Summative2

January 6, 2023

1 Q1

```
In [42]: ## Load dataset

         from sklearn import datasets
         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
         breast_cancer_data = datasets.load_breast_cancer()
         df = breast_cancer_data.data

         labels = breast_cancer_data.target
```

```
In [43]: # Reshaping labels to append to dataframe

         labels = np.reshape(labels,(569,1))
```

```
In [44]: breast_cancer_df = np.concatenate([df, labels], axis=1)
```

```
In [45]: # converting to dataframe

         breast_cancer_df = pd.DataFrame(breast_cancer_df)
```

```
In [46]: features = breast_cancer_data.feature_names
```

```
In [47]: # Adding label column name

         features_labels = np.append(features, "label")

         #Adding the labels to the dataframe columns
         breast_cancer_df.columns = features_labels
```

```
In [58]: #Separate features and target variables
         X = breast_cancer_df.loc[:, features].values
         y = breast_cancer_df.loc[:, 'label'].values

         #Normalising data using StandardScaler
         from sklearn.preprocessing import StandardScaler
         #x = breast_cancer_df.loc[:, features].values
         #x = StandardScaler().fit_transform(x)
```

In [84]:
```python
#3 Splitting the X and Y into the
# Training set and Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

In [85]:
```python
#4 performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [77]:
```python
#5 Applying PCA function on training
# and testing set of 2 component
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

In [80]:
```python
# Saving to new dataframe

df_pca2 = pd.DataFrame(X_train, columns = ['PC1', 'PC2'])

# to csv file
df_pca2.to_csv("pca_2Components.csv", index=False)
```

In [79]:
```python
df_pca2
```

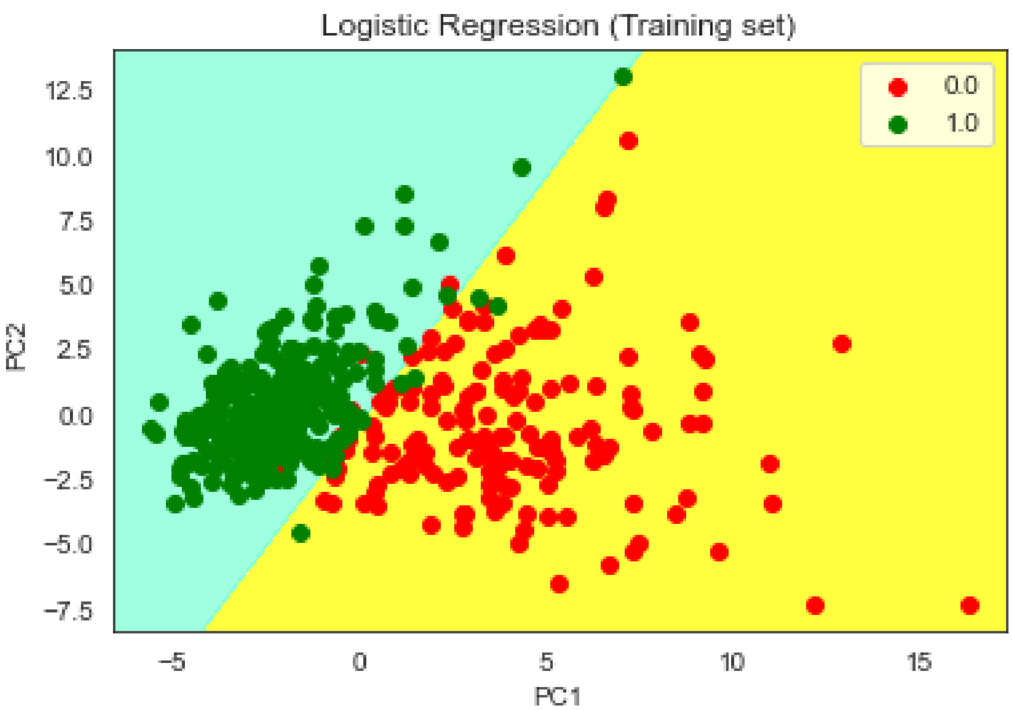
Out[79]:
```
          PC1       PC2
0   -3.039161  1.106670
1   -2.282314  0.400452
2   -1.084095 -1.995444
3   -2.561100  0.179338
4   -2.848305 -1.017616
..        ...       ...
450  0.081023 -3.420671
451  6.513019  7.996828
452 -3.170461  0.516009
453 -5.611046 -0.538805
454 -1.016371  1.395352

