**Report - Homework 1**

Author: Korede Adegboye

# 1 Abstract

In this project we investigated the performance of linear classification models on two benchmark datasets. We found that the logistic regression approach achieved worse accuracy than LDA and was slightly slower to train. Using Parkinsons data, 96% of the patients with Parkinson's disease experienced a variety of voice changes. Secondly, Sonar, Mines vs. Rocks data, used bouncing sonar signals to determine that there were 92% of the signals bounced off a metal.

# 2 Introduction

Throughout the project, features were continuously assesed. Specifically, the Parkison's data was found to be imbalanced. While, the Sonar, Mines vs. Rocks data included a string valued outputs, and were sorted by these outputs. After, we proceeded to look at the distributions of some of the features and with respect to there individual columns. Doing this allowed for us to determine, an approximate distribution of our features. Before applying our classification models, the data needed to be split. 80% of our data for the training set and the remaining for test set. This split was best for both data sets. To accomodate for the sorted output values of the Sonar, Mines vs. Rocks data set, random sampling of the data was used. This procedure was also considered for Parkinson's data, but the results were only slightly affected. Continuing, Logistic Regression was not favorable in regards to accuracy, and time. In other words, for both datasets, it was clear that LDA (Linear Discriminant Analysis) was the better of the models. Lastly, a 10 folds cross validation procedure was used to further limit the learning on our training set.

# 3 Datasets

The distribution of the Parkinsons data is given by the status ($y_i$). And thus, it has a binary classification, since status can only take on the values 1 and 0. Where 0 is set for healthy and 1 for Parkinsons Disease. The distribution of some of the features are unclear, and vary alot. As for the Sonar, Mines vs Rocks data, the distrubution is given by binary classification, since the output labels can only be of R - rocks or M - Mines. The distribution of some of the features are observed as a gaussian distribution. In addition, it is noticed that the Parkinsons data has some negative values, in the column named "spread1". Hence, we will to define a subset without this feature.

# 4 Results

We discovered a better understanding of Logistic regressions performance and dependence on the learning rate. Referring to the Sonar, Mines vs. Rocks data a learning rate of 0.04 was chosen. After much trial and error, analzing the change in weights, and number of iterations. Thus, 0.04 learning rate gave the highest, recall, accuracy and precision. Note, the number of iterations, is not satisfying. As for the Parkinsons data, since we noticed the output values were imbalanced. We need not to analyze the accuracy. Precision and recall, were the best indicators of a good learning rate.

```
convergence 2353
CPU times: user 3.59 s, sys: 105 ms, total: 3.69 s
Wall time: 3.59 s
Learning Rate:  0.04
Target:  [1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1]
Prediction:  [1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1]
Accuracy 0.7692307692307693
Predicted   0  1
Actual
0          11  2
1           4  9
accuracy 0.7692307692307693
recall 0.6923076923076923
precision 0.8181818181818182
```

Figure 1: Logistic Regression - SMR Data"

```
CPU times: user 7.83 ms, sys: 4.27 ms, total: 12.1 ms
Wall time: 22 ms
Target:  [1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1]
Prediction:  [1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1]
Accuracy 0.9230769230769231
Predicted   0   1
Actual
0          12   1
1           1  12
accuracy 0.9230769230769231
recall 0.9230769230769231
precision 0.9230769230769231
```

Figure 2: LDA - SMR Data"

```
CPU times: user 70.4 ms, sys: 622 µs, total: 71 ms
Wall time: 71.6 ms
Learning Rate:  0.02
Target:  [1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1]
Prediction:  [1 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1]
Accuracy 0.6666666666666666
Predicted  0   1
Actual
0          2   3
1          5  14
accuracy 0.6666666666666666
recall 0.7368421052631579
precision 0.8235294117647058
```

Figure 3: Logistic Regression - Parkinsons Data"

```
CPU times: user 2.06 ms, sys: 564 µs, total: 2.62 ms
Wall time: 2.82 ms
Target:  [1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1]
Prediction:  [1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1]
Accuracy 0.9583333333333334
Predicted  0   1
Actual
0          4   1
1          0  19
accuracy 0.9583333333333334
recall 1.0
precision 0.95
```

