Introduction to Data Science

Assignment 1

J.E. Smit - U1274850

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
May 2019

1 Grey Kangaroos Analysis

1.1 Explore the Data

Load the data set. Present a scatter plot displaying the relationship between these two variables. Label the x axis as nasal length (mm) and y axis as nasal width (mm). Add a title to the scatter-plot.

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

#Loading in the data set
data = pd.read_excel("slr07.xls")

#Splitting data into separate variables X and Y
X = np.array(data['X'])
Y = np.array(data['Y'])

"""Present a scatter-plot displaying the relationship
between the two variables
"""
plt.scatter(X,Y)
plt.title("Grey Kangaroos Nose Data")
plt.xlabel("Nasal Length")
plt.ylabel("Nasal Width")
```

See figure 1 for the scatter-plot of nasal width and length.

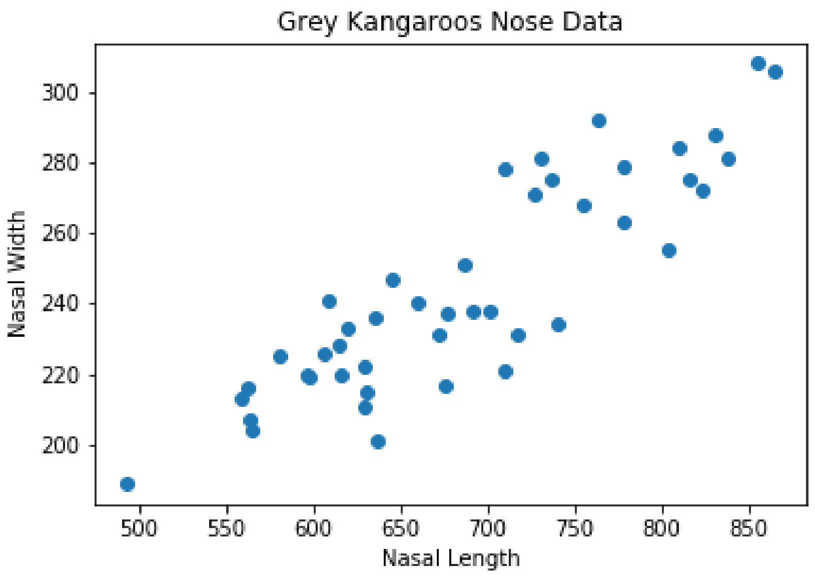


Figure 1: Scatter-Plot of Nasal width and length

1.2 Linear Regression

Fit a linear regression model to the data. Describe how successful the fit is (what is the R2 score?).

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

#Splitting the data into a test and training set
X = X.reshape(-1,1)
#I had to reshape X, because it has a single feature.
X_train, X_test, y_train, y_test = train_test_split(X,Y, random_state=0)

linear = LinearRegression()
linear.fit(X_train, y_train)
print(linear.score(X_test, y_test)) #This is the r-squared

#Visualizing the model fit in the data
pred = linear.predict(X_test)
plt.scatter(X,Y)
plt.title("Grey Kangaroos Nose Data")
plt.xlabel("Nasal Length")
plt.ylabel("Nasal Width")
plt.plot(X_test, pred, color = 'black')
```