[455 rows x 2 columns]
```

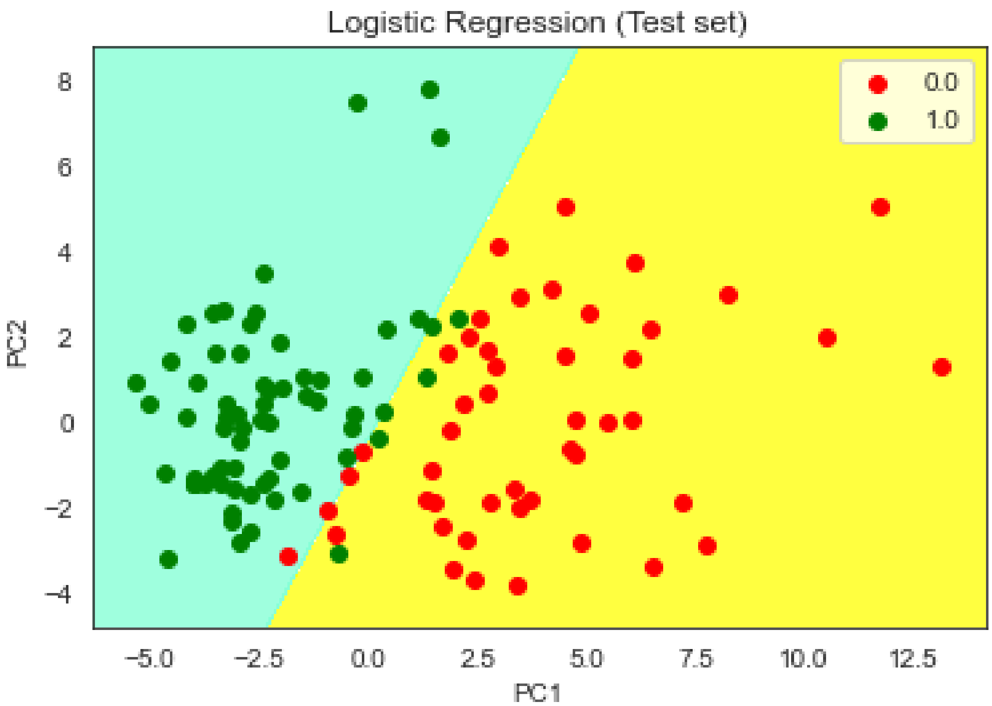
In [52]:
```python
#6 Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
#7 Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)

#8 making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

#9 Predicting the training set
# result through scatter plot
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                                                  X2.ravel()]).T).reshape(X1.shape), a
             cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend
# show scatter plot
plt.show()


#10 Visualising the Test set results through scatter plot
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,stop = X_set[:, 0].max()
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                                                  X2.ravel()]).T).reshape(X1.shape), a
             cmap = ListedColormap(('yellow', 'white', 'aquamarine')))
plt.xlim(X1.min(), X1.max())
```

```
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green', 'blue'))(i), label = j)

# title for scatter plot
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend()

# show scatter plot
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as valu
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as valu
```






```
In [86]: # Using 3 compenents

         pca = PCA(n_components = 3)
         X_train = pca.fit_transform(X_train)
         X_test = pca.transform(X_test)
         explained_variance = pca.explained_variance_ratio_

In [88]: # creating dataframe and to csv file

         df_pca3 = pd.DataFrame(X_train, columns = ['PC1', 'PC2', 'PC3'])

         # to csv file
         df_pca3.to_csv("pca_3Components.csv", index=False)

In [89]: df_pca3
```

Out[89]:

	PC1	PC2	PC3
0	-3.039161	1.106670	0.408520
1	-2.282314	0.400452	0.263275
2	-1.084095	-1.995444	-1.193536
3	-2.561100	0.179338	1.085279
4	-2.848305	-1.017616	-0.847290
..	...	...	...
450	0.081023	-3.420671	0.346485

```
451  6.513019  7.996828 -4.505699
452 -3.170461  0.516009  0.523623
453 -5.611046 -0.538805  4.127448
454 -1.016371  1.395352  1.339224

