Project_3-605840292

June 12, 2023

1 Project 3 - Classify your own data

For this project we're going to explore some of the new topics since the last project including Decision Trees and Un-supervised learning. The final part of the project will ask you to perform your own data science project to classify a new dataset.

1.1 Submission Details

Project is due June 14th at 11:59 am (Wednesday Afternoon). To submit the project, please save the notebook as a pdf file and submit the assignment via Gradescope. In addition, make sure that all figures are legible and sufficiently large. For best pdf results, we recommend downloading Latex and print the notebook using Latex.

1.2 Loading Essentials and Helper Functions

[1]:
```
#Here are a set of libraries we imported to complete this assignment.
#Feel free to use these or equivalent libraries for your implementation
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import matplotlib
import os
import time
#Sklearn classes
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV, KFold
from sklearn import metrics
from sklearn.metrics import confusion_matrix,silhouette_score
import sklearn.metrics.cluster as smc
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import StandardScaler, OneHotEncoder ,LabelEncoder,␣
 ↪MinMaxScaler
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.datasets import make_blobs

from matplotlib import pyplot
import itertools

%matplotlib inline

#Sets random seed
import random
random.seed(42)
```

[2]:
```
#Helper functions
def draw_confusion_matrix(y, yhat, classes):
    '''
        Draws a confusion matrix for the given target and predictions
        Adapted from scikit-learn and discussion example.
    '''
    plt.cla()
    plt.clf()
    matrix = confusion_matrix(y, yhat)
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.YlOrBr)
    plt.title("Confusion Matrix")
    plt.colorbar()
    num_classes = len(classes)
    plt.xticks(np.arange(num_classes), classes, rotation=0)
    plt.yticks(np.arange(num_classes), classes)
    plt.tick_params(top=True, bottom=False, labeltop=True, labelbottom=False)
    fmt = 'd'
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.
 ↪shape[1])):
        plt.text(j, i, format(matrix[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.gca().xaxis.set_label_position('top')
    plt.tight_layout()
    plt.show()

def heatmap(data, row_labels, col_labels, figsize = (20,12), cmap = "YlGn",
            cbar_kw={}, cbarlabel="", valfmt="{x:.2f}",
            textcolors=("black", "white"), threshold=None):
    """
    Create a heatmap from a numpy array and two lists of labels.
```

```
    Taken from matplotlib example.

    Parameters
    ----------
    data
        A 2D numpy array of shape (M, N).
    row_labels
        A list or array of length M with the labels for the rows.
    col_labels
        A list or array of length N with the labels for the columns.
    ax
        A `matplotlib.axes.Axes` instance to which the heatmap is plotted.  If
        not provided, use current axes or create a new one.  Optional.
    cmap
        A string that specifies the colormap to use. Look at matplotlib docs␣
↪for information.
        Optional.
    cbar_kw
        A dictionary with arguments to `matplotlib.Figure.colorbar`.  Optional.
    cbarlabel
        The label for the colorbar.  Optional.
    valfmt
        The format of the annotations inside the heatmap.  This should either
        use the string format method, e.g. "$ {x:.2f}", or be a
        `matplotlib.ticker.Formatter`.  Optional.
    textcolors
        A pair of colors.  The first is used for values below a threshold,
        the second for those above.  Optional.
    threshold
        Value in data units according to which the colors from textcolors are
        applied.  If None (the default) uses the middle of the colormap as
    """

    plt.figure(figsize = figsize)
    ax = plt.gca()

    # Plot the heatmap
    im = ax.imshow(data,cmap=cmap)

    # Create colorbar
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

    # Show all ticks and label them with the respective list entries.
    ax.set_xticks(np.arange(data.shape[1]), labels=col_labels)
    ax.set_yticks(np.arange(data.shape[0]), labels=row_labels)
```

```
    # Let the horizontal axes labeling appear on top.
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
             rotation_mode="anchor")

    # Turn spines off and create white grid.
    ax.spines[:].set_visible(False)

    ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
    ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)


    # Normalize the threshold to the images color range.
    if threshold is not None:
        threshold = im.norm(threshold)
    else:
        threshold = im.norm(data.max())/2.

    # Set default alignment to center, but allow it to be
    # overwritten by textkw.
    kw = dict(horizontalalignment="center",
              verticalalignment="center")

    # Get the formatter in case a string is supplied
    if isinstance(valfmt, str):
        valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)

    # Loop over the data and create a `Text` for each "pixel".
    # Change the text's color depending on the data.
    texts = []
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
            text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
            texts.append(text)


def make_meshgrid(x, y, h=0.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
```

```
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = plt.contourf(xx, yy, Z, **params)
    return out

def draw_contour(x,y,clf, class_labels = ["Negative", "Positive"]):
    """
    Draws a contour line for the predictor

    Assumption that x has only two features. This functions only plots the␣
 ↪first two columns of x.

    """

    X0, X1 = x[:, 0], x[:, 1]
    xx0, xx1 = make_meshgrid(X0,X1)

    plt.figure(figsize = (10,6))
    plot_contours(clf, xx0, xx1, cmap="PiYG", alpha=0.8)
    scatter=plt.scatter(X0, X1, c=y, cmap="PiYG", s=30, edgecolors="k")
    plt.legend(handles=scatter.legend_elements()[0], labels=class_labels)
```

```
    plt.xlim(xx0.min(), xx0.max())
    plt.ylim(xx1.min(), xx1.max())
```

2 Example Project using new techniques

Since project 2, we have learned about a few new models for supervised learning(Decision Trees and Neural Networks) and un-supervised learning (Clustering and PCA). In this example portion, we will go over how to implement these techniques using the Sci-kit learn library.

2.1 Load and Process Example Project Data

For our example dataset, we will use the Breast Cancer Wisconsin Dataset to determine whether a mass found in a body is benign or malignant. Since this dataset was used as an example in project 2, you should be fairly familiar with it.

Feature Information:

Column 1: ID number

Column 2: Diagnosis (M = malignant, B = benign)

Ten real-valued features are computed for each cell nucleus:

```
1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)
```

Due to the statistical nature of the test, we are not able to get exact measurements of the previous values. Instead, the dataset contains the mean and standard error of the real-valued features.

Columns 3-12 present the mean of the measured values

Columns 13-22 present the standard error of the measured values

[3]:
```
#Preprocess Data

#Load Data
data = pd.read_csv('datasets/breast_cancer_data.csv')

#Drop id column
data = data.drop(["id"],axis= 1)

#Transform target feature into numerical
```

```
le = LabelEncoder()
data['diagnosis'] = le.fit_transform(data['diagnosis'])

#Split target and data
y = data["diagnosis"]
x = data.drop(["diagnosis"],axis = 1)

#Train test split
train_raw, test_raw, target, target_test = train_test_split(x,y, test_size=0.2,␣
 ↪stratify= y, random_state=0)

#Standardize data
#Since all features are real-valued, we only have one pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler())
])

#Transform raw data
train = pipeline.fit_transform(train_raw)
test = pipeline.transform(test_raw) #Note that there is no fit calls

#Names of Features after Pipeline
feature_names = list(pipeline.get_feature_names_out(list(x.columns)))
```

[4]:
```
target.value_counts()
```

[4]:
```
0    285
1    170
Name: diagnosis, dtype: int64
```

[5]:
```
#Baseline accuracy of using the majority class
ct = target_test.value_counts()
print("Counts of each class in target_test: ")
print(ct)
print("Baseline Accuraccy of using Majority Class: ", np.max(ct)/np.sum(ct))
```

```
Counts of each class in target_test:
0    72
1    42
Name: diagnosis, dtype: int64
Baseline Accuraccy of using Majority Class:  0.631578947368421
```