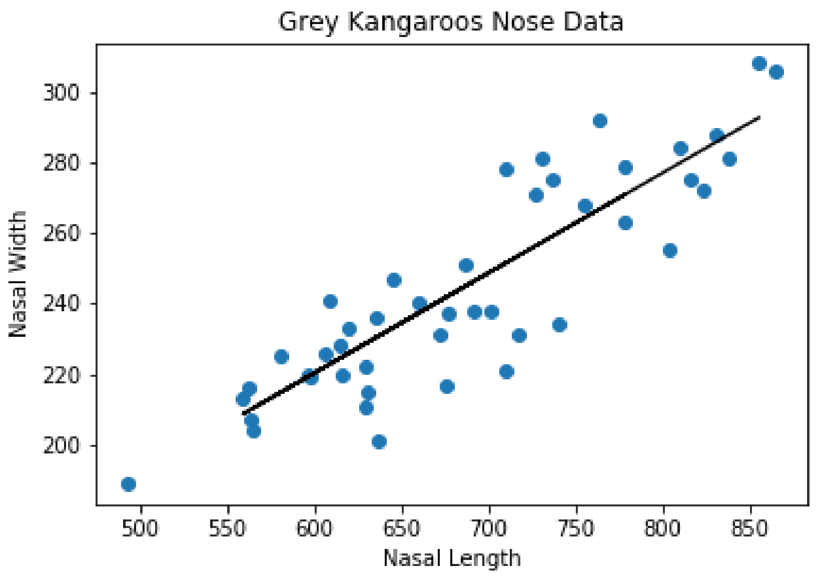


Figure 2: Linear Regression Fit

The fit was evaluated with the R2, which was 0.74. Therefore, it can said that the model fits the data reasonably well. The data is not clustered very tightly. Therefore, a (much) higher R2 score should not be expected. The model fit was visualized in figure 2.

1.3 Two more Regression Models

Besides a simple linear regression model, two other linear models were fitted to the data: a linear support vector machine and neural network regressor. These two were picked over other models based on performance. First, a linear support vector machine regression model was made. The model reached a R2 score of 0.68, which is slightly lower than the linear regression model. The model fit was visualized in figure 3.

```
from sklearn.svm import LinearSVR
lin_svm = LinearSVR()
lin_svm.fit(X_train, y_train)
print(lin_svm.score(X_test, y_test))


#Visualizing the model fit in the data
pred = lin_svm.predict(X_test)
plt.scatter(X,Y)
plt.title("Grey Kangaroos Nose Data")
```

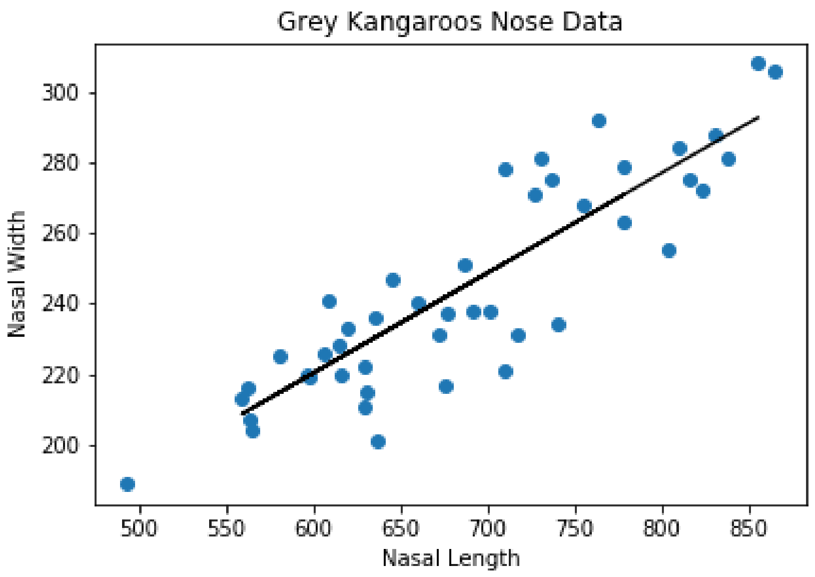


Figure 3: Linear Support Vector Machine Regression

```
plt.xlabel("Nasal Length")
plt.ylabel("Nasal Width")
plt.plot(X_test, pred, color = 'black')
```

In addition, a neural network regressor was fitted to the data. The model reached a R2 score of 0.61. The model fit was visualized in figure 4.

```
#Neural network regression
from sklearn.neural_network import MLPRegressor

nnr = MLPRegressor()
nnr.fit(X_train, y_train)
print(nnr.score(X_test, y_test))

#Visualizing the model fit in the data
pred = nnr.predict(X_test)
plt.scatter(X,Y)
plt.title("Grey Kangaroos Nose Data")
plt.xlabel("Nasal Length")
plt.ylabel("Nasal Width")
plt.plot(X_test, pred, color = 'black')
```

In table 1 the R2 scores of the three linear regression models are compared. Both regression models did not beat the linear regression model.

