Gabriel Van Dreel
800886655
November 6, 2021
ECGR 4105-C01

# Homework 3

The source code developed for homework 3 was uploaded to the Github repository at the following link:

https://github.com/Gabrielvd616/ECGR4105/tree/main/Homework3

## Problem 1

- The gradient descent training and cost evaluation for the logistic regression binary classifier predicting whether an individual had breast cancer was developed with consideration of the thirty input variables provided in the load_breast_cancer object. Using an 80% and 20% split of the labelled data between training and evaluation (test) sets, the model was trained and evaluated with both normalization (MIN MAX scaling) and standardization of the input variables. The logistic regression model was evaluated using the test data to obtain plots of the model cost and classification accuracy with respect to the number of iterations performed in the regression's gradient descent. To explictly evaluate the cost and accuracy over the iterations of the gradient descent, the logistic regression for only the first problem was implemented using the gradient_descent function. The accuracy of the model was calculated and recorded in section [1] using the gradient descent function applied with the test data. The training and test cost history of the model were plotted with respect to the number of iterations as shown in section [2]. The training and test accuracy of the model were also plotted with respect to the number of iterations as shown in section [3].

  The logistic regression was observed to be possible to train over the thirty input variables considered. Accordingly, the logistic regression model was observed to be effective as it had a prediction of accuracy of $0.9649$ as reflected in the first iteration of the test accuracy plot.

## Problem 2

- The gradient descent training and cost evaluation for the logistic regression binary classifier predicting whether an individual had breast cancer was repeated with principal component analysis (PCA) feature extraction of the input data. Using an 80% and 20% split of the labelled extracted data between training and test sets, independent trainings of the logistic regression were performed using increasing numbers of successive principal components starting with the first. The accuracy, precision, and recall were calculated using the test data for each independent training such that the number of principal components $K$ that resulted in the greatest classification accuracy could be obtained. The minimal value of $K$ resulting in the greatest accuracy was calculated and recorded along with its corresponding model's accuracy, precision, and recall in section [4]. The accuracies, precisions, and recalls of each model trained on $K$ principal components were plotted with respected to $K$ as shown in section [5].

Given the relatively high accuracy, precision, and recall of the logistic regression model trained on $K = 6$ principal components, the logistic regression model using PCA feature extraction was observed to perform significantly better than the model that did not use feature extraction. The model likely stagnated or decreased in performance for values of $K > 6$ because many of the input variables represented in higher $K$ principal components explained the model output less than others represented in lower $K$ principal components. As a result, the inclusion of too many principal components probably introduced noise and error into the logistic regression models trained on many more principal components.

## Problem 3

- The analysis of the training data and cost evaluation for the naïve Bayes binary classifier predicting whether an individual had breast cancer was developed with linear discriminant analysis (LDA) feature extraction of the input data and its labels. Using an 80% and 20% split of the labelled extracted data between training and test sets, the model was set to analyze the model features depicted in the training set and evaluated using the test set. The accuracy, precision, and recall of the model were calculated and recorded in section [6].

  The accuracy and precision of the naïve Bayes model using LDA feature extraction were observed to be significantly less than those of the logistic regression model using PCA feature extraction with $K = 6$ principal components. However, the two models were observed to have identical recall, which suggests that the naïve Bayes model is less generalizable given its high variation in evaluation metrics even with the use of LDA feature extraction. Interestingly, the accuracy of the naïve Bayes model using LDA feature extraction was observed to be equal to that of the logistic regression model not using any feature extraction. Given the equal accuracy of these two models and the increased accuracy of the logistic regression model using PCA feature extraction, the PCA feature extraction was observed to improve the performance of the logistic regression model for binary classification.

## Problem 4

- The gradient descent training and cost evaluation for the logistic regression binary classifier predicting whether an individual had breast cancer was repeated with LDA feature extraction of the input data and its labels. Using an 80% and 20% split of the labelled extracted data between training and test sets, the model was trained using the feature projections selected with the linear discriminant function and evaluated using the test set. The accuracy, precision, and recall of the model were calculated and recorded in section [7].