[455 rows x 3 columns]
```

The main components derived from a PCA analysis can be applied in a variety of ways in the future. Among the frequent uses are:

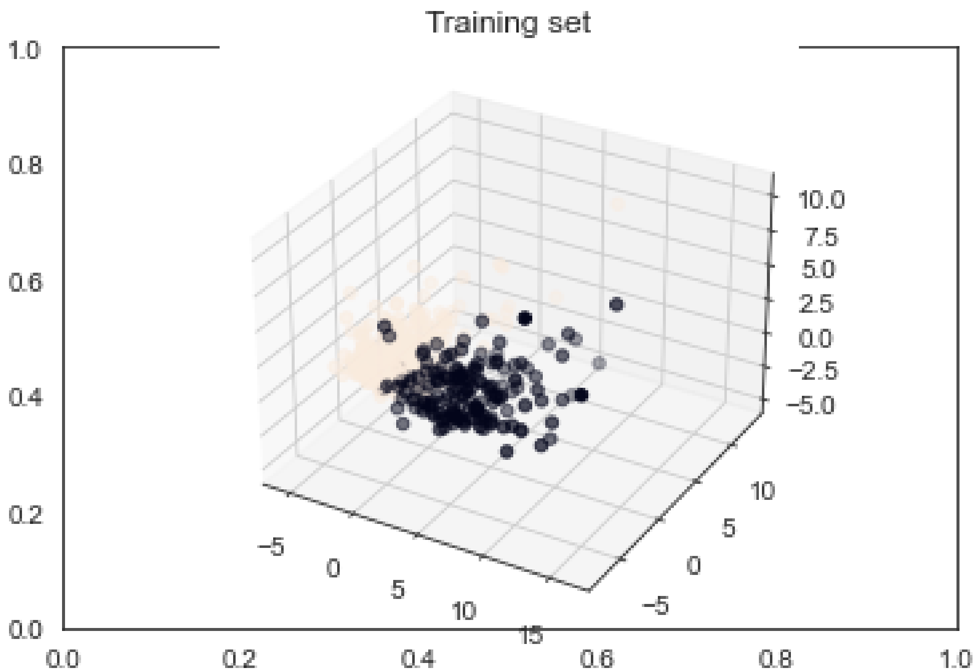
Data visualisation: It is frequently possible to produce more comprehensible visualisations of the data by condensing the dimensionality of the data to a small number of primary components. Given that it can be challenging to view data in more than three dimensions, this can be very helpful when working with high-dimensional datasets.

Data compression: Since the principle components are ranked according to how much variance in the data they explain, it is frequently possible to drastically reduce the size of the dataset by keeping only the most crucial elements. This may help with more effective data transmission or storage.

Feature selection: It can be advantageous to choose a subset of the most crucial features to include in the model while developing machine learning algorithms. The most crucial features in the data can be found by using the principle components that come from a PCA analysis. Noise reduction: Projecting the data onto the major components can frequently filter out noise and increase the signal-to-noise ratio in the data since they are orthogonal to one another and capture the most crucial elements in the data.

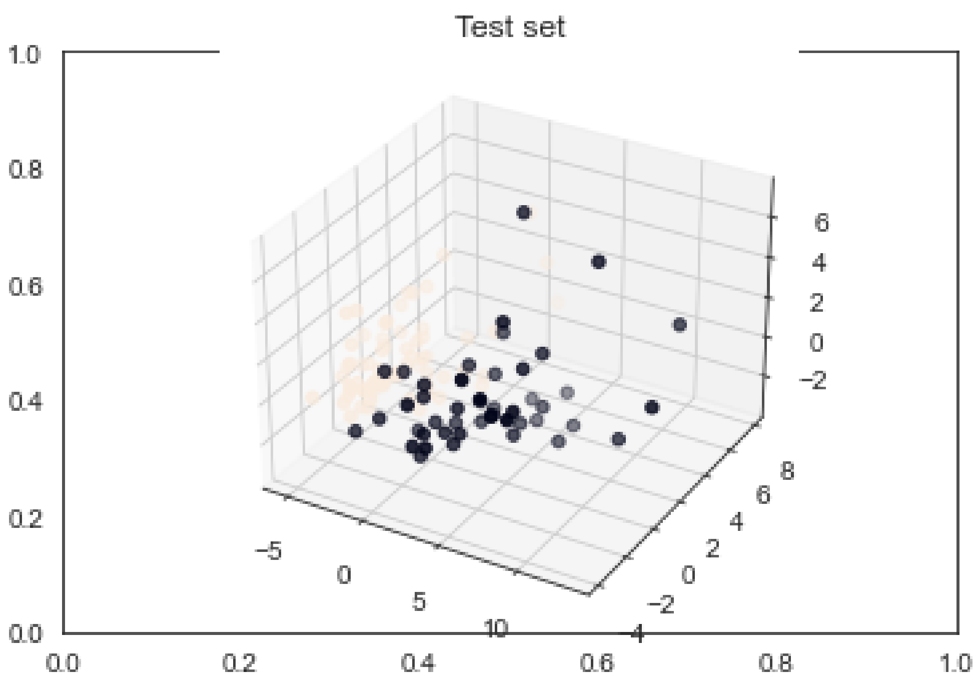
In [65]:
```
# plotting 3 componnets

from mpl_toolkits.mplot3d import Axes3D

fig =plt.figure()
plt.title("Training set")
ax = fig.add_subplot(111, projection="3d")
ax.scatter(X_train[:,0], X_train[:,1], X_train[:,2], c=y_train)
```

Out[65]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x16a06923fd0>
```




```
In [64]: fig =plt.figure()
         plt.title("Test set")
         ax = fig.add_subplot(111, projection="3d")
         ax.scatter(X_test[:,0], X_test[:,1], X_test[:,2], c=y_test)
```

```
Out[64]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x16a0688a770>
```



```
In [71]: # 95% component

         pca = PCA(n_components = 0.95)
         X_train = pca.fit_transform(X_train)
         X_test = pca.transform(X_test)
         explained_variance = pca.explained_variance_ratio_

In [72]: pca.n_components_

Out[72]: 3
```