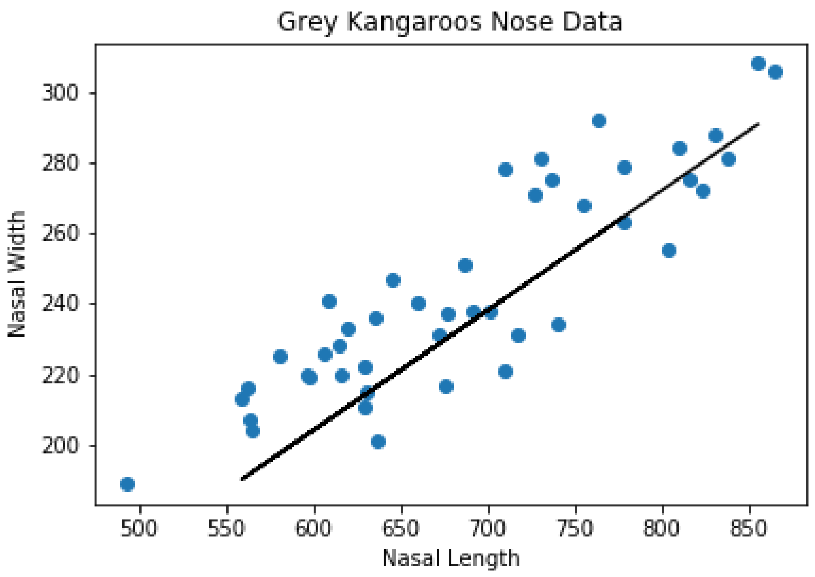


Figure 4: Neural Network Regression

Regression Model	R2 Score
Linear Regression	0,737218557448
Linear SVM Regression	0.682381036217
Neural Network Regression	0.608336781305

Table 1: R2 Scores

2 Facial Expression Prediction

2.1 Data set

Before starting with the first task, the data was explored and prepared for analysis.

```
data = np.load('exp_face.npz')

#Find the keys to split the data into variables
print(data.keys())

#Taking a look at the data
print(data.items())

#Splitting the data into variables based on the keys
X_train = data['X_train']
y_train = data['y_train']
X_valid = data['X_valid']
y_valid = data['y_valid']
X_test = data['X_test']
y_test = data['y_test']
```

2.2 Display an Image from each Category

When looking at the distribution of the categories, it stands out that the data set is imbalanced. There are 963 neutral images, 611 images, and 356 sad pictures. Knowing this, we can expect the model to perform worse on sad pictures than on neutral or happy pictures. The data will be scaled in order to counter this.

Since each instance of X in the training set is an 1-Dimensional array of length 2304, I assumed these numbers are the pixels of the picture and the picture should be square. Thus, each picture should consist of a 48 * 48 matrix of pixels. The example images were reshaped as such a matrix and could then be displayed as an image.

```
print(X_train)

#Exploring the data
print(X_train[0])
print(len(X_train)) #So there are 1930 pictures in the training set
print(len(X_train[0]))

print(y_train[0]) #The first picture is neutral
print(y_train[2]) #Third picture is happy
print(y_train[7]) #Eighth picture is sad

"""Looking at the balance of groups in the dataset
```

```
got this code from stackoverflow:
    https://stackoverflow.com/questions/28663856/how-to-count-the
-occurrence-of-certain-item-in-an-ndarray-in-python
"""
unique, counts = np.unique(y_train, return_counts=True)
print(dict(zip(unique, counts)))

#Saving the pictures as variables
neutral_example = X_train[0]
happy_example = X_train[2]
sad_example = X_train[7]
print(len(neutral_example))
print(len(happy_example))
print(len(sad_example))
#All pictures have the same amount of pixels

# Looking at the shape
print(neutral_example.shape)
#The shape is (2304,), this cannot be displayed as an image

#reshaping the image arrays
neutral_example = neutral_example.reshape(48,48)
happy_example = happy_example.reshape(48,48)
sad_example = sad_example.reshape(48,48)
```

Next, the images could be displayed. The images are displayed in a subplot in figure 5.

```
from matplotlib.pyplot import subplots
import matplotlib.image as mpimg

images = [neutral_example, happy_example, sad_example]
fig, axes = plt.subplots(2, 2)
for ax, img in zip(axes.ravel(), images):
    ax.imshow(img, cmap='gray')

fig.savefig('face_examples')
```

Then the data was scaled to counter imbalance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

