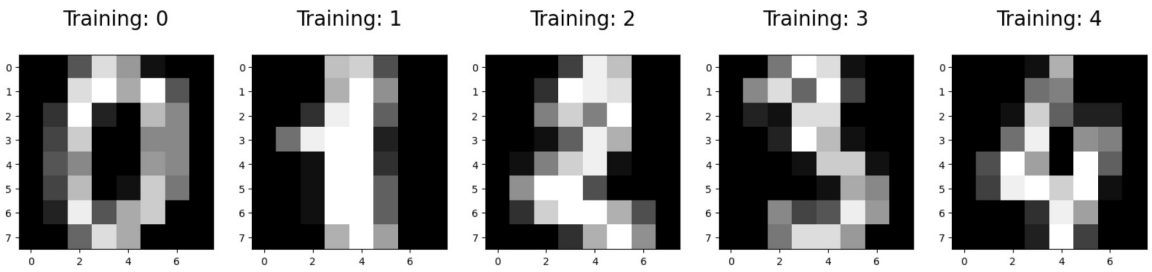
Task 46 MNIST Data

In [2]:
```
import numpy as np

import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits = load_digits()
```

In [3]:
```
# Print to show there are 1797 images (8 by 8 images for a dimensionality of 64)
print("Image Data Shape" , digits.data.shape)

# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", digits.target.shape)
```

```
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

In [4]:
```
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```



In [5]:
```
digits.data
```

Out[5]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [6]:
```
digits.target
```

Out[6]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [7]:
```
# Assign to X axis and y axis
X = digits.data      # Features
y = digits.target    # Labels
```

Split data into training, development and test set

In [9]:
```
from sklearn.model_selection import train_test_split
# Run once to split into train and test sets and then again to split train again

r = 4
X_train, X_test, y_train, y_test = train_test_split(X,
                y, test_size=0.25, random_state=r)                  # 75% traini

X_train, X_dev, y_train, y_dev = train_test_split(X_train,
                y_train, test_size=0.25, random_state=r)            # 75% traini
```

The purpose of the train, development and test set

Train Set

The train set is used to train the program by exposure to a collection of correct examples with answers. This supervised experience provides the model with the training to then go on to provide correct answers to similar questions it's not seen before.

Development Set

The development, or validation set, is used to see if the program is generalising well to data that is not in the training set. In doing this we can get an idea of underfitting or overfitting and fix it before using the test data. Overfitting means the model is too tight matched to the actual examples in the training set and won't be good at predicting results for data out of the training set. Underfitting is the opposite, in that the model is too loosely matched to the training set and will also not be good at predicting results.

Test Set

The test set is a collection of examples used to assess the performance of the program. This is used at the final stage and provides the unseen data to test the model.

Import RandomForestClassifier library and define our classifier

In [64]:
```
# Import Random Forest Model

from sklearn.ensemble import RandomForestClassifier
```

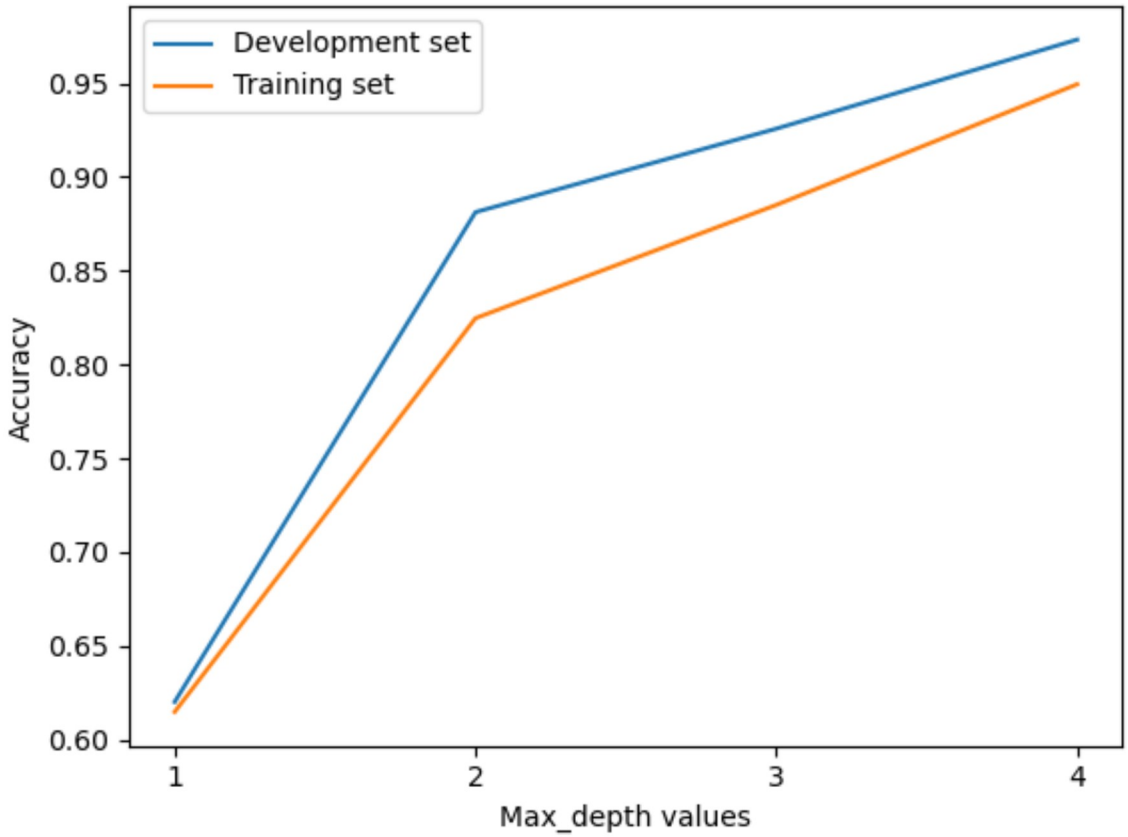
In [67]:
```
# Create Random Forest Model

forest = RandomForestClassifier(n_estimators=100, max_depth=4,random_state=7)

# Fit the model on the training sets
forest.fit(X_train, y_train)

y_pred = forest.predict(X_dev)#.reshape(-1,1)
```