The number of components needed to reach 95% of the explained variance is 3 which we have already plotted above

2 Q2

2.1 Neural network calculations

2.2 Steps to train a simple neural network

- Collect and preprocess the data: this includes data cleansing, formatting the data to have suitable data type and splitting these data into training and test sets.
- Define the model: Next, we must define the neural network architecture. This will include choosing the number of layers, the number of nodes in each layer, and the activation functions to use. Usually for classification problems the architecture is composed of : - A scaling layer - Tow perceptron layers - A probablistic layer
- Compile the model: After defining the model, you will need to compile it with a loss function, an optimizer, and any metrics that need to be tracked
- Train the model: train the model on the training data. This will involve providing the model with the training data and allowing it to learn the relationships between the input features and the target variables.
- Evaluate the model: After training, we need to evaluate the model's performance on the test data. This will give us a sense of how well the model is able to generalise to unseen data. For example we can switch between LogSoftMax, NLLLoss and Cross Entropy to check the difference in performance.
- Fine-tune the model: Depending on the results of the evaluation, we may want to adjust the model's architecture or hyperparameters to improve its performance. This process is known as fine-tuning.
- Make predictions: Finally, we can use the trained model to make predictions on new data.

Forward Pass

$$Sum\,h_1 = i_1 \times w_1 + i_2 \times w_3 + b_1$$

$$= (0.2 \times 0.2) + (0.4 \times 0.1) + 0.35$$

$$= 0.04 + 0.04 + 0.35$$

$$= 0.43$$

Pass the weighted sum through logistic function

$$Output\,h_1 = \frac{1}{1+e^{-Sum\,h_1}} = \frac{1}{1+e^{-0.43}} = 0.60587$$

H_2

$$Sum\,h_2 = i_1 \times w_2 + i_2 \times w_4 + b1$$

$$= (0.2 \times 0.4) + (0.4 \times 0.3) + 0.35$$

$$= 0.08 + 0.12 + 0.35$$

$$= 0.55$$

$$Output\,h_2 = \frac{1}{1+e^{-Sum\,h_2}} = \frac{1}{1+e^{-0.55}} = 0.63414$$

Using the outputs for the next layer

$$Sum\,O_1 = Output\,h_1 \times w_5 + output\,h_2 \times w_6 + b2$$

$$= (0.60587 \times 0.8) + (0.63414 \times 0.6) + 0.45$$

$$= 0.48469 + 0.380484 + 0.45$$

$$= 1.31518$$

$$Output\,o_1 = \frac{1}{1+e^{-Sum\,o_1}} = \frac{1}{1+e^{-1.31518}} = 0.788379$$

1

$$\text{Sum}_{o_2} = \text{output}_{h_1} \times w_7 + \text{output}_{h_2} \times w_8 + b_2$$

$$= \left(\overset{0.60587}{\cancel{0.62481}} \times 0.5\right) + \left(0.63414 \times 0.7\right) + 0.45$$

$$= \overset{0.302935}{\cancel{0.312405}} + 0.443898 + 0.45$$

$$= \cancel{1.206303}\ 1.196833$$

$$\text{Output}_{o_2} = \frac{1}{1+e^{-\text{Sum}_{o_2}}} = \frac{1}{1+e^{-\overset{\cancel{1.206303}}{1.19683}}} = \cancel{0.769644}$$

$$= 0.76796$$

Computing the total error

The expected outputs are $O_1 = 0.02$

$O_2 = 0.85$

$$\text{Error Formula} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

$$E_1 = \frac{1}{2}(\text{target}_1 - \text{output}_1)^2$$

$$= \frac{1}{2}(0.02 - 0.788379)^2$$

$$= 0.2952$$

$$E_2 = \frac{1}{2}(\text{target}_2 - \text{output}_2)^2$$

$$= \frac{1}{2}\left(0.85 - \overset{0.76796}{\cancel{0.769644}}\right)^2$$

$$= \cancel{0.003229}\ 0.003365$$

$$E_{total} = 0.2952 + \cancel{0.003229}\ 0.003365$$

$$= \cancel{0.2984}$$

$$0.2986$$

Backpr

Weights

$W_5, W_6,$

W_5

Usin

$+b_2$

$) + 0.45$

45

$= 0.769644$

$= 0.76796$

Backpropagation

Weights in the outer layer

W_5, W_6, W_7, W_8

W_5

Using chain rule: $\frac{\partial E_{total}}{\partial W_5} = \frac{\partial E_{total}}{\partial output_{o_1}} \times \frac{\partial output_{o_1}}{\partial sum_{o_1}} \times \frac{\partial sum_{o_1}}{\partial w_5}$