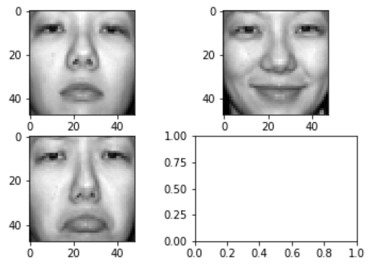


Figure 5: Neutral Image Example

2.2.1 Dimensionality Reduction

Dimensionality reduction was performed using Principal Component Analysis to check whether this could lead to a representation of the data that is more informative for the classifiers. The data-set was reduced to 50, 100 and 200 principal components and then tested on two classifiers in comparison to the original data. In all three cases the reduced data set performed (slightly) worse. Therefore, the original data was used to train the classifiers.

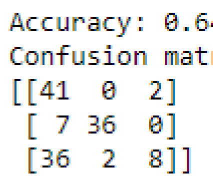
```
from sklearn.decomposition import PCA
import numpy as np

pca = PCA(n_components=100)

pca.fit(X_train)
X_pca_train = pca.transform(X_train)
X_pca_valid = pca.transform(X_valid)
X_pca_test = pca.transform(X_test)
```

2.3 K-Nearest Neighbour Classifier

Create a k-nearest neighbour classifier with uniform weights. This classifier will be baseline classifier.

First, the validation set was used to tune hyper parameters with GridSearchCV. It was chosen to only tune the number of neighbors parameter, since the rest did not seem relevant in this case. The code that was used was taken from the lecture slides. The best number of neighbors was five.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

#Validation
parameter = {'n_neighbors': np.arange(1,15,1)}
```



```
Accuracy: 0.644
Confusion matrix:
[[41  0  2]
 [ 7 36  0]
 [36  2  8]]
```

Figure 6: Confusion Matrix

```
grid_search = GridSearchCV(KNeighborsClassifier(weights='uniform'),
    parameter, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))
```

Next the best number of neighbors that was found (5), was used to train the model and the model was tested on the unseen data from the test-set. The model had a mean accuracy of 0.64. A confusion matrix was produced (see figure 6), to see if some categories were classified better or worse than others. The confusion matrix shows that 95.35 percent of all neutral images were correctly classified, 83.72 percent of all happy images were correctly classified and 17.39 percent of all sad images were correctly classified.

```
#Training the model
knn = KNeighborsClassifier(n_neighbors = 11, weights='uniform')
knn.fit(X_train, y_train)

#Testing model
print("Mean accuracy: {}".format(knn.score(X_test, y_test)))

#Making a confusion matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = knn.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

2.4 Beating the Baseline Classifier

In this section multiple classifiers will be fitted to and tested on the data in an attempt to beat the baseline classifier.

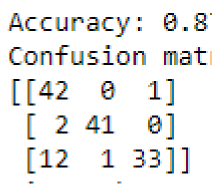
2.4.1 Logistic Regression

A logistic regression model was fitted to the data. Grid search cross-validation was used on the validation set to find the optimal hyper parameters: penalty (l1 vs. l2) regularization and 'C' value. The grid search cross-validation resulted in a 'l1' penalty with C = 10. The model was then fitted with these parameters using the training set. The resulting model had a mean accuracy score of 0.88. In addition, a confusion matrix (see figure 7) was made to determine where the classification errors occurred. Besides the increase in mean accuracy, the logistic regression model had more correctly classified instances of sad images compared to the k-nearest neighbour classifier. The confusion matrix shows that 97.67 percent of all neutral images were correctly classified, 95.35 percent of all happy images were correctly classified and 71.74 percent of all sad images were correctly classified.

```
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression()

#Grid search crossvalidation
param_grid = {'penalty':['l1','l2'],'C':[0.1,1,5,10,100]}
grid_search = GridSearchCV(logistic, param_grid, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))

from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(penalty='l2', C=0.1)
logistic.fit(X_train, y_train)

print("Mean accuracy: {}".format(logistic.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = logistic.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

2.4.2 Neutral Networks

A Multi-layer Perceptron Neural Network model was fitted to the data. Grid search cross-validation was used on the validation set to find the optimal hyper parameters: the amount of hidden layers (1 vs 2), the amount of nodes per hidden layer (10 vs 100 vs 1000) and the alpha value (0.0001 vs 0.01 vs 0.1 vs 1). The grid search cross-validation resulted in two hidden layers with one hundred

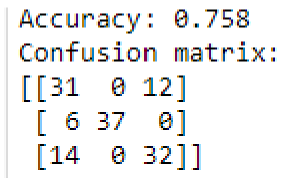