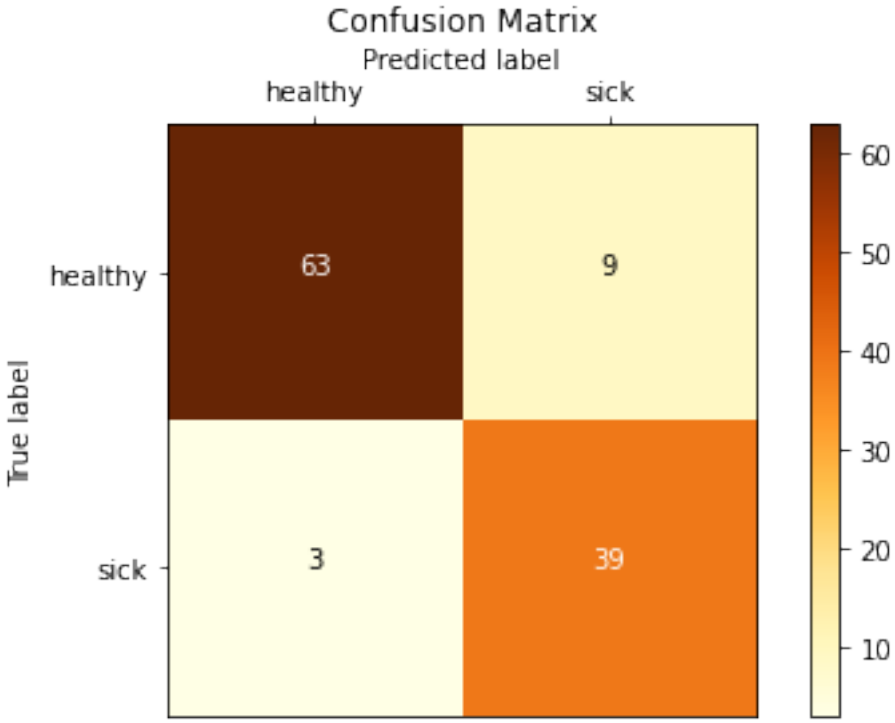
2.2 Supervised Learning: Decision Tree

2.2.1 Classification with Decision Tree

[6]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion="gini", random_state = 0)
clf.fit(train, target)
predicted = clf.predict(test)
```

[7]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,predicted))
draw_confusion_matrix(target_test, predicted, ['healthy', 'sick'])
```

```
Accuracy:    0.894737
Confusion Matrix:
 [[63  9]
 [ 3 39]]
```



2.2.2 Parameters for Decision Tree Classifier

In Sci-kit Learn, the following are just some of the parameters we can pass into the Decision Tree Classifier:

- criterion: {'gini', 'entropy', 'log_loss'} default="gini"

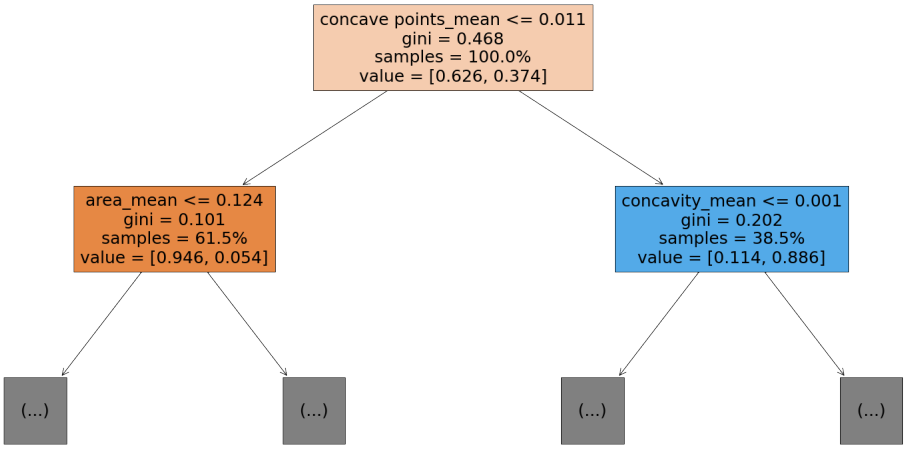
  - The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “log_loss” and “entropy” both for the Shannon information gain
- splitter: {“best”, “random”}, default=”best”
  - The strategy used to choose the split at each node. “best” aims to find the best feature split amongst all features. “random” only looks for the best split amongst a random subset of features.
- max_depth: int, default = 2 {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’
  - The maximum depth of the tree.
- min_samples_split: int or float, default=2
  - The minimum number of samples required to split an internal node. If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

2.2.3 Visualizing Decision Trees

Scikit-learn allows us to visualize the decision tree to see what features it choose to split and what the result is. Note that if the condition in the node is true, you traverse the left edge of the node. Otherwise, you traverse the right edge.

[8]:
```
plt.figure(figsize = (30,15))
#Note that we have to pass the feature names into the plotting function to get␣
 ↪the actual names
#We pass the column names through the pipeline in case any feature augmentation␣
 ↪was made
#For example, a categorical feature will be split into multiple features with␣
 ↪one hot encoding
#and this way assigns a name to each column based on the feature value and the␣
 ↪original feature name
tree.plot_tree(clf,max_depth=1, proportion=True,feature_names=feature_names,␣
 ↪filled=True)
```

[8]:
```
[Text(0.5, 0.8333333333333334, 'concave points_mean <= 0.011\ngini =
0.468\nsamples = 100.0%\nvalue = [0.626, 0.374]'),
 Text(0.25, 0.5, 'area_mean <= 0.124\ngini = 0.101\nsamples = 61.5%\nvalue =
[0.946, 0.054]'),
 Text(0.125, 0.16666666666666666, '\n  (…)  \n'),
 Text(0.375, 0.16666666666666666, '\n  (…)  \n'),
 Text(0.75, 0.5, 'concavity_mean <= 0.001\ngini = 0.202\nsamples = 38.5%\nvalue
= [0.114, 0.886]'),
 Text(0.625, 0.16666666666666666, '\n  (…)  \n'),
 Text(0.875, 0.16666666666666666, '\n  (…)  \n')]
```



We can even look at the tree in a textual format.

[9]:
```
from sklearn.tree import export_text
r = export_text(clf, feature_names=feature_names)
print(r)
```

```
|--- concave points_mean <= 0.01
|   |--- area_mean <= 0.12
|   |   |--- area_se <= 0.04
|   |   |   |--- compactness_mean <= 0.59
|   |   |   |   |--- fractal_dimension_se <= -0.83
|   |   |   |   |   |--- fractal_dimension_se <= -0.84
|   |   |   |   |   |   |--- smoothness_se <= -1.22
|   |   |   |   |   |   |   |--- compactness_se <= -0.98
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- compactness_se >  -0.98
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- smoothness_se >  -1.22
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- fractal_dimension_se >  -0.84
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- fractal_dimension_se >  -0.83
|   |   |   |   |   |--- class: 0
|   |   |   |--- compactness_mean >  0.59
|   |   |   |   |--- symmetry_se <= 0.20
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- symmetry_se >  0.20
|   |   |   |   |   |--- class: 0
|   |   |--- area_se >  0.04
|   |   |   |--- symmetry_mean <= -0.57
```