1

$E_{total} = \sum \frac{1}{2}(target - output)^2$

$$\frac{\partial total_1}{\partial output_1} = 2 \times \frac{1}{2}(target - output) \times -1$$

$$= (target - output) \times -1$$

$$= output - target = 0.788379 - 0.02 = \underline{0.768379}$$

2 : $\frac{\partial output_{o_1}}{\partial sum_{o_1}}$

$$output_{o_1} = \frac{1}{1+e^{-sum_{o_1}}}$$

$$\frac{\partial output_{o_1}}{\partial sum_{o_1}} = \sigma(x)(1-\sigma(x))$$

$$= output_{o_1}(1 - output_{o_1})$$

$$= 0.788379(1 - 0.788379)$$

$$= \underline{\underline{0.16684}}$$

3

$$Sum_{o_1} = output_{h_1} \times w_5 + output_{h_2} \times w_6 + b_2$$

$$\frac{\partial sum_{o_1}}{\partial w_5} = output_{h_1} = 0.60587$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial output_{o_1}} \times \frac{\partial output_{o_1}}{\partial sum_{o_1}} \times \frac{\partial sum_{o_1}}{\partial w_5}$$

$$= 0.768379 \times 0.16684 \times 0.60587$$

$$= 0.07767$$

W_6

$$\frac{\partial E_{total}}{\partial w_6} = \frac{\partial E_{total}}{\partial output_{o_1}} \times \frac{\partial output_{o_1}}{\partial sum_{o_1}} \times \frac{\partial sum_{o_1}}{\partial w_6}$$

$$= 0.768379 \times 0.16684 \times 0.63414$$

$$= 0.08129$$

W_7

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial output_{o_2}} \times \frac{\partial output_{o_2}}{\partial sum_{o_2}} \times \frac{\partial sum_{o_2}}{\partial w_7}$$

$$= \cancel{0.768379 \times 0.16684 \times 0.634}$$

$$\frac{\partial E_{total}}{\partial output_{o_2}} = output_{o_2} - target_2$$

$$= 0.76796 - 0.85$$

$$= \cancel{0.76796} \; -0.08204$$

$$\frac{\partial output_{o2}}{\partial sum_{o2}} = output_{o2}\,(1 - output_{o2})$$

$$= 0.76796\,(1 - 0.76796)$$

$$= \underline{0.17819}$$

$$\frac{\partial E_{total}}{\partial w_7} = \overset{-0.08204}{\cancel{0.26796}} \times 0.17819 \times 0.60587$$

$$= \cancel{0.0289} \quad \cancel{-0.000} \quad \underline{-0.00886}$$

W_8

$$= -0.08204 \times 0.17819 \times 0.63414$$

$$= \underline{-0.00927}$$

New weights

$n = 0.6$

W_5

$$new\_w_5 = w_5 - n \times \frac{\partial E_{total}}{\partial w_5}$$

$$= 0.8 - 0.6 \times 0.07767$$

$$= \underline{0.753398}$$

W_6

$$new\_w_6 = w_6 - n \times \frac{\partial E_{total}}{\partial w_6}$$

$= 0.6 - 0.6 \times 0.08129$

$= \underline{0.551226}$

mew $w_7 = w_7 - n \times \dfrac{\partial E_{total}}{\partial w_7}$

$= 0.5 - 0.6 \times -0.08204$

$= \underline{\underline{0.5492}}$

~~na~~ new_$w_8 = w_8 - n \times \dfrac{\partial E_{total}}{\partial w_8}$

$= 0.7 - 0.6(\times -0.00927)$

$= \underline{\underline{0.7056}}$

<u>Weights in hidden layer</u> (w_1, w_2, w_3, w_4)

$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial output_1} \times \frac{\partial output_{o_1}}{\partial sum_{o_1}} \times \frac{\partial sum_{o_1}}{\partial output_{h_1}} \times \frac{\partial output_{h_1}}{\partial sum_{h_1}} \times \frac{\partial sum_{h_1}}{\partial w_1}$$