In [70]:
```
print("Accuracy random forest:",forest.score(X_dev, y_dev))
print("At depth:", forest.max_depth)
```

```
Accuracy random forest: 0.9050445103857567
At depth: 4
```

Plot of training and development accuracies

In [73]:
```
# Looking at effect of pruning on train and dev data

forest_depths = range(1,forest.max_depth+1)
forest_scores = []
for d in forest_depths:
  forest = RandomForestClassifier(max_depth=d, random_state=r)
  forest.fit(X_dev, y_dev)
  score = forest.score(X_dev, y_dev)
  forest_scores.append(score)

fig, ax = plt.subplots()
ax.plot(forest_depths, forest_scores)
plt.xlabel("Max_depth values")
plt.ylabel("Accuracy")
ax.xaxis.set_ticks(forest_depths)
# plt.show()

# Add train data to plot

forest_depths = range(1,forest.max_depth+1)
forest_scores = []
for d in forest_depths:
  forest = RandomForestClassifier(max_depth=d, random_state=r)
  forest.fit(X_train, y_train)
  score = forest.score(X_train, y_train)
  forest_scores.append(score)

ax.plot(forest_depths, forest_scores)
plt.legend(['Development set','Training set'])
plt.show()
```



Chosing a parameter to tune

The most important parameter to tune is the n_estimators. It determines the number of trees. The more trees the more robust the model and as the trees go up the variance decreases. A less robust model will mean that small perturbations in training data will give substantially different predictions on the test data. Having a more robust model lessens this effect. However there is a performance issure in using a large number of trees as using more trees increases the computational time and power required.

Using GridSearchCV to try a range of n_estimators values

In [88]:
```
# Declare the parameter values we wish to investigate

parameters = {
    'n_estimators': [50, 100, 200, 300, 400],
    'max_depth' : [3]
}
```

In [89]:
```
# Use the GridSearchCV, specifying the parameters as stated above

from sklearn.model_selection import GridSearchCV

# The estimator we'll use is the forest Random Forest declared earlier
# The cv is cross validations to perform

GS_forest = GridSearchCV(estimator=forest, param_grid=parameters, cv= 5)
GS_forest.fit(X_train, y_train)
```

Out[89]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(max_depth=4, random_state=4),
             param_grid={'max_depth': [3],
                         'n_estimators': [50, 100, 200, 300, 400]})
```

In [90]:
```
# Find the best parameters using the best_params method

GS_forest.best_params_
```

Out[90]:
```
{'max_depth': 3, 'n_estimators': 400}
```

Using a for loop to find the best n_estimator

In [97]:
```
scores =[]
for i in range(1, 200):
    forest_test = RandomForestClassifier(n_estimators=i)
    forest_test.fit(X_train, y_train)
    y_pred = forest_test.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
```

In [98]:
```
max_set = max(scores)
max_index = scores.index(max_set)

print(f"Accuracy Score: {max_set} with n_estimators: {max_index}")
```

```
Accuracy Score: 0.9822222222222222 with n_estimators: 80
```

Using a for loop to find the best max_depth

In [107…
```
scores2 =[]
for j in range(1, 30):
    forest_test2 = RandomForestClassifier(max_depth=j)
    forest_test2.fit(X_train, y_train)
    y_pred2 = forest_test2.predict(X_test)
    scores2.append(accuracy_score(y_test, y_pred2))
```

In [108…
```
max_set2 = max(scores2)
max_index2 = scores2.index(max_set2)