  The accuracy and precision of the logistic regression model using LDA feature extraction were observed to be less than those of the logistic regression model using PCA feature extraction but greater than those of the naïve Bayes model using LDA feature extraction. On the other hand, the recall of the logistic regression model using LDA feature extraction was observed to be less than that of both the logistic regression model using PCA feature extraction and naïve

Bayes model using LDA feature extraction. Overall, the logistic regression model using LDA feature extraction was observed to perform better than the naïve Bayes model using LDA feature extraction because it had lower variation in its evaluation metrics as well as higher accuracy and precision, which is likely a reflection of the generative behavior of naïve Bayes classifiers. However, the logistic regression model using LDA feature extraction was observed to perform worse than the logistic regression model using PCA feature extraction because it had lower accuracy, precision, and recall.

As a result, the PCA feature extraction was observed to have higher performance than the LDA feature extraction for the logistic regression models likely because the number of considered classes was small. Specifically, the dimension of the extracted features for the LDA was restricted to $1$ because it could be no greater than one less than the number of considered classes. Applying this linear transformation with the standard matrix $W$ having a column space dimension (i.e. rank) of $1$ therefore results in a much more significant loss of information that does PCA with its $K = 6$ principal components. LDA may significantly improve the classification performance of a logistic regression model or naïve Bayes model and possibly outperform PCA if it is used with more classes, but evidently does not do so with only two considered classes.

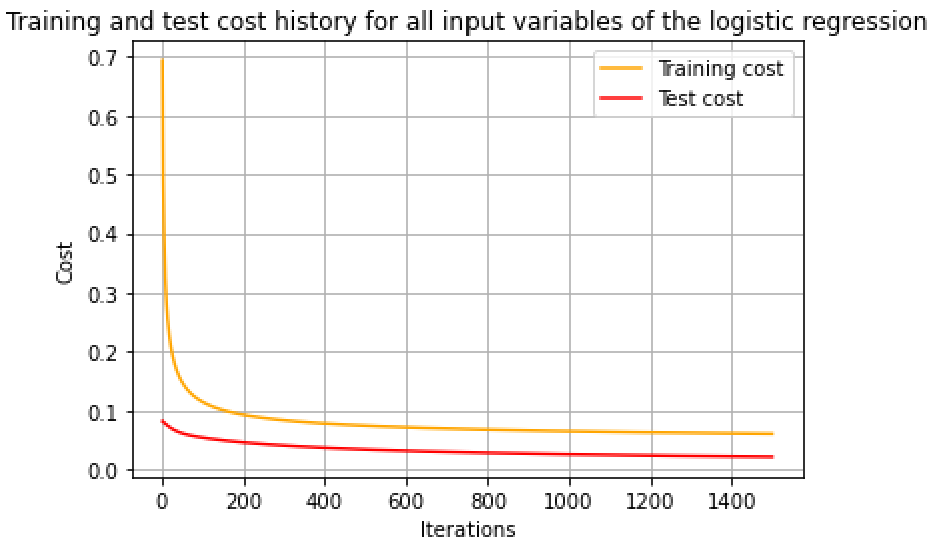
In [1]:

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler

'''
Computes gradient descent model parameters and cost for logistic regression

Input parameters:
x : m x n np array of training samples where m is the number of training samples and n
    is the number of model parameters
y : m x 1 np array of training labels
theta : 1 x n np array of model parameters
alpha : scalar value for learning rate
iterations : scalar value for the number of iterations

Output parameters:
theta : 1 x n np array of final model parameters
cost_history : 1 x iterations array of cost values for each iteration
acc_history : 1 x iterations array of accuracy for each iteration
'''


def gradient_descent(x, y, theta, alpha, iterations):
    cost_history = np.zeros(iterations)
    acc_history = np.zeros(iterations)
    m = x.shape[0]
    x = np.hstack((np.ones((m, 1)), x.reshape(m, n)))

    for i in range(iterations):
        # Computes ML model prediction using column vector theta and x values with
        # matrix vector product
        predictions = np.divide(1, 1 + np.exp(-1 * x.dot(theta)))
        theta = theta - (alpha / m) * x.transpose().dot(predictions - y)