$= 0.768379 \times 0.16684 \times 0.8 \times 0.23879 \times 0.2$

$= \underline{\underline{0.0048979}}$

$\dfrac{\partial E_2}{\partial w_1} = -0.08204 \times 0.178197 \times 0.5 \times 0.23879 \times 0.2$

$= \underline{\underline{-0.0003496}}$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_1}{\partial w_1} + \frac{\partial E_2}{\partial w_1}$$

$$= 0.0048979 + (-0.00034908)$$

$$= 0.0045489$$

$$\boxed{n = 0.6}$$

$$\text{new } w_1 = w_1 - n \times \frac{\partial E_{total}}{\partial w_1}$$

$$= 0.2 - 0.6 \times 0.0045489$$

$$= 0.19727066$$

W_2

$$\frac{\partial E_1}{\partial w_2} \; 0.76838 \times 0.16684 \times 0.6 \times 0.232006 \times 0.4$$
$$= 0.007138166$$

$$\frac{\partial E_1}{\partial w_2} = -0.08204 \times 0.178197 \times 0.7 \times 0.232006 \times 0.4$$

$$= -0.000949693$$

$$\text{new } w_2 = w_2 - n \times \frac{\partial E_{total}}{\partial w_2} = 0.4 - 0.5 \times 0.006188473$$
$$= 0.39691$$

$\underline{W_3}$ - Using same steps

$$\frac{\partial E_1}{\partial w_3} = 0.76838 \times 0.16684 \times 0.8 \times 0.23879 \times 0.1 =$$
$$= 0.002448963$$

$$\frac{\partial E_2}{\partial w_3} = -0.08204 \times 0.178197 \times 0.5 \times 0.23879 \times 0.1$$
$$= \cancel{-0.0024} \; -0.00017456$$

$$\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_1}{\partial w_3} + \frac{\partial E_2}{\partial w_3} = 0.002448963 - 0.00017456$$
$$= 0.002274416$$

$$\cancel{w} \, new\_w_3 = w_3 - n \times \frac{\partial E_{tot}}{\partial w_3}$$

$$= 0.1 - 0.5 \times 0.002274416$$
$$= \underline{\underline{0.0098863}}$$

$\underline{W_4}$

$$\frac{\partial E_1}{\partial w_4} = 0.76838 \times 0.16684 \times 0.6 \times 0.232006 \times 0.3$$
$$= 0.005353625$$

$$\frac{\partial E_2}{\partial w_4} = -0.08204 \times 0.178197 \times 0.7 \times 0.232006 \times 0.3$$
$$= -0.000712269$$

$$\frac{\partial E_{total}}{\partial w_4} = \frac{\partial E_1}{\partial w_4} + \frac{\partial E_2}{\partial w_4}$$

$$= 0.005353625 + -0.000712269$$

$$= \underline{0.00464135}$$

$$mew\_w4 = w_4 - n \times \frac{\partial E_{total}}{d w_4}$$

$$= 0.4 - 0.5 \times 0.004641355$$

$$= \underline{\underline{0.3976793224}}$$

2.3 Code example for train and testing a simple network

```
from keras.models import Sequential
from keras.layers import Dense
```

2.3.1 define the model

```
model = Sequential()
model.add(Dense(10, input_dim=8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

2.3.2 compile the model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

2.3.3 fit the model to the training data

```
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

2.3.4 evaluate the model on the test data

```
loss, accuracy = model.evaluate(X_test, y_test, batch_size=32)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

The input features and target variables for the training data in this example are X train and y train, while the input features and target variables for the test data are X test and y test. The model consists of two layers: an output layer with one node and a hidden layer with ten nodes. The output layer makes use of the sigmoid activation function, whereas the hidden layer makes use of the relu activation function. The binary crossentropy loss function and Adam optimizer are used in the model's construction, and it is trained using stochastic gradient descent with a batch size of 32 on the training set of data. The model is then assessed using the test data, and the accuracy and loss are printed.

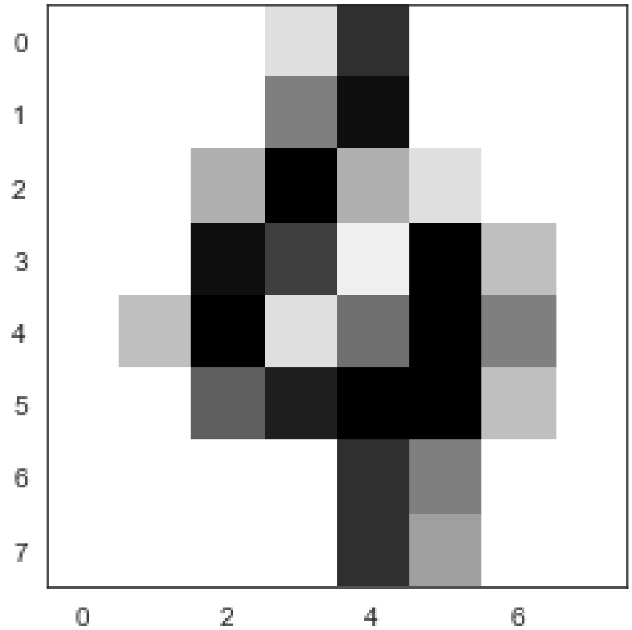
3 Q3

In [27]:
```
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
import numpy as np