print(f"Accuracy Score: {max_set2} with max_depth: {max_index2}")
```

```
Accuracy Score: 0.9844444444444445 with max_depth: 15
```

The best accuracy scores seems to be at max_depth 15 and n_estimators 80. I will use these parameters as they've given the highest accuracy scores.

Using the best n_estimator value: 80 and max_depth 15

In [109…
```
# Run the RandomForestClassifer again with the best parameters

forest2=RandomForestClassifier(random_state=0, n_estimators= 80, max_depth=15)
```

In [110…
```
# Refit the model on the training sets

forest2.fit(X_train, y_train)
```

Out[110]:
```
RandomForestClassifier(max_depth=15, n_estimators=80, random_state=0)
```

In [111…
```
# Calculate the predictions

y_pred=forest2.predict(X_test)
```

In [112…
```
# Calculate accuracy score
from sklearn.metrics import accuracy_score

print("Accuracy: ",accuracy_score(y_test, y_pred))
```

```
Accuracy:  0.9777777777777777
```

Confusion matrix for your Random Forest model on the test set

Confusion Matrix comparing the predictions to the gold labels

In [153…
```
import pandas as pd
from sklearn.metrics import confusion_matrix

classes = ['0','1', '2','3', '4','5', '6','7', '8','9']

conf_mat = confusion_matrix(y_test, y_pred)
cm_data = pd.DataFrame(conf_mat, columns=classes, index=classes)

cm_data
```

Out[153]:

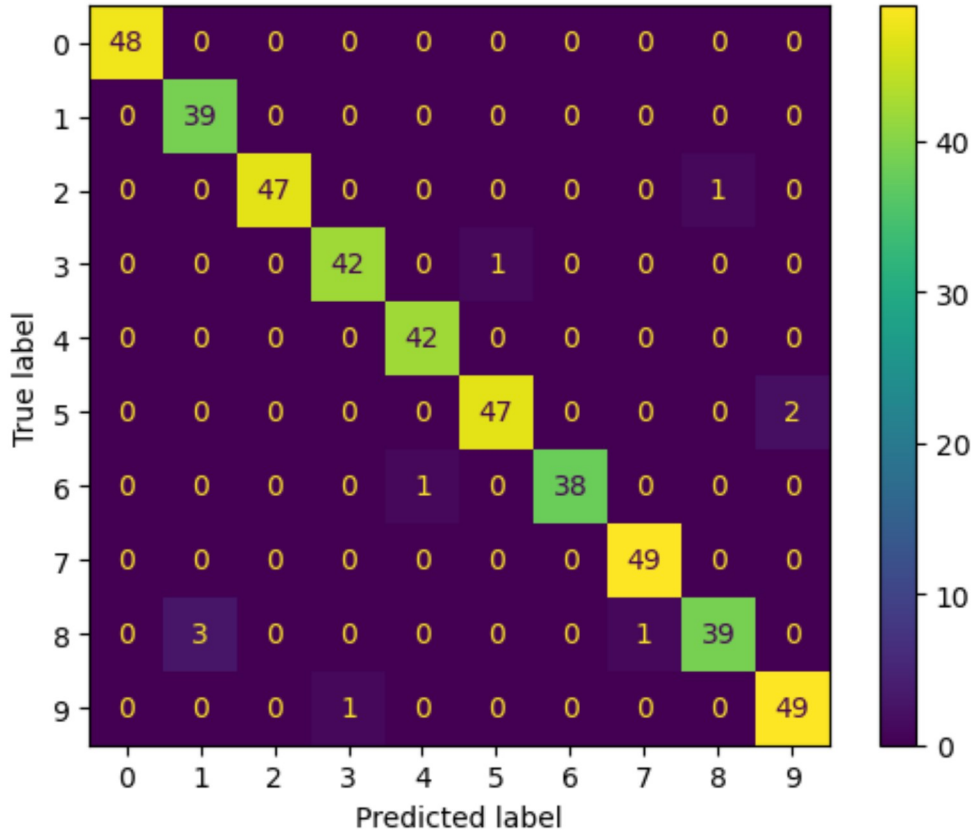
	0	1	2	3	4	5	6	7	8	9
0	48	0	0	0	0	0	0	0	0	0
1	0	39	0	0	0	0	0	0	0	0
2	0	0	47	0	0	0	0	0	1	0
3	0	0	0	42	0	1	0	0	0	0
4	0	0	0	0	42	0	0	0	0	0
5	0	0	0	0	0	47	0	0	0	2
6	0	0	0	0	1	0	38	0	0	0
7	0	0	0	0	0	0	0	49	0	0
8	0	3	0	0	0	0	0	1	39	0
9	0	0	0	1	0	0	0	0	0	49

In [154…

```
# Confusion Matrix Method 2
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
conf_mat2=ConfusionMatrixDisplay(confusion_matrix=conf_mat)
conf_mat2.plot()
```

Out[154]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1de97dd2070
>
```



True labels run down in rows Predicted labels run across in columns

Eg: In recognising the digit 8 the model mistakes it as a 1 on 3 occasions and a 7 in 1 observation.

