In [1]:
```
"""
Landon Buell
Marek Petrik
CS 750.01
5 Feb 2020
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Problem 1 [30%]

This problem examines the use and assumptions of Linear Discrimiant Analysis (LDA) and Quadratic Discrimiant Analysis (QDA). We will be using the dataset Default from ISLR.

I loaded the datset into R, and the rewrote it as a CSV to use in python!

In [2]:
```
# load in data from CSV and make all numeric
default = pd.read_csv("Default.txt",sep=",",header=0,usecols=[1,2,3,4])

default = default.replace("No",0)
default = default.replace("Yes",1)
default = default.astype(float)

# print head of dataframe
default.head()
```

Out[2]:

	default	student	balance	income
0	0.0	0.0	729.526495	44361.625074
1	0.0	1.0	817.180407	12106.134700
2	0.0	0.0	1073.549164	31767.138947
3	0.0	0.0	529.250605	35704.493935
4	0.0	0.0	785.655883	38463.495879

1. Split the data into a training set (70%) and a test set (30%). Then compare the classification error of LDA, QDA, and logistic regression when predicting default as a function of features of your choice. Which method appears to work best?

In [3]:
```
from sklearn.model_selection import train_test_split

# split data set into train & test
y = default['default'].astype(int)
X = default[['student','balance','income']]

xtrain,xtest,ytrain,ytest = train_test_split(X,y,test_size=0.3)

# Create linear disc. Classifier instance
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
LDA_CLF = LDA()
LDA_CLF.fit(xtrain,ytrain)
LDA_pred = LDA_CLF.predict(xtest)

# Create Quadratic Disc. Classifier instance
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
QDA_CLF = QDA()
QDA_CLF.fit(xtrain,ytrain)
QDA_pred = QDA_CLF.predict(xtest)

# Create Logstic Regression
from sklearn.linear_model import LogisticRegression as LRC
LRC_CLF = LRC()
LRC_CLF.fit(xtrain,ytrain)
LRC_pred = LRC_CLF.predict(xtest)

from sklearn.metrics import accuracy_score
LDA_score = accuracy_score(ytest,LDA_pred)
QDA_score = accuracy_score(ytest,QDA_pred)
LRC_score = accuracy_score(ytest,LRC_pred)


print("Linear Disc. Analysis Classification Error:",LDA_score)
print("Quadratic Disc. Analysis Classification Error:",QDA_score)
print("Logistic Regression Classification Error:",LRC_score)
```

```
Linear Disc. Analysis Classification Error: 0.9746666666666667
Quadratic Disc. Analysis Classification Error: 0.9756666666666667
Logistic Regression Classification Error: 0.9686666666666667

C:\Users\Landon\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:
432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
```

All three models produce very similar resuts for this particular problem. However, I would favor the QDA model due to it's general flexibility.

2. Report the confusion table for each classification method. Make sure to label which dimension is the predicted class and which one is the true class. What do you observe?

In [4]:
```
# build 3 confusion matricies
from sklearn.metrics import confusion_matrix

print("Linear Disc. Analysis Confusion Matrix:")
LDA_confmat = confusion_matrix(ytest,LDA_pred)
print(LDA_confmat)

print("Quadratic Disc. Analysis Confusion Matrix:")
QDA_confmat = confusion_matrix(ytest,QDA_pred)
print(QDA_confmat)

print("Logisitc Regression Confusion Matrix:")
LRC_confmat = confusion_matrix(ytest,LRC_pred)
print(LRC_confmat)
```