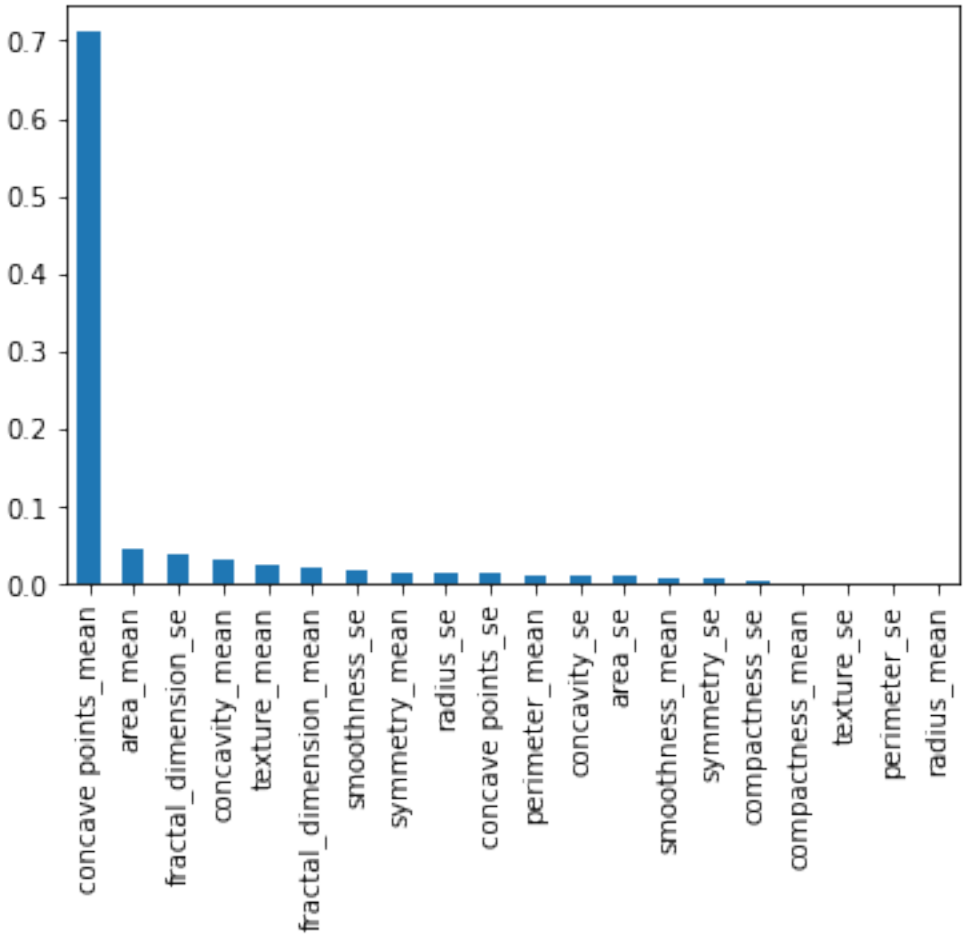
```
|   |   |   |   |--- class: 1
|   |   |   |--- symmetry_mean >  -0.57
|   |   |   |   |--- class: 0
|   |--- area_mean >  0.12
|   |   |--- texture_mean <= -0.72
|   |   |   |--- class: 0
|   |   |--- texture_mean >  -0.72
|   |   |   |--- smoothness_mean <= -1.52
|   |   |   |   |--- class: 0
|   |   |   |--- smoothness_mean >  -1.52
|   |   |   |   |--- class: 1
|--- concave points_mean >  0.01
|   |--- concavity_mean <= 0.00
|   |   |--- fractal_dimension_mean <= -0.83
|   |   |   |--- class: 1
|   |   |--- fractal_dimension_mean >  -0.83
|   |   |   |--- concave points_mean <= 0.11
|   |   |   |   |--- concavity_se <= -0.35
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- concavity_se >  -0.35
|   |   |   |   |   |--- class: 0
|   |   |   |--- concave points_mean >  0.11
|   |   |   |   |--- class: 0
|   |--- concavity_mean >  0.00
|   |   |--- fractal_dimension_se <= 2.39
|   |   |   |--- smoothness_se <= 1.87
|   |   |   |   |--- radius_se <= -0.77
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- radius_se >  -0.77
|   |   |   |   |   |--- concave points_se <= 2.59
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- concave points_se >  2.59
|   |   |   |   |   |   |--- radius_se <= 0.61
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- radius_se >  0.61
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |--- smoothness_se >  1.87
|   |   |   |   |--- concave points_se <= 1.03
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- concave points_se >  1.03
|   |   |   |   |   |--- class: 0
|   |   |--- fractal_dimension_se >  2.39
|   |   |   |--- perimeter_mean <= 0.59
|   |   |   |   |--- class: 0
|   |   |   |--- perimeter_mean >  0.59
|   |   |   |   |--- class: 1
```

2.2.4 Feature Importance in Decision Trees

Decision Trees can also assign importance to features by measuring the average decrease in impurity (i.e. information gain) for each feature. The features with higher decreases are treated as more important.

[10]:
```
imp_pd = pd.Series(data = clf.feature_importances_ ,index = feature_names)
imp_pd= imp_pd.sort_values(ascending=False)
imp_pd.plot.bar()
```

[10]: <AxesSubplot:>

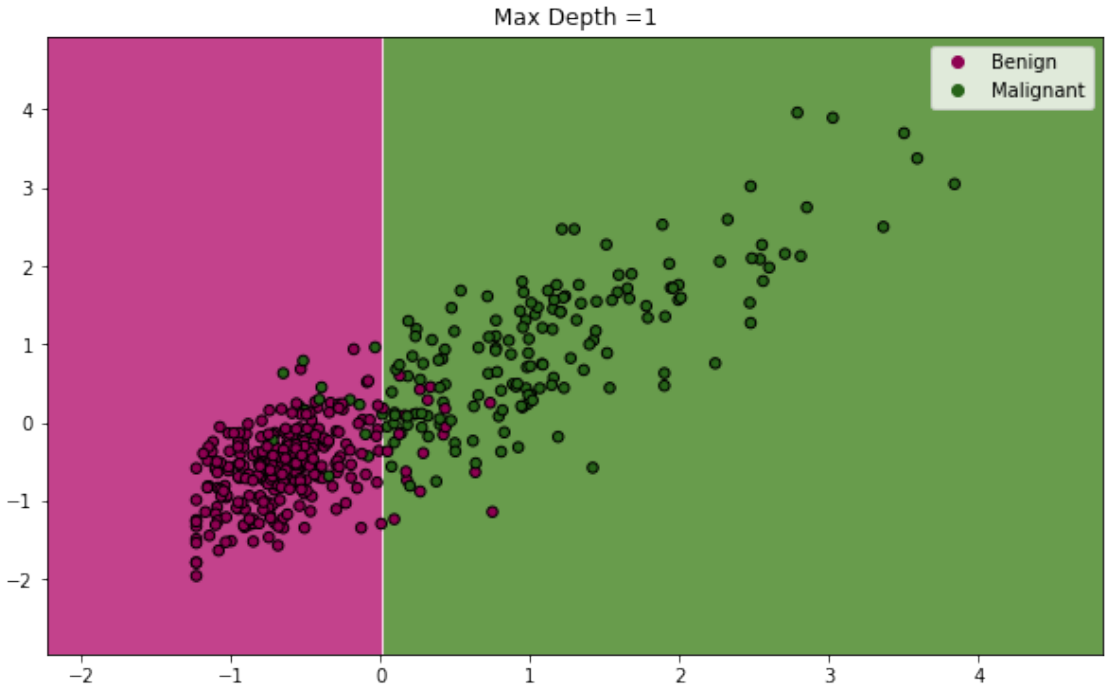


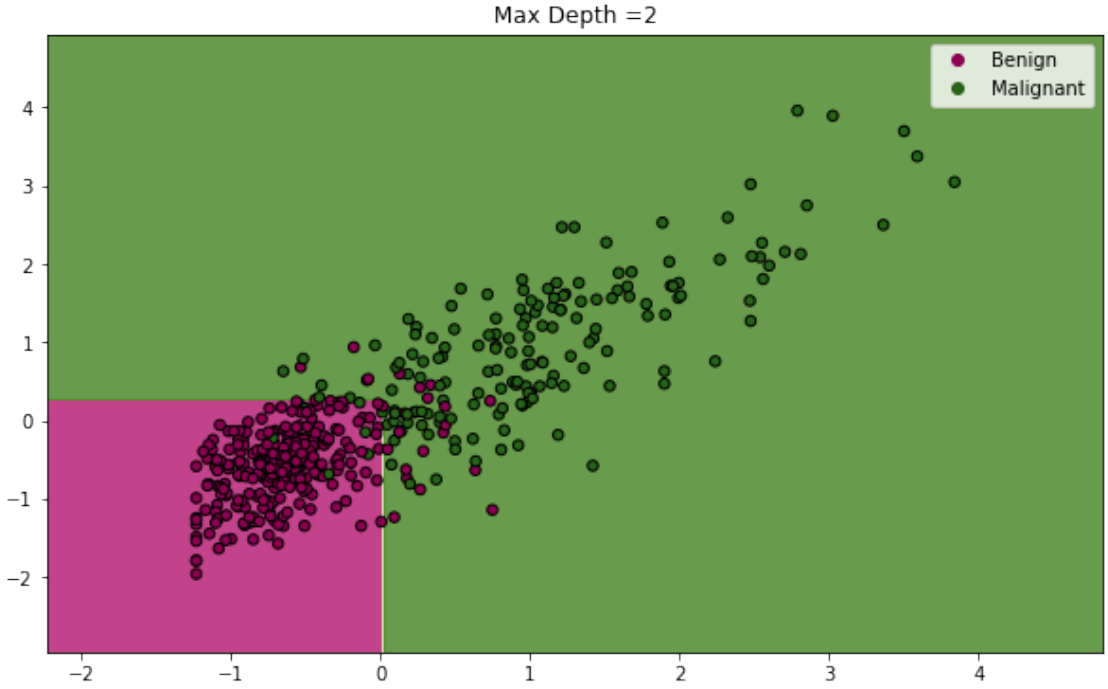
We can clearly see that "concave points_mean" has the largest importance due to it providing the most reduction in the impurity.

