # calculates average prediction error on dataset, in percentage
            errors = []
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                if np.sign(prediction) != label:
                    errors.append(1)
                else:
                    errors.append(0)

            return 100*(sum(errors) / float(X.shape[0]))
```

In [158]:
```
C_list = [100.0/873, 500.0/873, 700.0/873]
```

In [255]:
```
# Training and Testing errors for each C value
train_errors = []
test_errors = []
for c in C:
    svm = DualSVM(c,kernel=linear_kernel)
    svm.train(train_X,train_y)
    print('C {}: weights = {}'.format(c,svm.weights))
    train_errors.append(svm.evaluate(train_X,train_y))
    test_errors.append(svm.evaluate(test_X,test_y))
print(train_errors, test_errors)
```

```
C 0.1145475372279496: weights = [-0.94303948 -0.65147876 -0.73370349 -0.04098535  1.52256115]
C 0.572737686139748: weights = [-1.56426251 -1.0137622  -1.18050792 -0.15618296  1.91748011]
```

```
C 0.8018327605956472: weights = [-2.04253733 -1.28008058 -1.5132451  -0.24830283  2.18696284]
[1.4908256880733946, 0.8027522935779817, 0.8027522935779817] [1.4000000000000001, 0.8, 0.8]
```

Gaussian Kernel

```
In [244]: gamma_list = [0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 100.0]
```

```
In [249]: # Training and Testing errors for each C value
          train_errors = []
          test_errors = []
          for c in C:
              svm = DualSVM(c,kernel=gaussian_kernel())
              svm.train(train_X,train_y)
              train_errors.append(svm.evaluate(train_X,train_y))
              test_errors.append(svm.evaluate(test_X,test_y))
          print(train_errors, test_errors)
```

```
[0.0, 0.0, 0.0] [0.2, 0.2, 0.2]
```

```
In [245]: for g in gamma_list:
              train_errors = []
              test_errors = []
              for c in C:
                  svm = DualSVM(c,kernel=gaussian_kernel(g))
                  svm.train(train_X,train_y)
                  train_errors.append(svm.evaluate(train_X,train_y))
                  test_errors.append(svm.evaluate(test_X,test_y))
              print('Gamma {}: {}, {}'.format(g,train_errors, test_errors))
```

```
Gamma 0.01: [0.0, 0.0, 0.0], [0.2, 0.2, 0.2]
Gamma 0.1: [0.0, 0.0, 0.0], [0.2, 0.2, 0.2]
Gamma 0.5: [0.0, 0.0, 0.0], [0.2, 0.2, 0.2]
Gamma 1: [0.0, 0.0, 0.0], [0.2, 0.2, 0.2]
Gamma 2: [0.0, 0.0, 0.0], [0.2, 0.2, 0.2]
Gamma 5: [0.8027522935779817, 0.0, 0.0], [0.6, 0.2, 0.2]
Gamma 10: [0.8027522935779817, 0.0, 0.0], [0.6, 0.2, 0.2]
Gamma 100: [0.34403669724770647, 0.0, 0.0], [0.4, 0.0, 0.0]
```

```
In [260]: for g in gamma_list:
              supports = []
              for c in C:
                  svm = DualSVM(c,kernel=gaussian_kernel(g))
                  svm.train(train_X,train_y)
                  supports.append(svm.n_supports)
```

```
                #train_errors.append(svm.evaluate(train_X,train_y))
                #test_errors.append(svm.evaluate(test_X,test_y))
            print('N Supports: {}'.format(supports))
```

```
N Supports: [872, 872, 872]
N Supports: [869, 868, 864]
N Supports: [825, 730, 689]
N Supports: [805, 555, 519]
N Supports: [693, 389, 359]
N Supports: [442, 208, 193]
N Supports: [316, 130, 114]
N Supports: [290, 116, 98]
```

In [262]:
```
supports = []
for g in gamma_list:
    svm = DualSVM(500/873,kernel=gaussian_kernel(g))
    svm.train(train_X,train_y)
    supports.append(svm.S)
```

In [264]:
```
overlap_supports = []
for i in range(7):
    overlap_supports.append(np.sum(np.logical_and(supports[i], supports[i+1])))
print(overlap_supports)
```

```
[868, 730, 552, 379, 194, 122, 64]
```