```
Linear Disc. Analysis Confusion Matrix:
[[2899    7]
 [  69   25]]
Quadratic Disc. Analysis Confusion Matrix:
[[2899    7]
 [  66   28]]
Logisitc Regression Confusion Matrix:
[[2906    0]
 [  94    0]]
```

Each confusion matrix has a string entry at index (1,1) - the True Positive entry. This means that It is excellent at correctly identifying those who belong into class '0': i.e. those who will NOT default on their loans. However, entry (2,2) seems to be very weak, this means that each method is poor at identifying any instance in class "1": those who did default on their loans. Furthermore, the entry (2,1) is generally much stronger, indicating that that model predicted the person would not default, even though they did. We usually want a confusion matrix with a strong main diagonal and weaker off-diagonals. These matricies do not grant us that so well.

3. Are the LDA assumptions satisfied when predicting default as a function of balance only (i.e default ~ balance)? You can use qqnorm and qqline to examine whether the conditional class distributions are normally distributed. Also examine standard deviations of the class distributions. Are the QDA assumptions satisfied?

4. Would you ever want to use LDA in place of QDA even when you suspect that some of the assumptions are violated (e.g. different conditional standard deviations) for LDA? Hint: Check out TidyVerse for a collection of packages that can help with data manipulation. And see the Rstudio cheatsheets for a convenient and concise reference to the methods. This is entirely optional!

Problem 2 [30%]

Using the MNIST dataset, fit classification models in order to predict the digit 1 (vs all others).

1. Compare the classification error for each one of these methods:

The procedure that I am useing here is adapted from "Hands on Machine Learning w/ Sci-kit Learn & TensorFlow" By Aurelion Geron, Chapter 3

In [5]:
```
from sklearn.datasets import fetch_openml
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Load in MNIST data set
mnist = fetch_openml('mnist_784',version=1)
print(mnist.keys())
X,y = mnist['data'],mnist['target'].astype(np.uint8)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details', 'categorie
s', 'url'])
```

In [6]:
```
# split into train-test sets
xtrain,xtest,ytrain,ytest = X[:60000],X[60000:],y[:60000],y[60000:]
# isolate 1's in targets set
ytrain1,ytest1 = (ytrain == 1),(ytest == 1)
```

2. Logistic regression

In [7]:
```
from sklearn.linear_model import LogisticRegression

# create & train LR model
LR_model = LogisticRegression()
LR_model.fit(xtrain,ytrain1)
```

```
C:\Users\Landon\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:
432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
```

Out[7]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [8]:
```
# test the model & build confusion matrix
LR_predict = LR_model.predict(xtest)
LR_confmat = confusion_matrix(ytest1,LR_predict)
# print matrix
print("Confusion Matrix for Logisitic Regression:")
print(LR_confmat)
print("\nClassification Report:\n",classification_report(ytest1,LR_predict))
```

```
Confusion Matrix for Logisitic Regression:
[[8822   43]
 [  30 1105]]

Classification Report:
               precision    recall  f1-score   support

       False       1.00      1.00      1.00      8865
        True       0.96      0.97      0.97      1135

    accuracy                           0.99     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.99      0.99      0.99     10000
```

Of the 10,000 samples tested, the logisitic regression classifier correctly classifier ~8800 of them as 1's and ~ 1100 of them as not 1's. This is shown in the confusion matrx above, notice it's strng diagonal entries. By comparison, the off-diagonals have quite small enties. Additionally, the precision, recall and F1-Scores of this classifier are all quite strong. The model seems to do a good jonb a finding all of the 1's and extracting all of the not 1's

3. K-NN (with 2 reasonable choices of k)

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier as KNC

# KNN Classifier for K = 10
K = 10
KNN_Clf = KNC(n_neighbors=K)
KNN_Clf.fit(xtrain,ytrain1)
```

In [ ]:
```
# Test model & build confusion matrix for K = 10
KNN_pred = KNN_Clf.predict(xtest)
KNN_confmat = confusion_matrix(ytest1,KNN_pred)
# print matrix
print("Confusion Matrix for 10-Nearest Neighbors Classifier:")
print(KNN_confmat)
print("\nClassification Report:\n",classification_report(ytest1,KNN_pred))
```

In [ ]:
```
# KNN Classifier for K = 5
K = 5
KNN_Clf = KNC(n_neighbors=K)
KNN_Clf.fit(xtrain,ytrain1)
```

In [ ]:
```
# Test model & build confusion matrix for K = 50
KNN_pred = KNN_Clf.predict(xtest)
KNN_confmat = confusion_matrix(ytest1,KNN_pred)
# print matrix
print("Confusion Matrix for 50-Nearest Neighbors Classifier:")
print(KNN_confmat)
print("\nClassification Report:\n",classification_report(ytest1,KNN_pred))
```

4. LDA

In [9]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# create and train classifier
LDA_Clf = LDA()
LDA_Clf.fit(xtrain,ytrain1)

# test & evaluate model
LDA_pred = LDA_Clf.predict(xtest)
LDA_confmat = confusion_matrix(ytest1,LDA_pred)

# print matrix & report
print("Confusion Matrix for 50-Nearest Neighbors Classifier:")
print(LDA_confmat)
print("\nClassification Report:\n",classification_report(ytest1,LDA_pred))
```

```
C:\Users\Landon\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:
388: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")