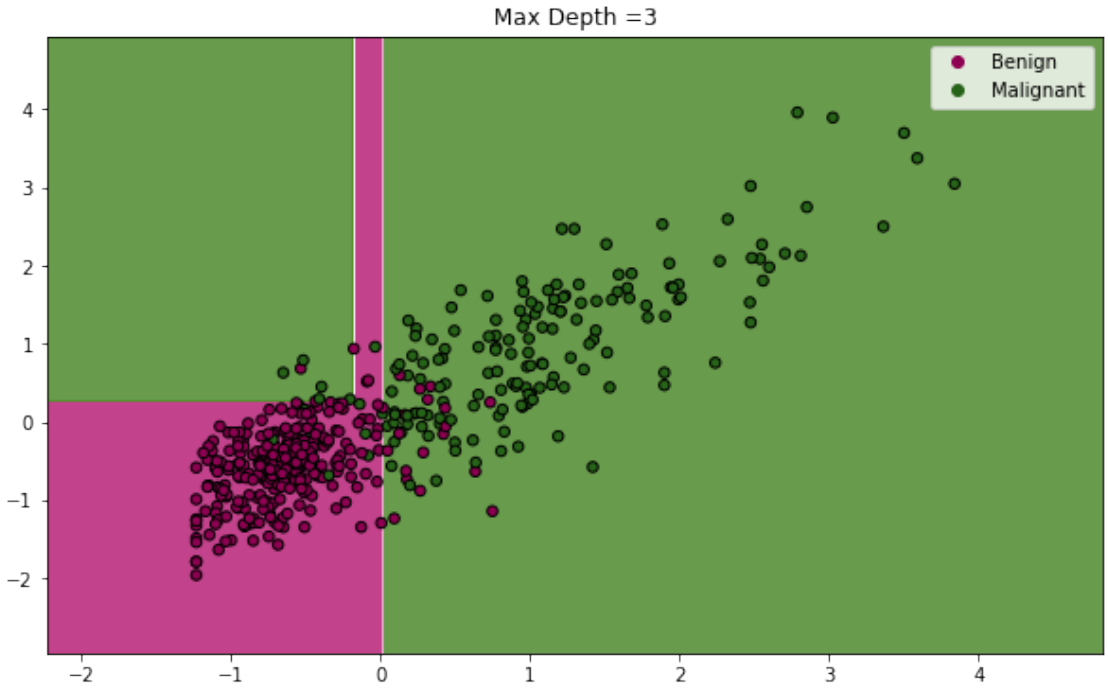
2.2.5 Visualizing decision boundaries for Decision Trees

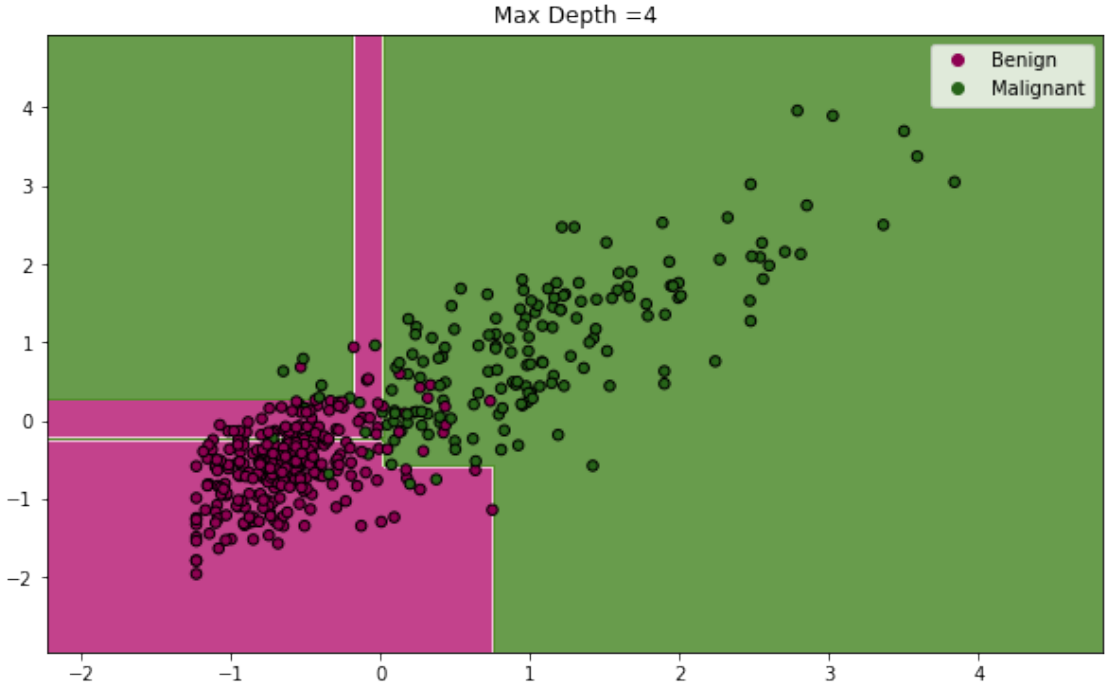
Similar to project 2, lets see what decision boundaries that a Decision Tree creates. We use the two most correlated features to the target labels: concave_points_mean and perimeter_mean.

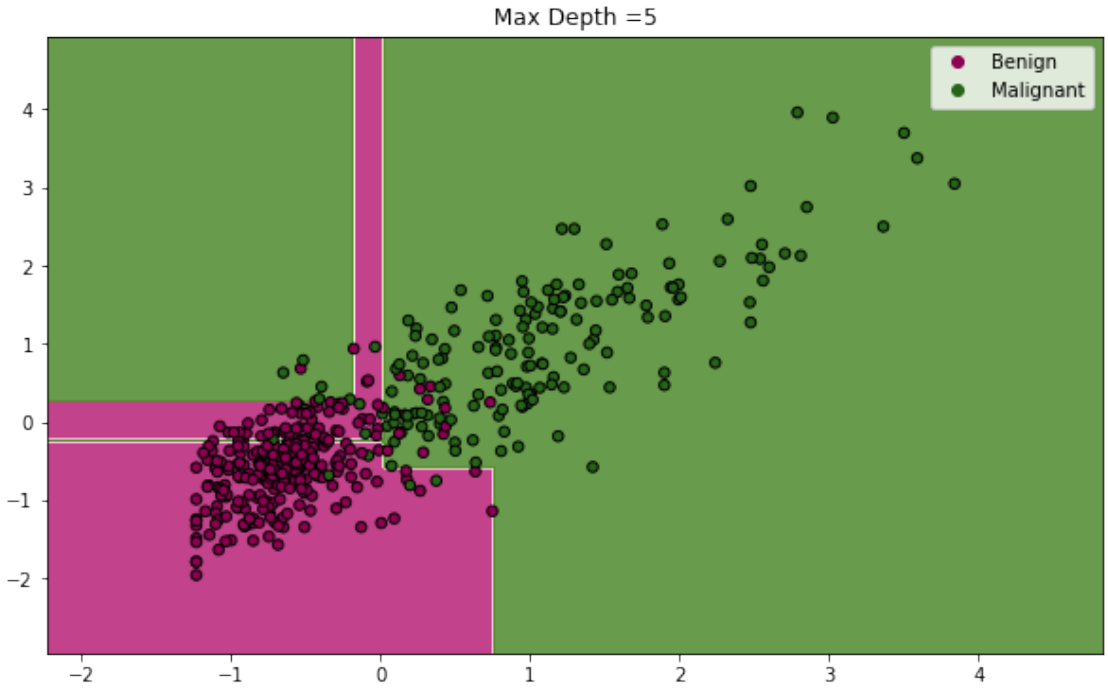
[11]:
```
#Extract first two feature and use the standardscaler
train_2 = StandardScaler().fit_transform(train_raw[['concave␣
 ↪points_mean','perimeter_mean']])

depth  = [1,2,3,4,5,10,15]
for d in depth:
    dt = DecisionTreeClassifier(max_depth = d, min_samples_split=7)
    dt.fit(train_2, target)
    draw_contour(train_2,target,dt,class_labels = ['Benign', 'Malignant'])

    plt.title(f"Max Depth ={d}")
```

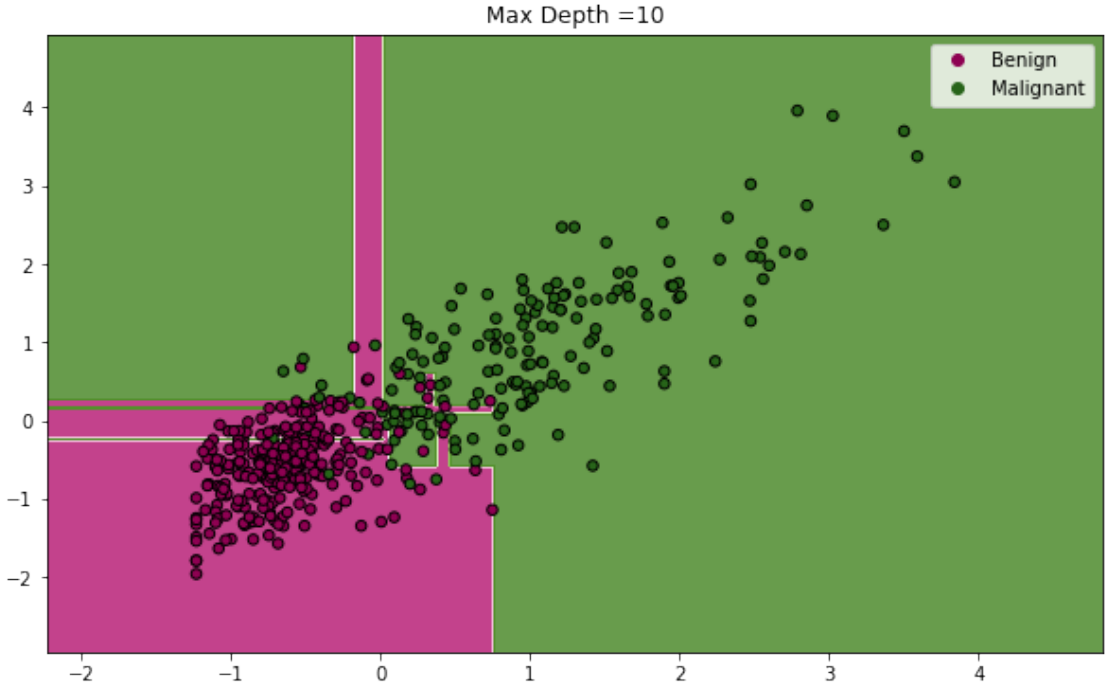


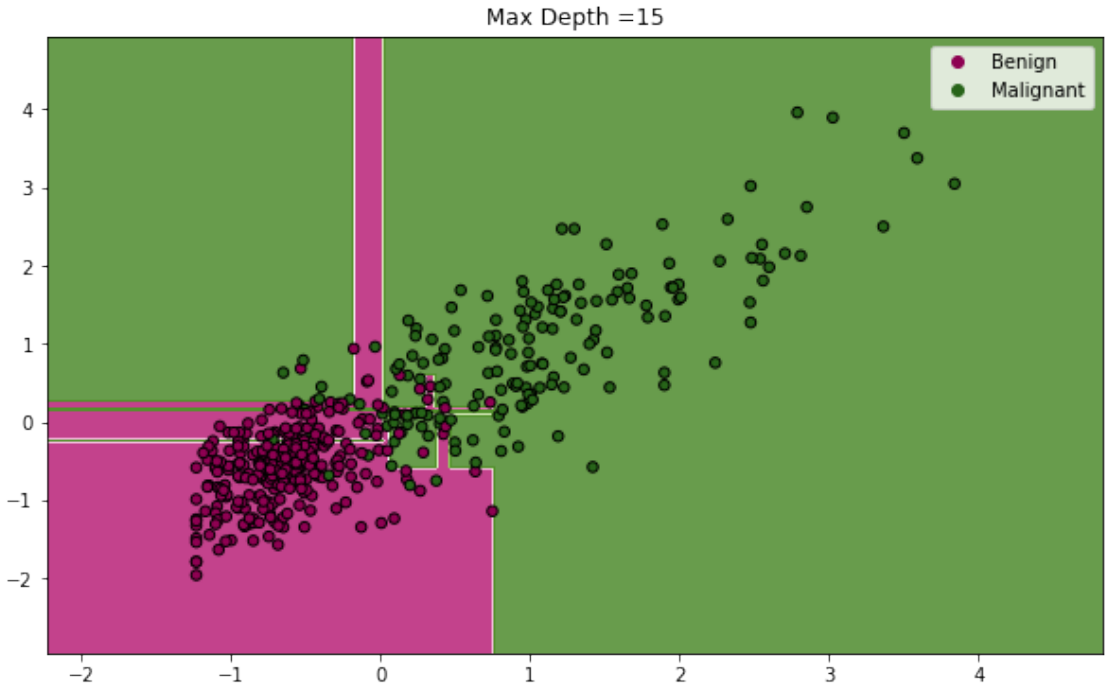

Max Depth =2
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4


Max Depth =3
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4


Max Depth =4
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4


Max Depth =5
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

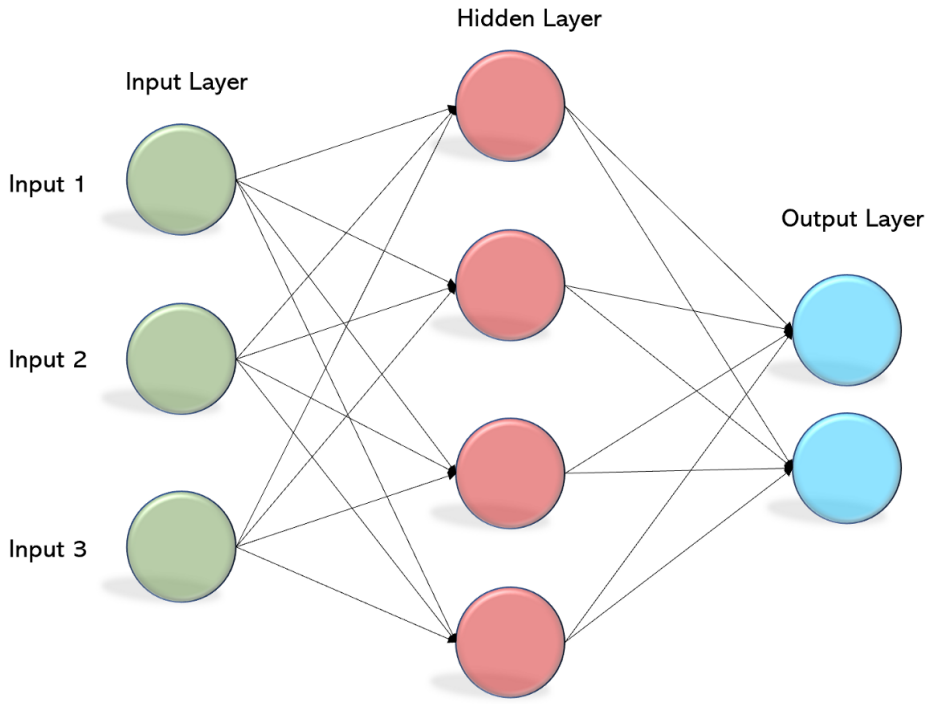




We can see that the model gets more and more complex with increasing depth until it converges somewhere in between depth 10 and 15.

2.3 Supervised Learning: Multi-Layer Perceptron (MLP)

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks are very powerful tools that are used a in a variety of applications including image and speech processing. In class, we have discussed one of the earliest types of neural networks known as a Multi-Layer Perceptron.



2.3.1 Using MLP for classification

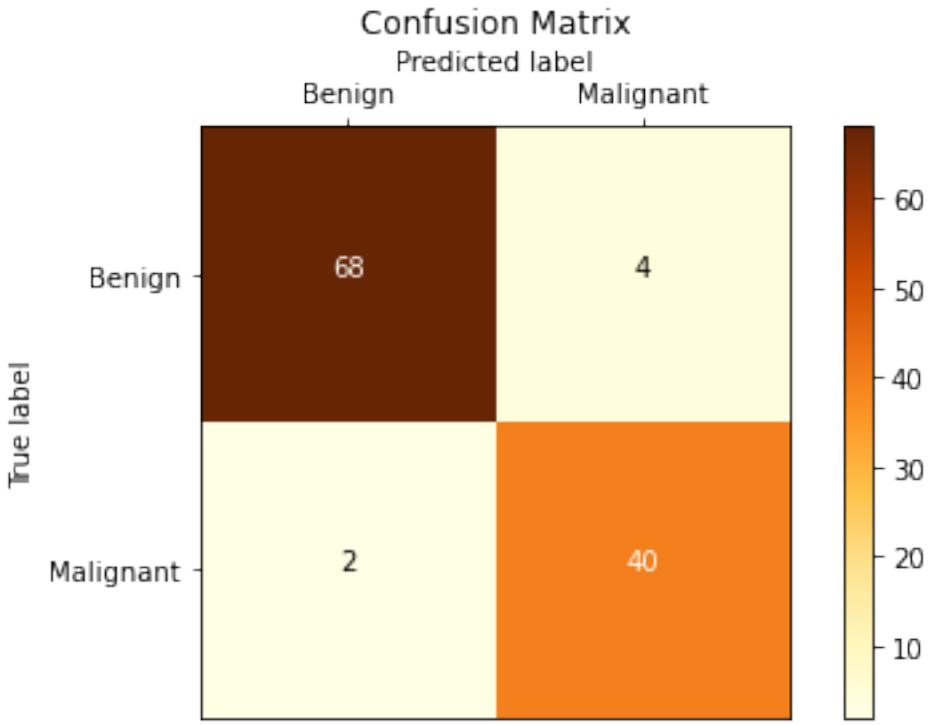
[12]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter = 400)
clf.fit(train, target)
predicted = clf.predict(test)
```