        # Computes value of cost function J for each iteration
        log1 = np.multiply(y, np.log(predictions))
        log2 = np.multiply(1 - y, np.log(1 - predictions))
        cost_history[i] = -1 / m * np.sum(log1 + log2)

        # Computes accuracy for each iteration
        acc_history[i] = (m - np.sum(np.abs(np.round(predictions) - y))) / m

    return theta, cost_history, acc_history


# Problem 1
# Loads breast labelled training data
breast = load_breast_cancer()

# Formats np array since breast object contains separate fields for data and labels
breast_data = breast.data
labels = np.reshape(breast.target, (breast_data.shape[0], 1))
df = pd.DataFrame(np.concatenate([breast_data, labels], axis=1))

n = breast_data.shape[1]
x = df.values[:, :n]
y = df.values[:, n]

# Performs MIN MAX scaling
mms = MinMaxScaler()
x = mms.fit_transform(x)

# Performs standardization
ss = StandardScaler()
x = ss.fit_transform(x)

# Performs 80% and 20% split of the labelled data into training and test sets
np.random.seed(0)
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8,
                                                    test_size=0.2,
                                                    random_state=np.random)

# Initializes theta, the learning rate alpha, and the number of iterations
theta = np.zeros(x.shape[1] + 1)
alpha = 0.07
it = 1500

# Trains ML model with all input variables and evaluates the model on the test set
theta, train_cost, train_acc = gradient_descent(x_train, y_train, theta, alpha, it)
test_cost, test_acc = gradient_descent(x_test, y_test, theta, alpha, it)[1:]

# Evaluates model using accuracy evaluation metric
print('Accuracy:', test_acc[0])
```

```
Accuracy: 0.9649122807017544
```

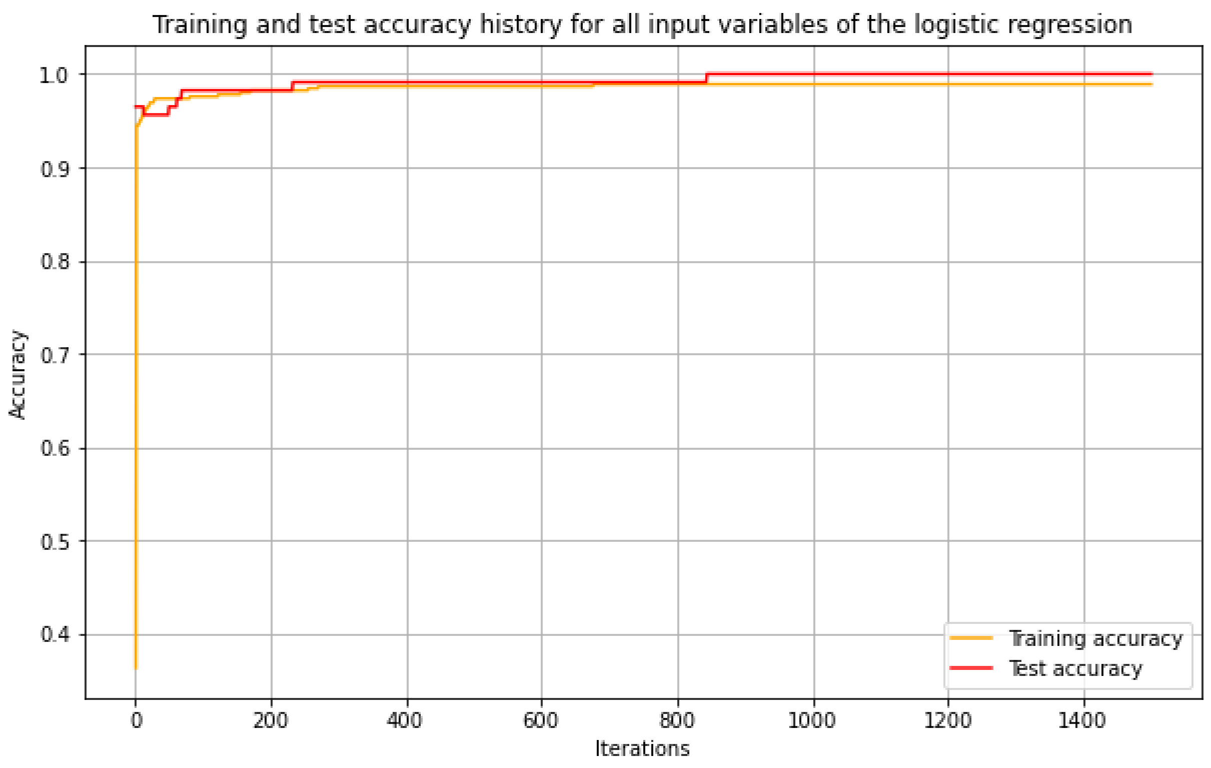
In [2]:
```
# Plots training and test cost history for the logistic regression
plt.figure(1)
plt.plot(np.linspace(1, it, it), train_cost, color='orange',
         label='Training cost')
plt.plot(np.linspace(1, it, it), test_cost, color='red',
         label='Test cost')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Training and test cost history for all input variables of the logistic'
          ' regression')
plt.legend()
```

Out[2]:
```
<matplotlib.legend.Legend at 0xbabeee0>
```



In [3]:
```
# Plots training and test accuracy history for the logistic regression
plt.figure(2)
plt.plot(np.linspace(1, it, it), train_acc, color='orange',
         label='Training accuracy')
plt.plot(np.linspace(1, it, it), test_acc, color='red',
         label='Test accuracy')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Training and test accuracy history for all input variables of the logistic'
          ' regression')
plt.legend()
```

Out[3]:
```
<matplotlib.legend.Legend at 0xc268f40>
```



In [4]:

```
# Problem 2
# Performs PCA on the data
pca = PCA()
pcs = pca.fit_transform(x)

# Performs 80% and 20% split of the labelled data into training and test sets
x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(pcs, y, train_size=0.8,
                                                            test_size=0.2,
                                                            random_state=np.random)

# Initializes evaluation metrics for logistic regression model PCA
k = pcs.shape[1]
accuracy = np.zeros(k)
precision = np.zeros(k)
recall = np.zeros(k)

# Iteratively trains and evaluates model for principal components
acc_max = 0
k_opt = 0
for i in range(k):
    # Performs logistic regression by instantiating LogisticRegression object
    lr = LogisticRegression()
    lr.fit(x_train_p[:, :i + 1], y_train_p)
    y_pred = lr.predict(x_test_p[:, :i + 1])

    # Evaluates model using accuracy, precision, and recall evaluation metrics
    accuracy[i] = metrics.accuracy_score(y_test_p, y_pred)
    precision[i] = metrics.precision_score(y_test_p, y_pred)
    recall[i] = metrics.recall_score(y_test_p, y_pred)

    if accuracy[i] > acc_max:
        acc_max = accuracy[i]
        k_opt = i + 1

# Displays optimal K and corresponding accuracy, precision, and recall
```

```
print('Optimal value of K:', k_opt)
print('Accuracy:', acc_max)
print('Precision:', precision[k_opt - 1])
print('Recall:', recall[k_opt - 1])
```

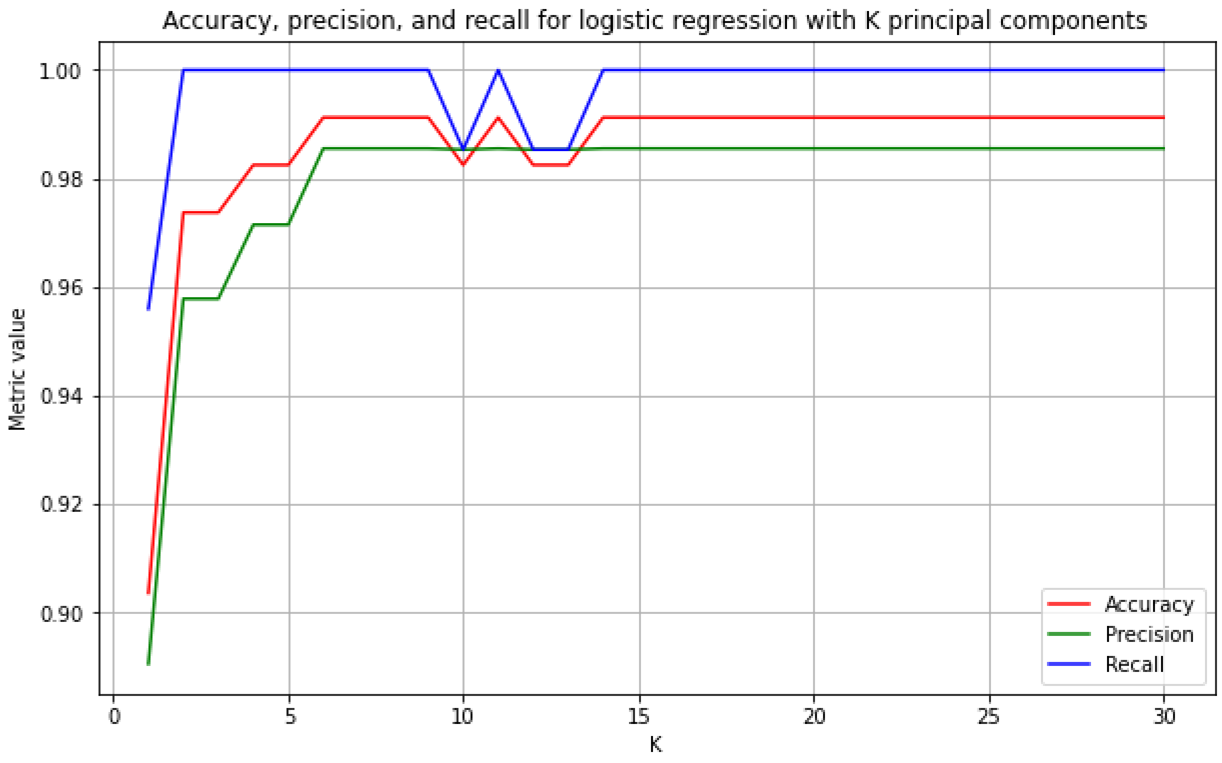
```
Optimal value of K: 6
Accuracy: 0.9912280701754386
Precision: 0.9855072463768116
Recall: 1.0
```

In [5]:

```
# Plots accuracy, precision, and recall for varying numbers of principal components
plt.figure(3)
plt.plot(np.linspace(1, k, k), accuracy, color='red',
         label='Accuracy')
plt.plot(np.linspace(1, k, k), precision, color='green',