```
Accuracy: 0.879
Confusion matrix:
[[42  0  1]
 [ 2 41  0]
 [12  1 33]]
```

Figure 7: Confusion Matrix

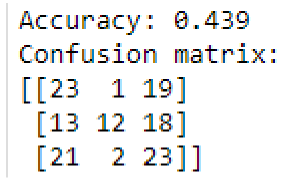
nodes and an alpha value of 1. The model was then fitted with these parameters using the training set. The resulting model had a mean accuracy score of 0.76. In addition, a confusion matrix (see figure 8) was made to determine where the classification errors occurred. The model had a higher mean accuracy score than the baseline classifier. The confusion matrix shows that 72.09 percent of all neutral images were correctly classified, 86.05 percent of all happy images were correctly classified and 69.57 percent of all sad images were correctly classified. The classifier was more success-full classifying sad pictures than the baseline classifier. However, it performed worse on neutral images. The model took quite some time to run.

```
#Neural network
from sklearn.neural_network import MLPClassifier

neural_network = MLPClassifier()
param_grid = {
    'hidden_layer_sizes':[[10],[10,10],[100],[100,100],[1000],
    [1000,1000]],'alpha':[0.0001, 0.01, 0.1, 1]}
grid_search = GridSearchCV(neural_network, param_grid, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))

from sklearn.neural_network import MLPClassifier
neural_network = MLPClassifier(hidden_layer_sizes=1000, alpha=0.01)
neural_network.fit(X_train, y_train)

print("Mean accuracy: {}".format(neural_network.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = neural_network.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```



```
Accuracy: 0.758
Confusion matrix:
[[31  0 12]
 [ 6 37  0]
 [14  0 32]]
```

Figure 8: Confusion Matrix



```
Accuracy: 0.439
Confusion matrix:
[[23  1 19]
 [13 12 18]
 [21  2 23]]
```

Figure 9: Confusion Matrix

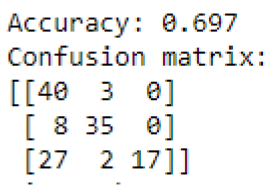
2.4.3 Naive Bayes

A Naive Bayes model (Gaussian) was fitted to the data. The resulting model had a mean accuracy score of 0.44. In addition, a confusion matrix (see figure 9) was made to determine where the classification errors occurred. The model had a lower mean accuracy score than the baseline classifier and also had difficulty correctly classifying sad images. The confusion matrix shows that 97.7 percent of all neutral images were correctly classified, 51.2 percent of all happy images were correctly classified and 21.7 percent of all sad images were correctly classified.

```
#Naive bayes
from sklearn.naive_bayes import GaussianNB
naive = GaussianNB()
naive.fit(X_train, y_train)

print("Mean accuracy test set: {}".format(naive.score(X_valid, y_valid)))
print("Mean accuracy test set: {}".format(naive.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = naive.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

2.4.4 Decision Tree

A Decision Tree Classifier model was fitted to the data. Grid search cross-validation was used on the validation set to find the optimal hyper parameter

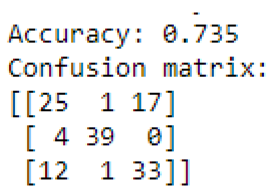


```
Accuracy: 0.697
Confusion matrix:
[[40  3  0]
 [ 8 35  0]
 [27  2 17]]
```

Figure 10: Confusion Matrix

for the maximum depth. The grid search cross-validation resulted in a maximum depth of 30. The model was then fitted with this parameter using the training set. The resulting model had a mean accuracy score of 0.70. In addition, a confusion matrix (see figure 10) was made to determine where the classification errors occurred. The model had slightly higher mean accuracy score than the baseline classifier and also had difficulty correctly classifying sad images. The confusion matrix shows that 93.02 percent of all neutral images were correctly classified, 81.40 percent of all happy images were correctly classified and 36.96 percent of all sad images were correctly classified.

