bayes_optimal

September 18, 2017

1 Bayes optimal classifier

1.1 Generating 10 centroids for inputs (x's)

```
In [124]: %matplotlib notebook

          import matplotlib.pyplot as plt
          import numpy as np, pickle
          from sklearn import linear_model
          from sklearn.metrics import confusion_matrix

          # Generating 10 centroids for inputs (x's)
          x_centroids_class1 = np.random.multivariate_normal(mean = [1, 0], cov = np.identity(2
          x_centroids_class2 = np.random.multivariate_normal(mean = [0, 1], cov = np.identity(2
          x_centroids = np.append(x_centroids_class1, x_centroids_class2, axis = 0) #All centro

          #plotting
          ax = plt.subplots()[1]
          ax.plot(x_centroids_class1[:, 0], x_centroids_class1[:, 1], marker = '^', linestyle =
          ax.plot(x_centroids_class2[:, 0], x_centroids_class2[:, 1], marker = '^', linestyle =
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Out[124]: [<matplotlib.lines.Line2D at 0x29a7b7f0>]
```

1.2 Generating training data around centroids

```
In [125]: # Generating data around centroids
          x_train, y_train = [], []

          for _ in range(100):
              centroid_idx = np.random.choice(10) #sample i for m_i from 1 to 10
              x_train.append(np.random.multivariate_normal(mean = x_centroids[centroid_idx], co
              if centroid_idx < 5: y_train.append(0) #if i in {0, ..4}, y = 0
```

```
            else: y_train.append(1) #if i in {5, ..9}, y = 1

        x_train, y_train = np.array(x_train), np.array(y_train) #lists to arrays
        #plotting
        ax = plt.subplots()[1]
        ax.plot(x_train[y_train == 0, 0], x_train[y_train == 0, 1], linestyle = '', marker =
        ax.plot(x_train[y_train == 1, 0], x_train[y_train == 1, 1], linestyle = '', marker =
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Out[125]: [<matplotlib.lines.Line2D at 0x29c5c668>]
```

1.3 Boundary plotting function

```
In [126]: def plot_boundary(ax, clf, x, y, **params):
              """Plot the decision boundaries for a classifier.

              Parameters
              ----------
              ax: matplotlib axes object
              clf: a classifier
              x: data to base x-axis meshgrid on
              y: data to base y-axis meshgrid on
              params: dictionary of params to pass to contourf, optional
              """

              def make_meshgrid(x, y):
                  grid_res = 1000
                  x_min, x_max = x.min(), x.max()
                  y_min, y_max = y.min(), y.max()
                  xx, yy = np.meshgrid(np.arange(x_min, x_max, (x_max - x_min)*1./grid_res)[:g
                                       np.arange(y_min, y_max, (y_max - y_min)*1./grid_res)[:g
                  return xx, yy

              xx, yy = make_meshgrid(x, y)
              Z = clf(np.c_[xx.ravel(), yy.ravel()])
              Z = Z.reshape(xx.shape)
              ax.contourf(xx, yy, Z, **params)
              ax.set_xlim(xx.min(), xx.max())
              ax.set_ylim(yy.min(), yy.max())
```

1.4 Testing a linear model

We fit a linear model to the training data and draw the decision boundary (subspace where y ~= 0.5). We highlight the false positive and false negatives in the model predictions and print the rates of both.

```
In [127]: #train regression model
          model = linear_model.LinearRegression()
          model.fit(X = x_train, y = y_train)

          #plotting
          ax = pickle.loads(pickle.dumps(ax))
          plot_boundary(ax, lambda x: np.floor(model.predict(x) + 0.5), x_train[:, 0], x_train

          # False positives and false negatives for training data
          y_pred = np.vectorize(lambda y: 0 if y < 0.5 else 1)(model.predict(X = x_train)) #ru
          x_fp, x_fn = x_train[np.logical_and(y_pred == 1, y_train == 0)], x_train[np.logical_a
          #plotting
          ax.plot(x_fp[:, 0], x_fp[:, 1], linestyle = '', marker = 's', color = 'red', alpha =
          ax.plot(x_fn[:, 0], x_fn[:, 1], linestyle = '', marker = 's', color = 'blue', alpha =

          #fp and fn rates
          tn, fp, fn, tp = confusion_matrix(y_train, y_pred).ravel()
          fp_rate, fn_rate = (fp + 0.)/(fp + tn)*100., (fn + 0.)/(fn + tp)*100.

          print 'false positive rate = ' + str(fp_rate) + '% , false negative rate = ' + str(fn
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
false positive rate = 38.8888888889% , false negative rate = 30.4347826087%
```

```
In [ ]:
```