Figure 4: LDA - Parkinsons Data"

```
Logistic Regression - Kfold Cross Validation - SMR Data
fold:  1  -> Accuracy:  0.6666666666666666
fold:  2  -> Accuracy:  0.7380952380952381
fold:  3  -> Accuracy:  0.7142857142857143
fold:  4  -> Accuracy:  0.7142857142857143
fold:  5  -> Accuracy:  0.7238095238095238
fold:  6  -> Accuracy:  0.7063492063492064
fold:  7  -> Accuracy:  0.7142857142857143
fold:  8  -> Accuracy:  0.7142857142857143
fold:  9  -> Accuracy:  0.7074468085106383
fold:  10  -> Accuracy:  0.7115384615384616
```

Figure 5: Logistic Regression - Kfold Cross Validation - SMR Data

```
LDA - Kfold Cross Validation - SMR Data
fold:  1  -> Accuracy:  0.8571428571428571
fold:  2  -> Accuracy:  0.7857142857142857
fold:  3  -> Accuracy:  0.7936507936507936
fold:  4  -> Accuracy:  0.75
fold:  5  -> Accuracy:  0.7333333333333333
fold:  6  -> Accuracy:  0.7380952380952381
fold:  7  -> Accuracy:  0.7482993197278912
fold:  8  -> Accuracy:  0.7738095238095238
fold:  9  -> Accuracy:  0.7659574468085106
fold:  10  -> Accuracy:  0.7644230769230769
```

Figure 6: LDA - Kfold Cross Validation - SMR Data"

## 5 Discussion and Conclusion

A better analysis of the features, would have presented a better performance across both data sets. With this project, it has re-occured that features have a great affect on the implementation, and much time should be spent assessing them.

```
Logistic Regression - Kfold Cross Validation - Parkinsons Data
fold:  1  -> Accuracy:  0.8
fold:  2  -> Accuracy:  0.775
fold:  3  -> Accuracy:  0.6
fold:  4  -> Accuracy:  0.65
fold:  5  -> Accuracy:  0.69
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10:
  # Remove the CWD from sys.path while we load stuff.
fold:  6  -> Accuracy:  0.6722689075630253
fold:  7  -> Accuracy:  0.6159420289855072
fold:  8  -> Accuracy:  0.6305732484076433
fold:  9  -> Accuracy:  0.6477272727272727
fold:  10  -> Accuracy:  0.6564102564102564
```

Figure 7: Logistic Regression - Kfold Cross Validation - Parkinsons Data"

```
LDA - Kfold Cross Validation - Parkinsons Data
fold:  1  -> Accuracy:  0.9
fold:  2  -> Accuracy:  0.95
fold:  3  -> Accuracy:  0.9333333333333333
fold:  4  -> Accuracy:  0.925
fold:  5  -> Accuracy:  0.93
fold:  6  -> Accuracy:  0.9327731092436975
fold:  7  -> Accuracy:  0.9057971014492754
fold:  8  -> Accuracy:  0.89171974522293
fold:  9  -> Accuracy:  0.8977272727272727
fold:  10  -> Accuracy:  0.882051282051282
```

Figure 8: LDA - Kfold Cross Validation - Parkinsons Data"

# 6 Statement of Contributions

All the work was done by the author

# 7 Appendix

```python
class LogisticRegression:
  #Constructor
  def __init__(self, learning_rate , max_iter=1000, epsilon = 1e-6):
    self.learning_rate = learning_rate
    self.max_iter = max_iter
    self.epsilon = epsilon

  def logistic_function (self, a):
    a = a.astype(float)
    return (1 / (1 + np.exp(-a)))

  def fit(self, X, y):

    self.weights = np.zeros(X.shape[1]) # initialize weights
    k_iter = 0
    change_in_weights = 1
    for k in range(self.max_iter):
      # y = true_output
      # sigma = predicted output
      # X.T = X^T, i.e. X transpose
      a = np.dot(X, self.weights)

      sigma_y_pred = self.logistic_function(a)
      gradient = (np.dot(X.T, (sigma_y_pred - y))/X.shape[0])

      # update rule

      w_k = self.weights
      self.weights = self.weights - (self.learning_rate * gradient)
      change_in_weights = np.linalg.norm(abs(self.weights - w_k))

      # print("change in weights = ", change_in_weights)
      # convergence check
      if(change_in_weights < self.epsilon):
        print("convergence", k)
        break

      # y_pred = self.predict(X,)
      # print('iteration', k, 'accuracy', self.Accu_eval(y_pred,y))

  def predict(self, X,y):
    # calculate probabilities for a given feature vector x_new
    a = np.dot(X, self.weights)
    y_pred = self.logistic_function(a)
    # print(y_pred)
    threshold = 0.5
    return (y_pred > threshold).astype(int)

  def Accu_eval(self,y_pred, test):
    accuracy = np.mean(y_pred == test)
    return accuracy
```

