Assignment (MiniProject) 3

Machiry Aravind Kumar

UCSB

1 Dataset

1.1 Seeds Data Set

This dataset is from UCI Archive: `https://archive.ics.uci.edu/ml/datasets/seeds`. Dataset contains various Geometric parameters of wheat kernels measured using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The problem is to classify these into one of the 3 different classes viz Kama, Rosa and Canadian. For this Assignment, I selected samples for only 2 classes: Kama (Class label: 1) and Rosa (Class label: 2). Available Features in the dataset are:

- area A.
- perimeter P.
- compactness C.
- length of kernel.
- width of kernel,
- asymmetry coefficient
- length of kernel groove.

2 Running SVM

I used SVM package from `sklearn` module. I used following kernel functions: `linear,rbf` and `sigmoid` with different values of Penalty Parameter(C) which trades off misclassification of training examples against simplicity of the decision surface, in other words this parameter controls over fitting, larger value of C results in over fitting and Kernel coefficient (gamma) which defines how much influence a single training example has. Its clear that, we need to select a kernel function which gives highest accuracy with least C and gamma. The error rates (with 10-fold validation) of using different kernel functions with different C and gamma are as shown in Table 1. I selected kernel function `linear` with $C = 1.4$ and $gamma = 0$ as it gives the best result satisfying our constraints.
The results for different values of n are as shown in Table 2. The results are pretty accurate with 96.42% Accuracy and doesn't vary much with n, which proves that features are good for the classification. It also proves that a linear kernel with SVM is good fit for this classification problem.

Kernel Function	**C**	**gamma**	**Mean Accuracy**
rbf	0.1	0	93.57
sigmoid	0.1	0	50.0
linear	0.1	0	94.28
linear	1	0	94.28
linear	1.1	0	94.42
linear	1.2	0	95.0
linear	1.3	0	95.71
linear	**1.4**	**0**	**96.42**
linear	1.5	0	96.42
linear	10	0	96.42

Table 1: Effectiveness of various Kernel Functions

N	**Mean Accuracy**	**Mean Error**
3	94.98	5.01
4	95.00	5.00
5	94.28	5.71
6	94.26	5.73
7	95.00	5.00
8	94.89	5.10
9	94.95	5.04
10	96.42	3.57
11	95.74	4.25

Table 2: Cross Validation Results for Seeds Dataset using SVM with linear kernel

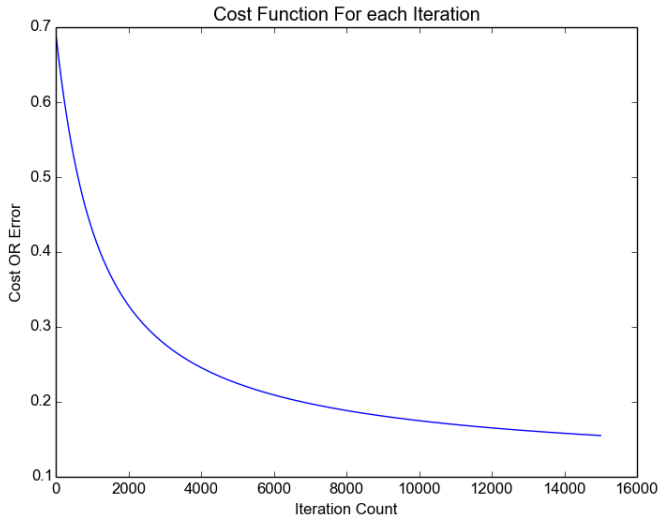

Figure 1: Cost Function Trend Against Iterations

3 LDF with gradient descent

I implemented LDF with gradient descent where I used sigmoid function:$1/(1+e^{-y})$ as hypothesis to predict the value. Predicting class 2 if result is greater than or equal to 0.5 (>= 0.5) and Predicting class 1 otherwise. Using sigmoid function for hypotheses results in cost function as shown:

$-\frac{1}{m}\sum_{i=1}^{m} y^i \log(h_\theta(x^i)) + (1-y^i)\log(1-h_\theta(x^i))$

Another important reason to use sigmoid function as hypotheses rather than default $W^T X$ is because, **the resulting cost function during computation of gradient descent using default method is non-convex thus resulting in various local minima and thus poor accuracy. Where as using sigmoid function results in cost function being convex resulting in global minima and good accuracy. The trend of cost function is as shown in Figure 1**