Confusion Matrix for 50-Nearest Neighbors Classifier:
[[8742  123]
 [  71 1064]]

Classification Report:
              precision    recall  f1-score   support

       False       0.99      0.99      0.99      8865
        True       0.90      0.94      0.92      1135

    accuracy                           0.98     10000
   macro avg       0.94      0.96      0.95     10000
weighted avg       0.98      0.98      0.98     10000
```

Linear Discriminant Analysis seems to do a very nice job with this classiciation task. The diagonal entried of the confusion matrix are stronger than the off-diagonals and the precision, recall and f1 scores are all very high!

Problem 3 [20%]

Logistic regression uses the logistic function to predict class probabilities:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Which is Equivalent to the linear model for the predicition of the log-odds:

$$\log\left(\frac{p(X)}{1-p(x)}\right) = \beta_0 + \beta_1 X$$

We can show equivalence by first making the variable substitution such that : $\alpha = \beta_0 + \beta_1 X$ to ease reading. Thus, now we can write our Log-odds function given the defintion of $p(X)$ from logisitc regression:

$$\log\left(\frac{\frac{e^{\alpha}}{1+e^{\alpha}}}{1 + \frac{e^{\alpha}}{1+e^{\alpha}}}\right)$$

We can then modify the denominator of the expression, and re-write 1 as $\frac{1+e^{\alpha}}{1+e^{\alpha}}$ to allow for the combination of fraction terms with a common denominator

Thus, now our log-odds equation reads:

$$\log\left(\frac{\frac{e^{\alpha}}{1+e^{\alpha}}}{\frac{1+e^{\alpha}}{1+e^{\alpha}} + \frac{e^{\alpha}}{1+e^{\alpha}}}\right)$$

To which we combine common denominators:

$$\log\left(\frac{\frac{e^{\alpha}}{1+e^{\alpha}}}{\frac{1+e^{\alpha}-e^{\alpha}}{1+e^{\alpha}}}\right)$$

Which in turn becomes:

$$\log\left(\frac{\frac{e^{\alpha}}{1+e^{\alpha}}}{\frac{1}{1+e^{\alpha}}}\right)$$

We can now rearange the argument inside the $\log()$ operation by taking advantage of the relationship between multiplication and division:

$$\log\left(\left(\frac{e^{\alpha}}{1+e^{\alpha}}\right)\left(1+e^{\alpha}\right)\right)$$

We can then cross out the cancelling terms, and simply to arrive at:

$$\log\left(e^{\alpha}\right)$$

Which by definition is equivalent to α, which we have defined to be $\alpha = \beta_0 + \beta_1 X$. Thus The expressions are mathematically equivalent.

Problem 4 [20%]

This problem examines the differences between LDA and QDA.

1. For an arbitrary training set, would you expect for LDA or QDA to work better on the training set?

In an arbitrary training, set I would expect QDA to function a little bit better than LDA. In the real world, there are very few relations between variables that are truly linear, this a quadratic approach would seem more generally fitting. Also, in some case of an actual linear relationship, the β_2 coefficient would go close to zero, indiciating a linear approach is more useful for that particular case. Similarly, the overall flexibility of a QDA would suite a more arbitrary data set.

2. If the Bayes decision boundary between the two classes is linear, would you expect LDA or QDA to work better on the training set? What about the test set?

Is the decision boundary is linear, then the linear model, LDA would seem to be the far more reasonable model to use. Thus it would produce a better fit on the training data set, and be a better prediction model for the testing data set as well. Using a QDA algorithm also has the potential to apply an overfit to the data.

3. As the sample size increases, do you expect the prediction accuracy of QDA with respect to LDA increase or decrease

With an increasing sample size, I would expect the prediciton accuracy of the QDA to increase slower than that of LDA. Some of the assumptions made by an LDA classifier cause a high bias to occur in the model (James, 150). By choosing a QDA for a larger set, the variance of the classifier is not of great concern. Thus the prediction accuracy of QDA decreases with respect to LDa for larger and larger sample sizes

4. True or False: Even if the Bayesian decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is more flexible and can model a linear decision boundary. Justify your answer.

False. Sometimes, if a relationship is linear, the approximation of it should be a linear model. In the case of a quadratic model, like QDA, the best best fit curve may actually produce and overfit, or be useless at the bounds of the expression. Even if QDA is more flexible, it has the potential to misbehave in these senses, cause misinformation to arise in the model.