```
#Decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

decision_tree = DecisionTreeClassifier()
param_grid = {'max_depth':[2,4,8,12,16,20,30]}
grid_search = GridSearchCV(decision_tree, param_grid, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))

decision_tree = DecisionTreeClassifier(max_depth=8)
decision_tree.fit(X_train, y_train)

print("Mean accuracy: {}".format(decision_tree.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = decision_tree.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```



```
Accuracy: 0.735
Confusion matrix:
[[25  1 17]
 [ 4 39  0]
 [12  1 33]]
```

Figure 11: Confusion Matrix

2.4.5 Linear Support Vector Machine

A Linear Support Vector Machine Classifier model was fitted to the data. Grid search cross-validation was used on the validation set to find the optimal hyper parameter for the C-value. The grid search cross-validation resulted in a C-value of 0.001. The model was then fitted with this parameter using the training set. The resulting model had a mean accuracy score of 0.73. In addition, a confusion matrix (see figure 11) was made to determine where the classification errors occurred. The model had a higher mean accuracy score than the baseline classifier. The confusion matrix shows that 58.14 percent of all neutral images were correctly classified, 90.70 percent of all happy images were correctly classified and 71.74 percent of all sad images were correctly classified.

```
#Linear svm
from sklearn.svm import LinearSVC

linear_svm = LinearSVC()
param_grid = {'C':[0.00001,0.001,0.01,0.1,1,10,100]}

grid_search = GridSearchCV(linear_svm, param_grid, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))

linear_svm = LinearSVC(C=1e-05)
#very small c-value, so model will underfit the data
linear_svm.fit(X_train, y_train)

print("Mean accuracy: {}".format(linear_svm.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = linear_svm.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

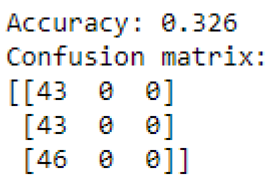
2.4.6 Kernel Support Vector Machine

A Kernel Support Vector Machine Classifier model was fitted to the data. Grid search cross-validation was used on the validation set to find the optimal hyper parameter for the C-value. The grid search cross-validation resulted in a C-value of 10. The model was then fitted with this parameter using the training set. The resulting model had a mean accuracy score of 0.33. In addition, a confusion matrix (see figure 12) was made to determine where the classification errors occurred. The model had a lower mean accuracy score than the baseline classifier.The model classified every instance as a neutral picture. The confusion matrix shows that 100 percent of all neutral images were correctly classified, 0 percent of all happy images were correctly classified and 0 percent of all sad images were correctly classified.

```
#Kernel svm
from sklearn.svm import SVC

kernel_svm = SVC()
param_grid = {'C':[0.00001,0.001,0.01,0.1,1,10,100]}

grid_search = GridSearchCV(kernel_svm, param_grid, cv = 5)
grid_search.fit(X_valid, y_valid)

print("Best score: {}".format(grid_search.best_score_))
print("Best number of neighbors: {}".format(grid_search.best_params_))

kernel_svm = SVC(C=1e-05)
#very small c-value, so model will underfit the data
kernel_svm.fit(X_train, y_train)

print("Mean accuracy: {}".format(kernel_svm.score(X_test, y_test)))

#Making a confusion matrix to see, how the model performs.
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

pred = kernel_svm.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

2.5 Conclusion

Multiple approaches have been tested on the data. Overall, the logistic regression classifier performed best. The model had a higher mean accuracy and was classified most sad images correctly, which was the main flaw of the baseline classifier. The mean accuracy scores and the run-time of each model are displayed in table 2.



```
Accuracy: 0.326
Confusion matrix:
[[43  0  0]
 [43  0  0]
 [46  0  0]]
```

Figure 12: Confusion Matrix

Classifier	Mean Accuracy Score	Run-time
KNN	0.64	28.99
Logistic Regression	0.88	69.48
Neural Network	0.76	892.37
Naive Bayes	0.44	0.20
Decision Tree	0.70	18.99
Linear SVM	0.745	88.19
Kernel SVM	0.33	38.89

Table 2: Mean accuracy scores and run-time

No cross-validation was done. Normally, I would have used stratified k-fold cross-validation to ensure the test was representative to the data set. However, since the data set was purposely split (into a validation, training and test set) to avoid having a person's face in the training, validation and test set, I chose not to apply cross-validation.