# Laoding the digits dataset
digits = datasets.load_digits()

print(digits.images.shape)
print(digits.data.shape)
```

```
(1797, 8, 8)
(1797, 64)
```

In [28]:
```
#Viewing the digits within the dataset.
# The image appears to show a 4 but we can see if the model can predict this
plt.imshow(digits.images[100], cmap=plt.cm.gray_r, interpolation='nearest')
```

Out[28]: <matplotlib.image.AxesImage at 0x16a717c4430>



In [13]:
```
# Splitting the dataset into test and train sets

X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, randor
```

In [36]:
```
# Using the Support Vector Machine (SVM) classifier to classify the digits.
# First training the classifier on the training set then test it on the testinig set

# Import the SVM classifier
from sklearn.svm import SVC

# Create an SVM classifier
clf = SVC(gamma='auto')

# Train the classifier on the training set
clf.fit(X_train, y_train)

# Test the classifier on the testing set
accuracy = clf.score(X_test, y_test)

print("Test set accuracy: {:.2f}".format(accuracy))
```

```
Test set accuracy: 0.52
```

```
In [37]: ## Using RandomForest classifier

         from sklearn.ensemble import RandomForestClassifier

         clf = RandomForestClassifier()
         clf.fit(X_train, y_train)

         accuracy = clf.score(X_test, y_test)
         print("Test set accuracy: {:.2f}".format(accuracy))
```

```
Test set accuracy: 0.98
```

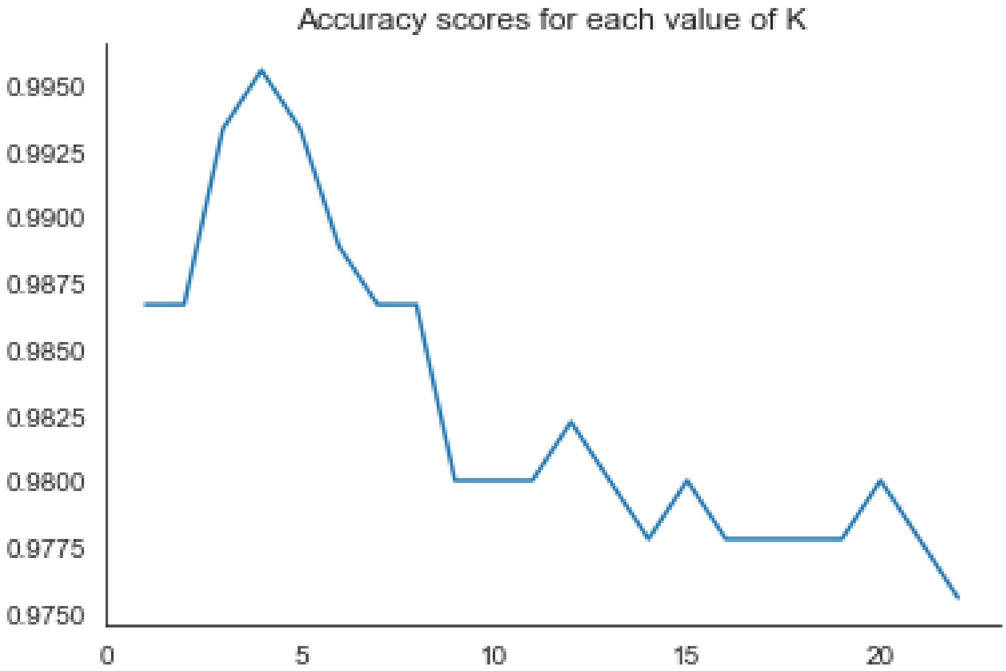
```
In [16]: # Creating the KNN model to use with n equal to 4 (chosen at random by default)
         # General rule for default n is to use odd n if no of variables is even and vice vers
         from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=4)

         # Fitting the knn model with the feature and target data created previously
         knn.fit(X_train, np.ravel(y_train))

         # getting the predicition values for whole dataset using n=4
         y_pred4 = knn.predict(X_test)

         # Check the accuracy score of using n=4
         accuracy_score4 = round(metrics.accuracy_score(y_test, y_pred4), 2)

         print("Test set accuracy: {:.2f}".format(accuracy_score4))
```

```
Test set accuracy: 0.99
```

We can see in these two examples that the KNN model is much superior than the SVM and RandomForest by a significant margin