Classes the model struggles with the most

Accuracy, precision, recall, and f1-score

In [136...
```
from sklearn.metrics import f1_score, precision_score, recall_score

# average f1 score
av_f1 = f1_score(y_test, y_pred, average='micro')
print('Average f1 score: ', av_f1)

# f1 score per class
f = f1_score(y_test, y_pred, average=None)
lowest_score = min(f)
hardest_class = classes[list(f).index(lowest_score)]
print('Hardest class:', hardest_class)
```

```
Average f1 score:  0.9777777777777777
Hardest class: 8
```

In [139...
```
print(f)
```

```
[1.         0.96296296 0.98947368 0.97674419 0.98823529 0.96907216
 0.98701299 0.98989899 0.93975904 0.97029703]
```

The top 3 hardest classes to distinguish the numbers are: 8, 1 and then 5

In [115...
```
# Use score method to get accuracy of model
score = forest.score(X_test, y_test)

print('Accuracy: {}'.format(score))
```

```
Accuracy: 0.92
```

Precision is the proportion of predictions of the positive class that is correct.

Recall is a measure of how many instances of a class the model was able to recognise

In [116...
```
# precision and recall for 0
prec = precision_score(y_test == classes.index('0'), y_pred == classes.index('0'
rec = recall_score(y_test == classes.index('0'), y_pred == classes.index('0'))

print('For 0:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 0:
Precision: 1.0
Recall: 1.0
```

In [117...
```
# precision and recall for 1

prec = precision_score(y_test == classes.index('1'), y_pred == classes.index('1'
rec = recall_score(y_test == classes.index('1'), y_pred == classes.index('1'))

print('For 1:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 1:
Precision: 0.9285714285714286
Recall: 1.0
```

In [118…

```
# precision and recall for 2

prec = precision_score(y_test == classes.index('2'), y_pred == classes.index('2'
rec = recall_score(y_test == classes.index('2'), y_pred == classes.index('2'))

print('For 2:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 2:
Precision: 1.0
Recall: 0.9791666666666666
```

In [119…

```
# precision and recall for 3

prec = precision_score(y_test == classes.index('3'), y_pred == classes.index('3'
rec = recall_score(y_test == classes.index('3'), y_pred == classes.index('3'))

print('For 3:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 3:
Precision: 0.9767441860465116
Recall: 0.9767441860465116
```

In [120…

```
# precision and recall for 4

prec = precision_score(y_test == classes.index('4'), y_pred == classes.index('4'
rec = recall_score(y_test == classes.index('4'), y_pred == classes.index('1'))

print('For 4:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 4:
Precision: 0.9767441860465116
Recall: 1.0
```

In [121…

```
# precision and recall for 5

prec = precision_score(y_test == classes.index('5'), y_pred == classes.index('5'
rec = recall_score(y_test == classes.index('5'), y_pred == classes.index('5'))

print('For 1:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 5:
Precision: 0.9791666666666666
Recall: 0.9591836734693877
```

In [122…

```
# precision and recall for 6

prec = precision_score(y_test == classes.index('6'), y_pred == classes.index('6'
rec = recall_score(y_test == classes.index('6'), y_pred == classes.index('6'))

print('For 6:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 6:
Precision: 1.0
Recall: 0.9743589743589743
```

In [123…

```
# precision and recall for 7

prec = precision_score(y_test == classes.index('7'), y_pred == classes.index('7'
rec = recall_score(y_test == classes.index('7'), y_pred == classes.index('7'))

print('For 7:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 7:
Precision: 0.98
Recall: 1.0
```

In [124…

```
# precision and recall for 8

prec = precision_score(y_test == classes.index('8'), y_pred == classes.index('8'
rec = recall_score(y_test == classes.index('8'), y_pred == classes.index('8'))

print('For 8:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 8:
Precision: 0.975
Recall: 0.9069767441860465
```

In [125…

```
# precision and recall for 9

prec = precision_score(y_test == classes.index('9'), y_pred == classes.index('9'
rec = recall_score(y_test == classes.index('9'), y_pred == classes.index('9'))

print('For 9:')
print('Precision:', prec)
print('Recall:', rec)
```

```
For 9:
Precision: 0.9607843137254902
Recall: 0.98
```

In [ ]:

In [ ]: