iris_dataset_classification

March 18, 2018

1 Libraries

```
In [1]: import numpy as np
        import time
        from sklearn import datasets
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score
```

```
In [2]: iris = datasets.load_iris()
        list(iris.keys())
```

```
Out[2]: ['data', 'target', 'target_names', 'DESCR', 'feature_names']
```

```
In [3]: print(iris.DESCR)
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
```

```
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [4]:
```python
print(iris.target) #gives a detailed descriptipon of the Iris dataset
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
```

```
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

2 Merging data features

```
In [5]: data = np.array(iris['data'])
        # print(data)
        data_with_labels=np.insert(data, 0, values=iris['target'], axis=1) # first element is th
        # print(data_with_labels)
```

3 Creation of Test and Train dataset

```
In [6]: train_set, test_set=train_test_split(data_with_labels,test_size=0.3,random_state=42)
```

4 Binary Classification

```
In [7]: X=train_set[:,(3,4)] # taking feature petal length and petal width
        # Y=(train_set[:,0]==2).astype(np.int) # to map true and false to 1 and 0 respectively
        # print(X)
        # print(Y)
        test_data=test_set[:,(3,4)] # taking feature petal length and petal width
        train_labels1=(train_set[:,0]==0).astype(np.int)
        train_labels2=(train_set[:,0]==1).astype(np.int)
        train_labels3=(train_set[:,0]==2).astype(np.int)
        # print(len(X))
        # print(len(train_labels1))
        test_labels1=(test_set[:,0]==0).astype(np.int)
        test_labels2=(test_set[:,0]==1).astype(np.int)
        test_labels3=(test_set[:,0]==2).astype(np.int)
        # print(test_labels2)
        # print(test_labels3)
```

```
In [8]: # Y=(Y==2).astype(np.int) # to map true and false to 1 and 0 respectively
        # print(Y)
```

5 Nearest Neighbours

5.1 Training time - Nearest Neighours is zero

```
In [9]: predicted_labels1=[]
        predicted_labels2=[]
        predicted_labels3=[]
        for i in range(len(test_data)):
            # euclidean distance
            minimum_distance=((np.dot(test_data[i],test_data[i]))-2*(np.dot(test_data[i],X[0]))+
```

```
            closest_neighbour1=train_labels1[0]
            closest_neighbour2=train_labels2[0]
            closest_neighbour3=train_labels3[0]
            for j in range(1,len(X)):
                # euclidean distance
                distance=((np.dot(test_data[i],test_data[i]))-2*(np.dot(test_data[i],X[j]))+(np.
                if(distance < minimum_distance):
                    minimum_distance=distance
                    closest_neighbour1=train_labels1[j]
                    closest_neighbour2=train_labels2[j]
                    closest_neighbour3=train_labels3[j]
            predicted_labels1.append(closest_neighbour1)
            predicted_labels2.append(closest_neighbour2)
            predicted_labels3.append(closest_neighbour3)
        # print(predicted_labels1)
```

5.2 Accuracy score - Nearest Neighbours

```
In [10]: print("Accuracy for Iris-Setosa: "+str(accuracy_score(test_labels1,predicted_labels1)))
         print("Accuracy for Iris-Versicolour: "+str(accuracy_score(test_labels2,predicted_label
         print("Accuracy for Iris-Verginica: "+str(accuracy_score(test_labels3,predicted_labels3
```

```
Accuracy for Iris-Setosa: 1.0
Accuracy for Iris-Versicolour: 1.0
Accuracy for Iris-Verginica: 1.0
```

6 Naive Bayes Classifier

```
In [11]: # Assuming data is fitted to a Gaussian
         def probability(mean, std, x):
             exponential=np.exp(-1*(x-mean)**2/(2*(std**2)))
             return ((1/(std*((22/7.0)**0.5)))*(exponential))
```

```
In [12]: # Fitting Gausian
         def gaussian_parameters(X):
             mean=np.mean(X,axis=0)
             std=np.std(X,axis=0)
             return (mean,std)
```

6.1 Training Time - Naive Bayes Classifier

```
In [13]: number_of_classes=len(np.unique(iris.target))
         train_labels=np.concatenate(([train_labels1], [train_labels2], [train_labels3]), axis=0
         predicted_labels=[]
         totalTrainingTime=[]
         for i in range(number_of_classes):
             data_class1=[X[j] for j in range(len(train_labels[i])) if train_labels[i][j]==1] #
```

```
            data_class2=[X[j] for j in range(len(train_labels[i])) if train_labels[i][j]==0] #
            start_time = time.time()
            (mean_class1,std_class1)=gaussian_parameters(data_class1) # get each features gauss
            (mean_class2,std_class2)=gaussian_parameters(data_class2) # get each features gauss
            end_time = time.time()
            totalTrainingTime.append(end_time-start_time)
            total_class1=0
            for j in range(len(train_labels[i])):
                if(train_labels[i][j]==1):
                    total_class1=total_class1+1
            class1_probability=float(total_class1)/len(train_labels[i])
            class2_probability=1-class1_probability
            class_predicted_labels=[]
            for j in range(len(test_data)):
                probability_class1=1
                probability_class2=1
                for k in range(len(test_data[j])):
                    probability_class1=probability_class1*probability(mean_class1[k],std_class1
                    probability_class2=probability_class2*probability(mean_class2[k],std_class2
                probability_class1=probability_class1*class1_probability
                probability_class2=probability_class2*class2_probability
                if(probability_class1>probability_class2):
                    class_predicted_labels.append(1)
                else:
                    class_predicted_labels.append(0)
            predicted_labels.append(class_predicted_labels)
        print("Training time for Iris-Setosa: "+str(totalTrainingTime[0]))
        print("Training time for Iris-Versicolour: "+str(totalTrainingTime[1]))
        print("Training time for Iris-Verginica: "+str(totalTrainingTime[2]))
```

```
Training time for Iris-Setosa: 0.00044107437133789062
Training time for Iris-Versicolour: 0.0017528533935546875
Training time for Iris-Verginica: 0.0003819465637207031
```

6.2 Accuracy score - Naive Bayes Classifier

```
In [14]: print("Accuracy for Iris-Setosa: "+str(accuracy_score(test_labels1,predicted_labels[0])
         print("Accuracy for Iris-Versicolour: "+str(accuracy_score(test_labels2,predicted_label
         print("Accuracy for Iris-Verginica: "+str(accuracy_score(test_labels3,predicted_labels[
```

```
Accuracy for Iris-Setosa: 1.0
Accuracy for Iris-Versicolour: 1.0
Accuracy for Iris-Verginica: 0.9777777777777777
```

7 Logistic Regression - Gradient Descent

Create a copy of features of test data and insert value "1" as first feature in every data point of test_data

```
In [15]: X_data=np.copy(X)
         X_data=np.insert(X_data, 0, values=[1], axis=1)
```

```
In [16]: def sigmoid(z):
             return 1.0/(1+np.exp(-1*z))
         def gradient_descent_logistic_regression(X_data,Y,learning_rate,number_iterations):
             theta=np.zeros(X_data.shape[1])
             for i in range(number_iterations):
                 z=np.dot(X_data,theta)
                 p=sigmoid(z)
                 gradient=np.dot(X_data.T, (p - Y)) / Y.size
                 theta=theta-learning_rate*gradient
             return theta
```

7.1 Training Time - Logistic Regression (Gradient Descent)

```
In [17]: learning_rate=0.1
         number_iterations=3000
         start_time = time.time()
         theta1=gradient_descent_logistic_regression(X_data,train_labels1,learning_rate,number_i
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Setosa: "+str(training_time))

         start_time = time.time()
         theta2=gradient_descent_logistic_regression(X_data,train_labels2,learning_rate,number_i
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Versicolour: "+str(training_time))


         start_time = time.time()
         theta3=gradient_descent_logistic_regression(X_data,train_labels3,learning_rate,number_i
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Verginica: "+str(training_time))
```

```
Training time for Iris-Setosa: 0.051275014877319336
Training time for Iris-Versicolour: 0.04697895050048828
Training time for Iris-Verginica: 0.03890109062194824
```

```
In [18]: test_data_new=np.copy(test_data)
         test_data_new=np.insert(test_data_new, 0, values=[1], axis=1);
```

```
predicted_labels1=[]
predicted_labels2=[]
predicted_labels3=[]
for i in range(len(test_data_new)):
    if(sigmoid(np.dot(test_data_new[i],theta1))>0.5):
        predicted_labels1.append(1)
    else:
        predicted_labels1.append(0)
    if(sigmoid(np.dot(test_data_new[i],theta2))>0.5):
        predicted_labels2.append(1)
    else:
        predicted_labels2.append(0)
    if(sigmoid(np.dot(test_data_new[i],theta3))>0.5):
        predicted_labels3.append(1)
    else:
        predicted_labels3.append(0)
# print(predicted_labels)
```

7.2 Accuracy score - Logistic Regression (Gradient Descent)

```
In [19]: print("Accuracy for Iris-Setosa: "+str(accuracy_score(test_labels1,predicted_labels1)))
         print("Accuracy for Iris-Versicolour: "+str(accuracy_score(test_labels2,predicted_label
         print("Accuracy for Iris-Verginica: "+str(accuracy_score(test_labels3,predicted_labels3
```

```
Accuracy for Iris-Setosa: 1.0
Accuracy for Iris-Versicolour: 0.6888888888888889
Accuracy for Iris-Verginica: 1.0
```