With the above cost function, I used learning rate (i.e α) of 0.0005 with regularization parameter (i.e λ) as 0.01 (to avoid over fitting) with 10,000 iterations. Also, I normalized the feature values for all the features by subtracting mean and dividing it by standard deviation: $x_i = (x_i - \mu(x))/\rho(x)$ The core algorithm in python as shown below:

```
for i in xrange(0, iters):

  theta_temp = self.theta

  #compute_hypothesis
  h_theta = self._sigmoid(numpy.dot(self.X, self.theta))

  #compute error
  diff = h_theta - self.Y

  #update theta
```

N	Mean Accuracy	Mean Error
3	96.40	3.59
4	96.42	3.57
5	95.00	5.00
6	94.98	5.01
7	95.00	5.00
8	94.24	5.75
9	94.21	5.78
10	94.28	5.71
11	94.11	5.88

Table 3: Cross Validation Results for Seeds Dataset using Gradient Descent

```
self.theta[0] = theta_temp[0]    self.learning_rate * \
   (1.0 / m) * sum(diff * self.X[:, 0])
for j in xrange(1, n):
 val = theta_temp[
       j] - self.learning_rate * (1.0 / m) * \
       (sum(diff * self.X[:, j]) + self.reg_param * m * theta_temp[j])

self.theta[j] = val
```

The complete code for this is present in file `gradient_descent.py`. Cross validation results for various values of n is as shown in Table 3. The results are pretty accurate with 94.00% Accuracy and doesn't vary much with n, which proves that features are linearly separable and gradient descent is a good searching strategy.
Weighted Vector computed using Gradient Descent is: [0.56232669359224408, 0.003148429546239455, 0.49146013233794189, 0.48967826722053615, 0.3474530090250722, 0.66867992711948321]

4 Comparison: SVM v/s GradientDescent LDF

SVM is both faster (by 3 seconds) and slightly more accurate(1.0%) than LDF implemented using Gradient Descent. But it took some time to figure out correct kernel parameters for SVM where as for Gradient Descent, we need to normalize the feature values.
ROC Curve, whose area is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. The area should be more for classifier to be Good.
ROC Curves For SVM and Gradient Descent are shown in Figure 2 and 3.
Area under **ROC curves for SVM and Gradient Descent is same and its 98.69**. which proves that both are equally accurate for this binary classification problem.

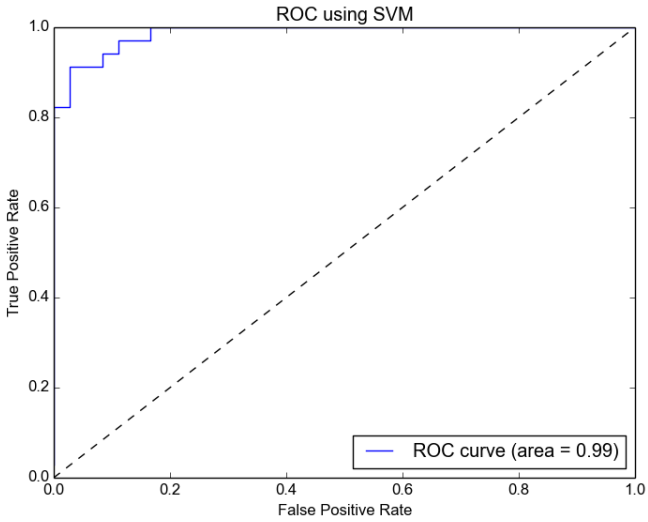

Figure 2: ROC Curve using SVM

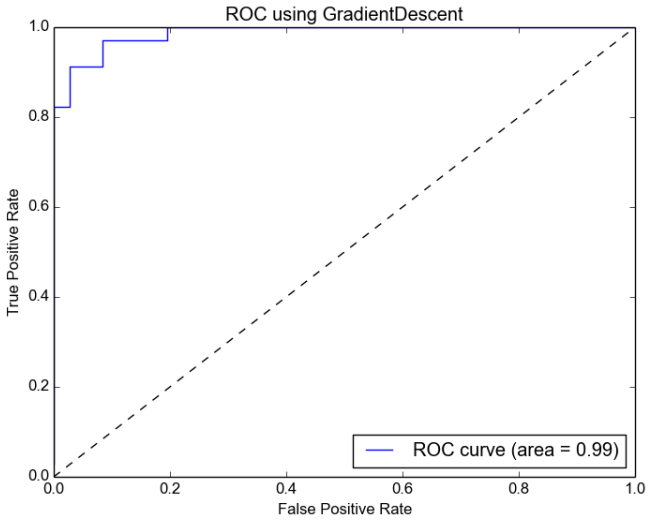

Figure 3: ROC Curve using Gradient Descent