         label='Precision')
plt.plot(np.linspace(1, k, k), recall, color='blue',
         label='Recall')
plt.rcParams['figure.figsize'] = (10, 6)
plt.grid()
plt.xlabel('K')
plt.ylabel('Metric value')
plt.title('Accuracy, precision, and recall for logistic regression with K principal'
          ' components')
plt.legend()
```

Out[5]:

```
<matplotlib.legend.Legend at 0xc725640>
```



In [6]:

```
# Problem 3
# Performs LDA on the training data
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_pred = lda.predict(x_test)
```

```python
# Evaluates naive Bayes model using accuracy, precision, and recall evaluation metrics
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
print('Precision:', metrics.precision_score(y_test, y_pred))
print('Recall:', metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
Precision: 0.9436619718309859
Recall: 1.0
```

In [7]:

```python
# Problem 4
# Transforms input data using linear discriminant function from LDA
lds = lda.fit_transform(x, y)

# Performs 80% and 20% split of the labelled data into training and test sets
x_train_l, x_test_l, y_train_l, y_test_l = train_test_split(lds, y, train_size=0.8,
                                                            test_size=0.2,
                                                            random_state=np.random)

# Performs logistic regression by instantiating LogisticRegression object
lr = LogisticRegression()
lr.fit(x_train_l, y_train_l)
y_pred = lr.predict(x_test_l)

# Evaluates model using accuracy, precision, and recall evaluation metrics
print('Accuracy:', metrics.accuracy_score(y_test_l, y_pred))
print('Precision:', metrics.precision_score(y_test_l, y_pred))
print('Recall:', metrics.recall_score(y_test_l, y_pred))
```

```
Accuracy: 0.9736842105263158
Precision: 0.971830985915493
Recall: 0.9857142857142858
```