Figure 9: Logistic Regression Algorithm"

```python
class LDA:
  def mean_class(self, X):
    sum_mu = np.zeros(shape=(1, len(X[0])))
    for i in range(len(X)):
      sum_mu += np.array(X[i])
    sum_mu = sum_mu/len(X)
    return sum_mu
  def sigma_class(self, X, mu):
    sum_sigma = np.zeros(shape=(len(X[0]), len(X[0])))
    for i in range(X.shape[0]):
      diff = np.array(X[i].T) - mu
      sum_sigma += (diff*diff.T)
    return sum_sigma

  def log_odds_ratio(self,X,y):
    log_odds_ratio = np.zeros_like(y, float)
    for i in range(X.shape[0]):
      xTw = np.linalg.multi_dot([X[i], self.sigma_inv, (self.mu_1.T - self.mu_0.T)])
      log_odds_ratio[i] = self.w_0 + xTw
    return log_odds_ratio

  def fit(self, X, y):
    X_0  = X[y==0]
    X_1 = X[y==1]
    X_0 = np.array(X_0)
    X_1 = np.array(X_1)
    self.mu_0 = self.mean_class(X_0)
    self.mu_1 = self.mean_class(X_1)
    mu_0 = self.mu_0
    mu_1 = self.mu_1
    N_0 = X_0.shape[0]
    N_1 = X_1.shape[0]
    prob_0 = N_0/int(N_0+N_1)
    prob_1 = N_1/int(N_0+N_1)
    sigma_0 = self.sigma_class(X_0, mu_0)
    sigma_1 = self.sigma_class(X_1, mu_1)
    sigma = (sigma_0+sigma_1)/(N_0+N_1-2)
    # print("sigma_total = ", sigma)
    log_probs = np.log(prob_1/prob_0)
    # print(log_probs)
    self.sigma_inv = np.linalg.pinv(sigma)
    sigma_inv = self.sigma_inv
    self.w_0 = log_probs - (0.5)*np.linalg.multi_dot([mu_1,sigma_inv,mu_1.T]) + (0.5)*np.linalg.multi_dot([mu_0,sigma_inv,mu_0.T])
  def predict(self, X,y):
    log_odds_ratio = self.log_odds_ratio(X,y)
    return (log_odds_ratio > 0).astype(int)

  def Accu_eval(self,pred, true):
    accuracy = np.mean(pred == true)
    return accuracy
```

Figure 10: LDA Algorithm"

```python
class KfoldCrossValidation:
  #Constructor
  def __init__(self, k, classifier):
    self.k = k
    self.classifier = classifier

  def k_folds(self, X,y):
    self.folds_X = np.array_split(X, self.k)
    self.folds_y = np.array_split(y, self.k)

  def cross_validation(self):
    pred = []
    true = []
    for i in range(self.k):
      training_set_X = self.folds_X.copy()
      training_set_y = self.folds_y.copy()

      validation_set_X = self.folds_X[i].copy()
      validation_set_y = self.folds_y[i].copy()

      del training_set_X[i]
      del training_set_y[i]

      training_set_X = np.concatenate(training_set_X)
      training_set_y = np.concatenate(training_set_y)

      self.classifier.fit(training_set_X.copy(),training_set_y.copy())

      pred.extend(self.classifier.predict(validation_set_X, validation_set_y))
      true.extend(validation_set_y)

      print("fold: ", i+1," -> Accuracy: " ,self.classifier.Accu_eval(np.asarray(pred), np.asarray(true)))
```

Figure 11: Kfold Cross Validation Algorithm