8 Logistic Regression - Newton's method

```
In [20]: def newton_method_logistic_regression(X_data,Y,number_iterations):
             theta=np.zeros(X_data.shape[1])
             for i in range(number_iterations):
                 z=np.dot(X_data,theta)
                 p=sigmoid(z)
                 gradient=np.dot(X_data.T, (p - Y)) / Y.size
                 column=(np.ones(p.size)).T
                 prob_product = np.dot(p,column-p)
                 learning_rate=np.linalg.inv(np.dot(prob_product,np.dot(X_data.T,X_data)/ Y.size
                 theta=theta-np.dot(learning_rate,gradient)
             return theta
```

8.1 Training Time - Logistic Regression (Newton's Method)

```
In [21]: start_time = time.time()
         theta1=newton_method_logistic_regression(X_data,train_labels1,number_iterations)
         end_time = time.time()
```

```
        training_time=end_time-start_time
        print("Training time for Iris-Setosa: "+str(training_time))

        start_time = time.time()
        theta2=newton_method_logistic_regression(X_data,train_labels2,number_iterations)
        end_time = time.time()
        training_time=end_time-start_time
        print("Training time for Iris-Versicolour: "+str(training_time))

        start_time = time.time()
        theta3=newton_method_logistic_regression(X_data,train_labels3,number_iterations)
        end_time = time.time()
        training_time=end_time-start_time
        print("Training time for Iris-Verginica: "+str(training_time))
```

```
Training time for Iris-Setosa: 0.159376859664917
Training time for Iris-Versicolour: 0.1267549991607666
Training time for Iris-Verginica: 0.16057109832763672
```

```
In [22]: predicted_labels1=[]
         predicted_labels2=[]
         predicted_labels3=[]
         for i in range(len(test_data_new)):
             if(sigmoid(np.dot(test_data_new[i],theta1))>0.5):
                 predicted_labels1.append(1)
             else:
                 predicted_labels1.append(0)
             if(sigmoid(np.dot(test_data_new[i],theta2))>0.5):
                 predicted_labels2.append(1)
             else:
                 predicted_labels2.append(0)
             if(sigmoid(np.dot(test_data_new[i],theta3))>0.5):
                 predicted_labels3.append(1)
             else:
                 predicted_labels3.append(0)
```

8.2 Accuracy score - Logistic Regression (Newton's method)

```
In [23]: print("Accuracy for Iris-Setosa: "+str(accuracy_score(test_labels1,predicted_labels1)))
         print("Accuracy for Iris-Versicolour: "+str(accuracy_score(test_labels2,predicted_label
         print("Accuracy for Iris-Verginica: "+str(accuracy_score(test_labels3,predicted_labels3
```

```
Accuracy for Iris-Setosa: 1.0
Accuracy for Iris-Versicolour: 0.7111111111111111
Accuracy for Iris-Verginica: 1.0
```

9 Logistic Regression (Library)

9.1 Training Time - Logistic Regression (Library)

```
In [24]: logistic_regression=LogisticRegression()
         start_time = time.time()
         logistic_regression.fit(X,train_labels1)
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Setosa: "+str(training_time))
         predicted_labels1=logistic_regression.predict(test_data) # prediction of labels for tes
```

```
Training time for Iris-Setosa: 0.001950979232788086
```

```
In [25]: start_time = time.time()
         logistic_regression.fit(X,train_labels2)
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Versicolour: "+str(training_time))
         predicted_labels2=logistic_regression.predict(test_data) # prediction of labels for tes
```

```
Training time for Iris-Versicolour: 0.0011680126190185547
```

```
In [26]: start_time = time.time()
         logistic_regression.fit(X,train_labels3)
         end_time = time.time()
         training_time=end_time-start_time
         print("Training time for Iris-Verginica: "+str(training_time))
         predicted_labels3=logistic_regression.predict(test_data) # prediction of labels for tes
```

```
Training time for Iris-Verginica: 0.0012712478637695312
```

9.2 Accuracy score - Logistic Regression (library)

```
In [27]: print("Accuracy for Iris-Setosa: "+str(accuracy_score(test_labels1,predicted_labels1)))
         print("Accuracy for Iris-Versicolour: "+str(accuracy_score(test_labels2,predicted_label
         print("Accuracy for Iris-Verginica: "+str(accuracy_score(test_labels3,predicted_labels3
```

```
Accuracy for Iris-Setosa: 1.0
Accuracy for Iris-Versicolour: 0.7111111111111111
Accuracy for Iris-Verginica: 0.9555555555555556
```