```
In [17]: #Tuning the KNN model to find best k value

         k_range = range(3, 25)

         accuracy_scores = []

         for k in k_range:
             knn = KNeighborsClassifier(n_neighbors = k)
             knn.fit(X_train, np.ravel(y_train))
             y_pred = knn.predict(X_test)
             accuracy = metrics.accuracy_score(y_test, y_pred)
             accuracy_scores.append(accuracy)
```

In [21]:
```
import seaborn as sns
sns.set_style("white")
#sns.set(rc={'figure.figsize':(20, 10)})
sns.lineplot(x=range(1, 23), y=accuracy_scores).set(title="Accuracy scores for each va

sns.despine(top=True, right=True);
```



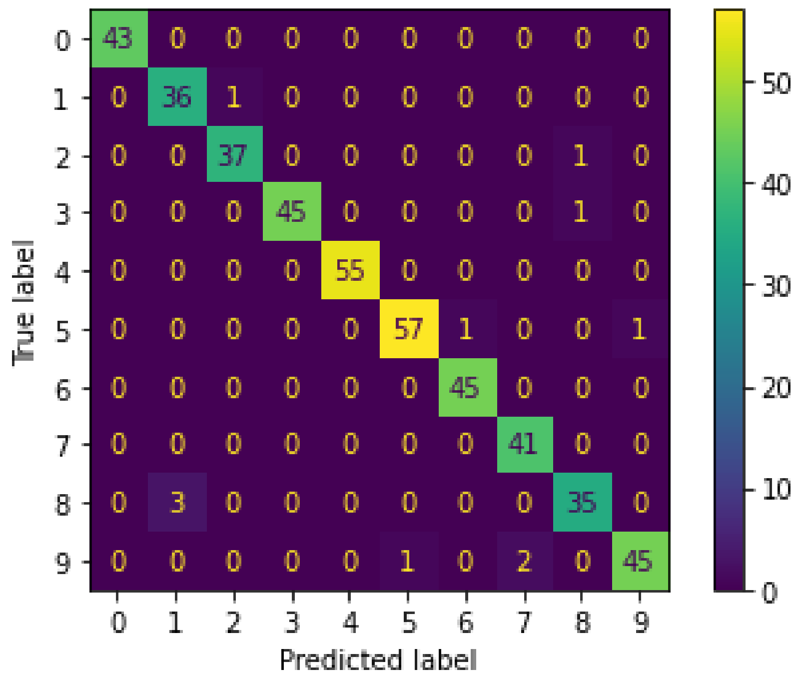
In [18]:
```
print("The best value of K seems to be", accuracy_scores.index(max(accuracy_scores))+
```

```
The best value of K seems to be 4
```

Given we used the value of 4 in our KNN model earlier this doesnt need to change.

In [19]:
```
# Evaluating the model
## Running a confustion matrix on the knn model.

metrics.plot_confusion_matrix(knn, X_test, y_test);

CM = metrics.confusion_matrix(y_test, y_pred)
```

```
C:\Users\Dylan\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\depreca
  warnings.warn(msg, category=FutureWarning)
```



We can see from the above confusion matrix that the digits were more or less correctly predicted with very few outliers. This is a model we can rely upon especially given its 99% accuracy rate.

```
In [25]: # Verification of the model using a classification report

         print(
             f"Classification report for {knn}:\n"
               f"{metrics.classification_report(y_test, y_pred4)}:\n")
```

```
Classification report for KNeighborsClassifier(n_neighbors=24):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        43
           1       0.95      1.00      0.97        37
           2       1.00      1.00      1.00        38
           3       0.98      1.00      0.99        46
           4       0.98      0.98      0.98        55
           5       0.98      1.00      0.99        59
           6       1.00      1.00      1.00        45
           7       1.00      1.00      1.00        41
           8       0.97      0.97      0.97        38
           9       1.00      0.92      0.96        48

    accuracy                           0.99       450
   macro avg       0.99      0.99      0.99       450
```

```
weighted avg       0.99      0.99      0.99       450
:
```

The F1 scores for this model is optimal giving it is very close to 1. In addition to the f1 score we can see that the precision and recall scores and similar particularly for where n classifiers is 4, which is what was used in the model, this indicates that there are not so many false positives and false negatives in the model predictions

4 References

Principal component analysis (2023) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Principal_component_analysis (Accessed: January 6, 2023).

2.5. decomposing signals in components (matrix factorization problems) (no date) scikit. Available at: https://scikit-learn.org/stable/modules/decomposition.html#pca (Accessed: January 6, 2023).

PC