```
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (400) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[13]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,predicted))
draw_confusion_matrix(target_test, predicted, ['Benign', 'Malignant'])
```

```
Accuracy:    0.947368
Confusion Matrix:
```

```
[[68  4]
 [ 2 40]]
```



2.3.2 Parameters for MLP Classifier

In Sci-kit Learn, the following are just some of the parameters we can pass into MLP Classifier:

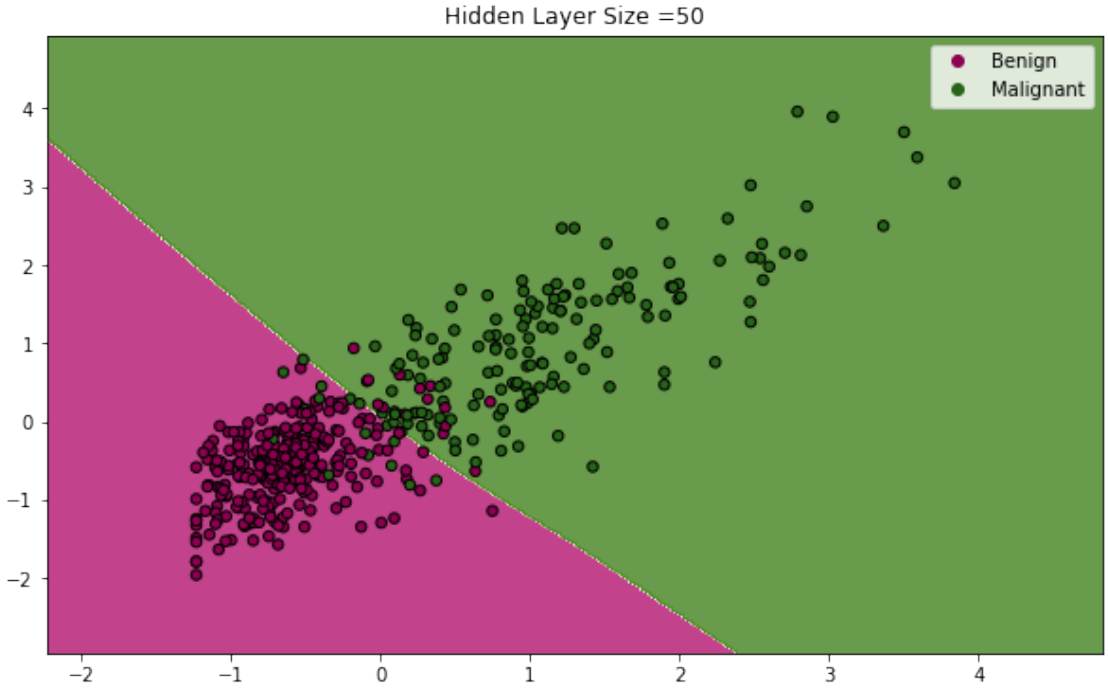
- hidden_layer_sizes: tuple, length = n_layers - 2, default=(100,)
  - The ith element represents the number of neurons in the ith hidden layer.
- activation: {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
  - Activation function for the hidden layer.
- alpha: float, default = 0.0001
  - Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss.
- max_iter: int, default=200
  - Maximum number of iterations taken for the solvers to converge.

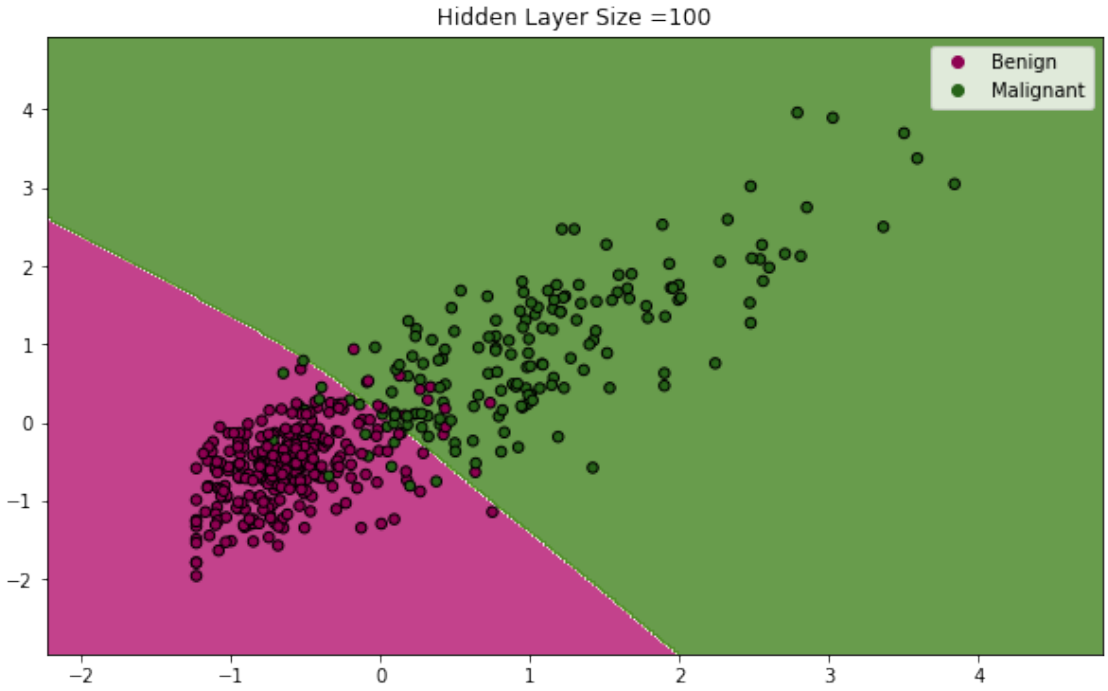
2.3.3 Visualizing decision boundaries for MLP

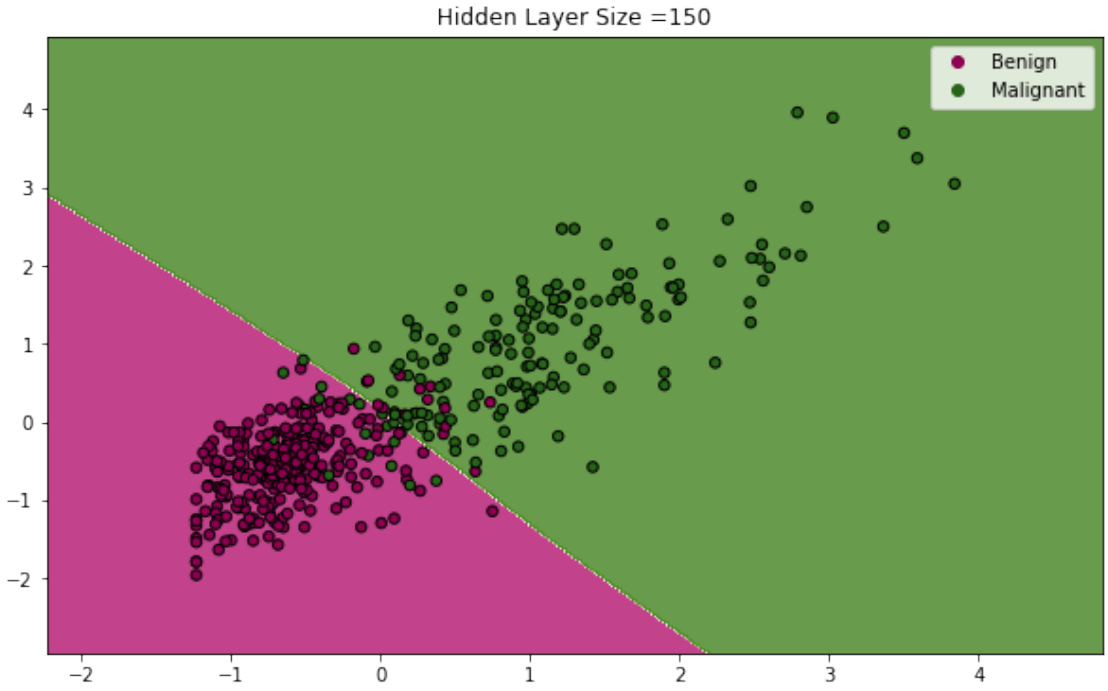
Now, lets see how the decision boundaries change as a function of both the activation function and the number of hidden layers.

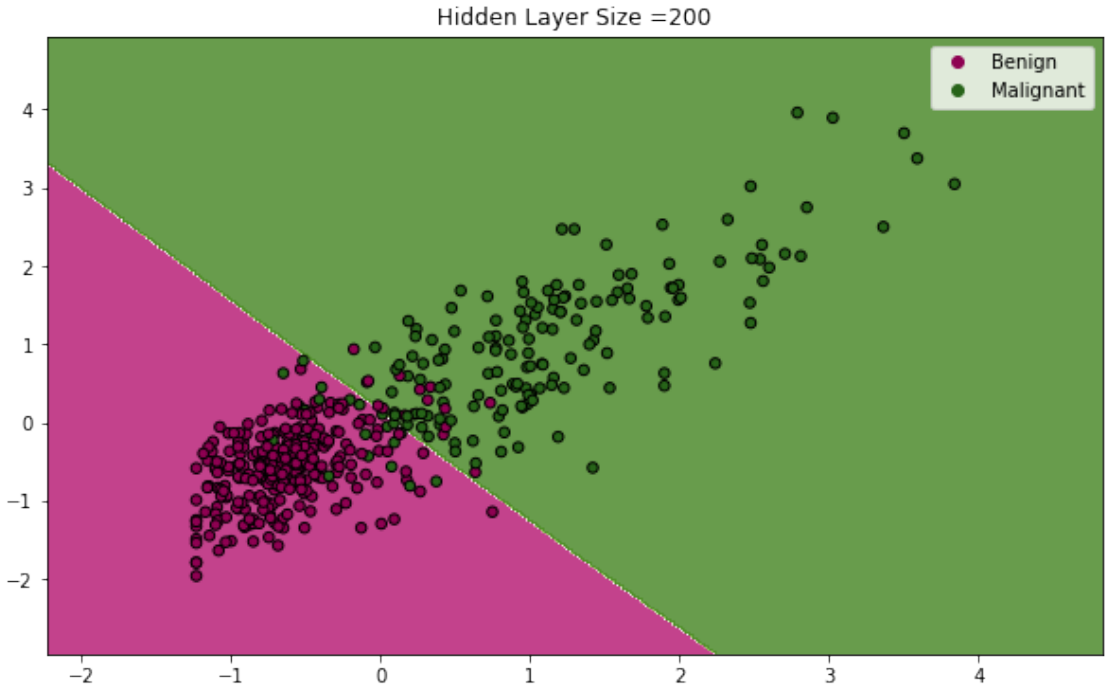
[14]:

```
#Example of using the default Relu activation while altering the number of␣
 ↪hidden layers
train_2 = StandardScaler().fit_transform(train_raw[['concave␣
 ↪points_mean','perimeter_mean']])

layers  = [50,100,150,200]
for l in layers:
    mlp = MLPClassifier(hidden_layer_sizes=(l,), max_iter = 400)
    mlp.fit(train_2, target)
    draw_contour(train_2,target,mlp,class_labels = ['Benign', 'Malignant'])

    plt.title(f"Hidden Layer Size ={l}")
```

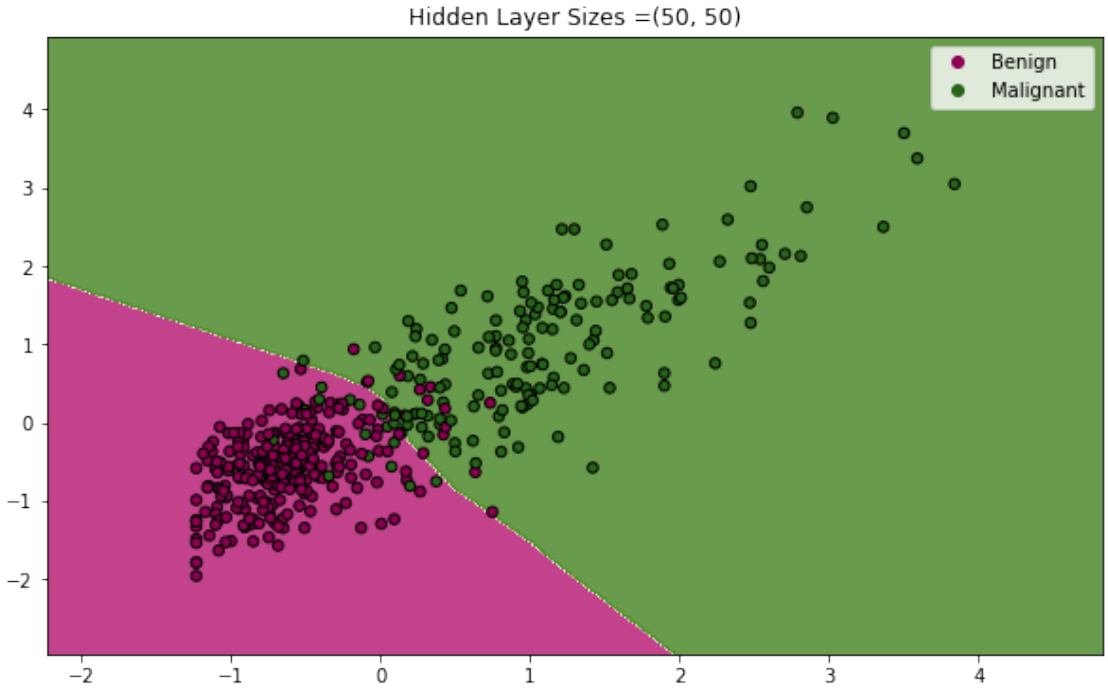


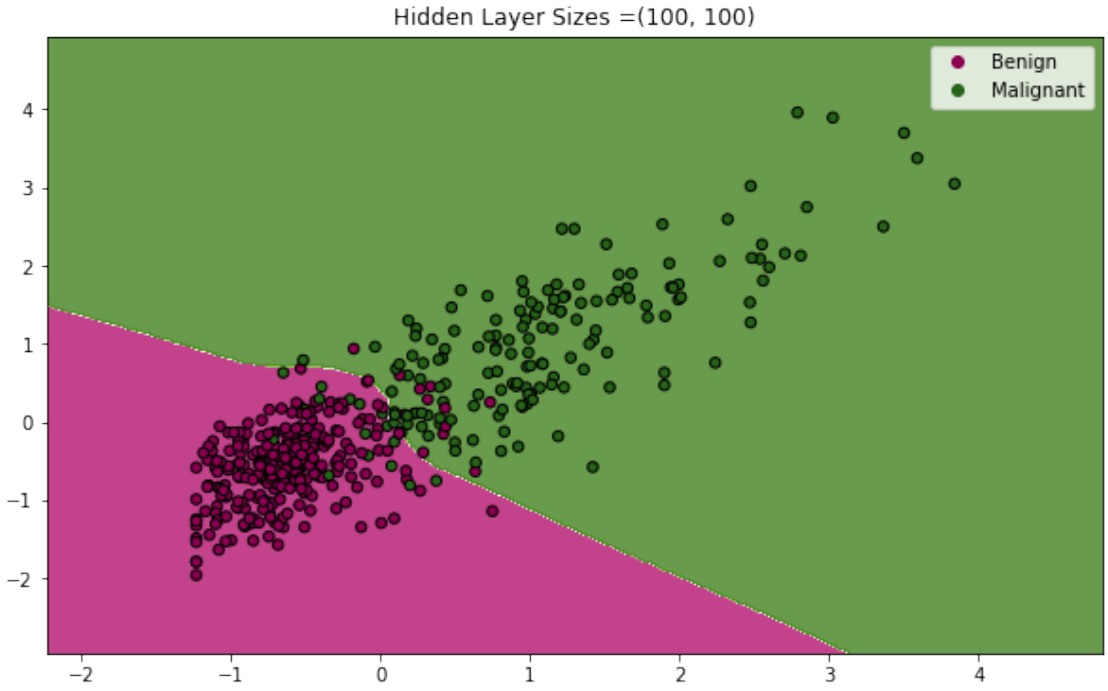

Hidden Layer Size =100
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

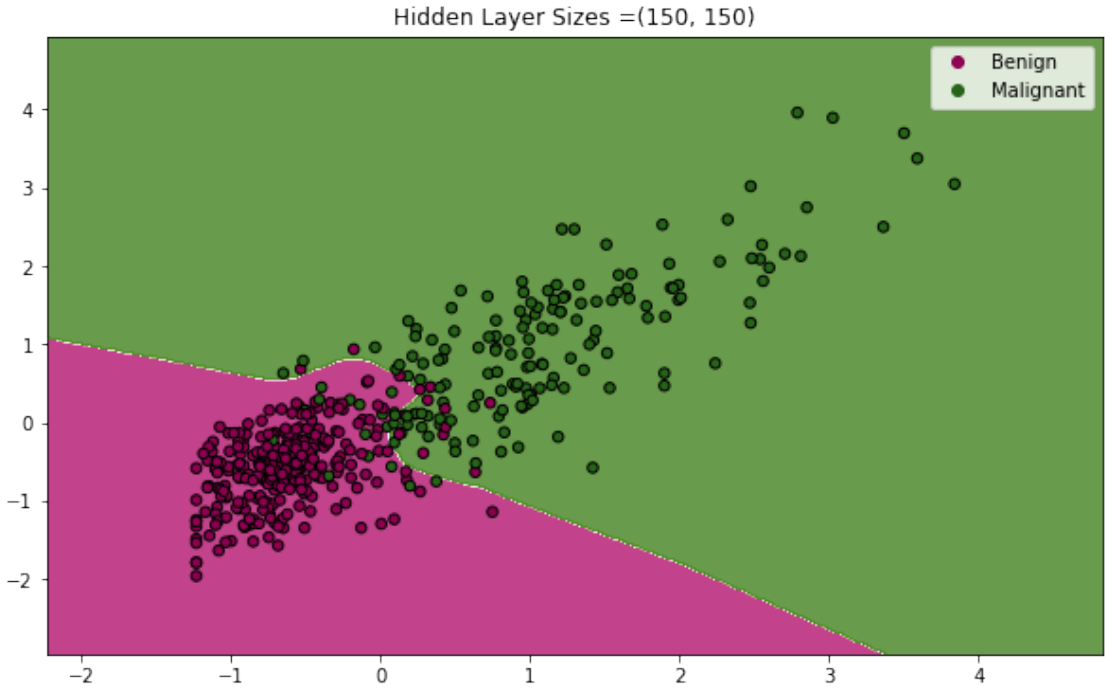

Hidden Layer Size =150
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

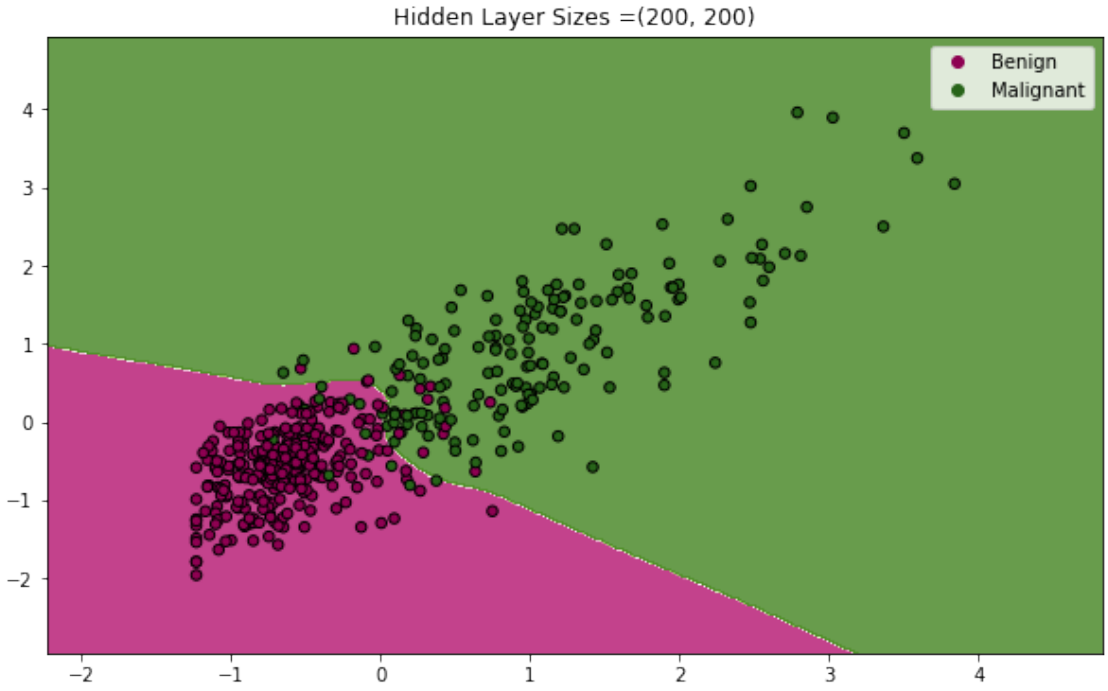


[15]:
```
#Example of using the default Relu activation
#while altering the number of hidden layers with 2 groups of hidden layers
train_2 = StandardScaler().fit_transform(train_raw[['concave␣
 ↪points_mean','perimeter_mean']])

layers  = [50,100,150,200]
for l in layers:
    mlp = MLPClassifier(hidden_layer_sizes=(l,l), max_iter = 400)
    mlp.fit(train_2, target)
    draw_contour(train_2,target,mlp,class_labels = ['Benign', 'Malignant'])

    plt.title(f"Hidden Layer Sizes ={(l,l)}")
```

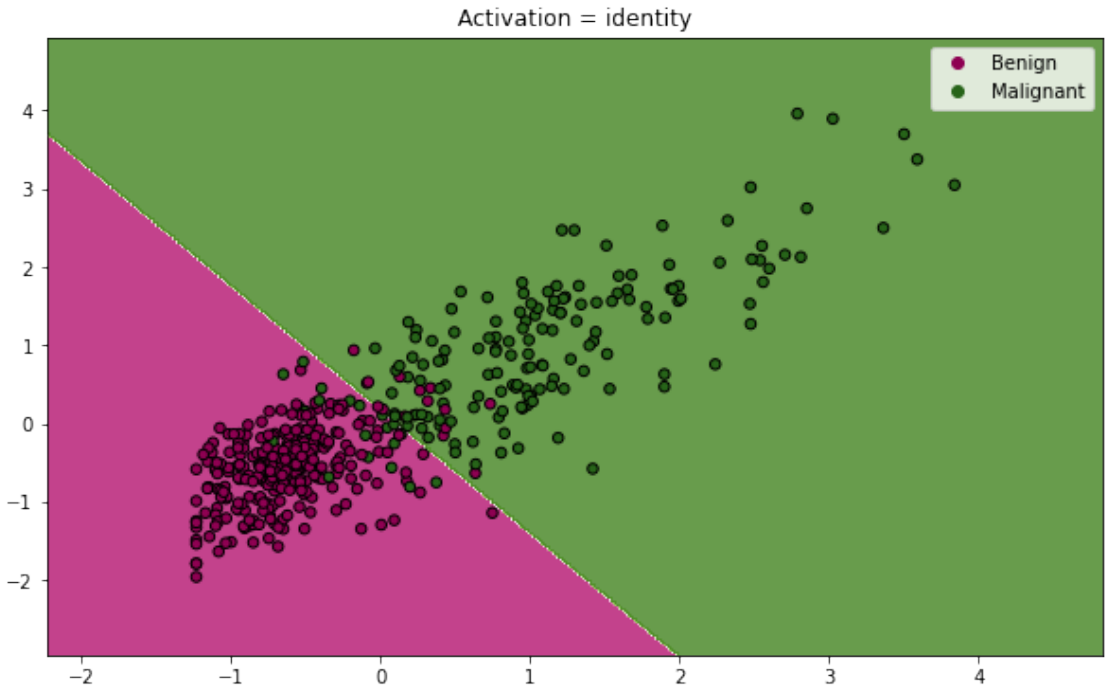

Hidden Layer Sizes =(50, 50)
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

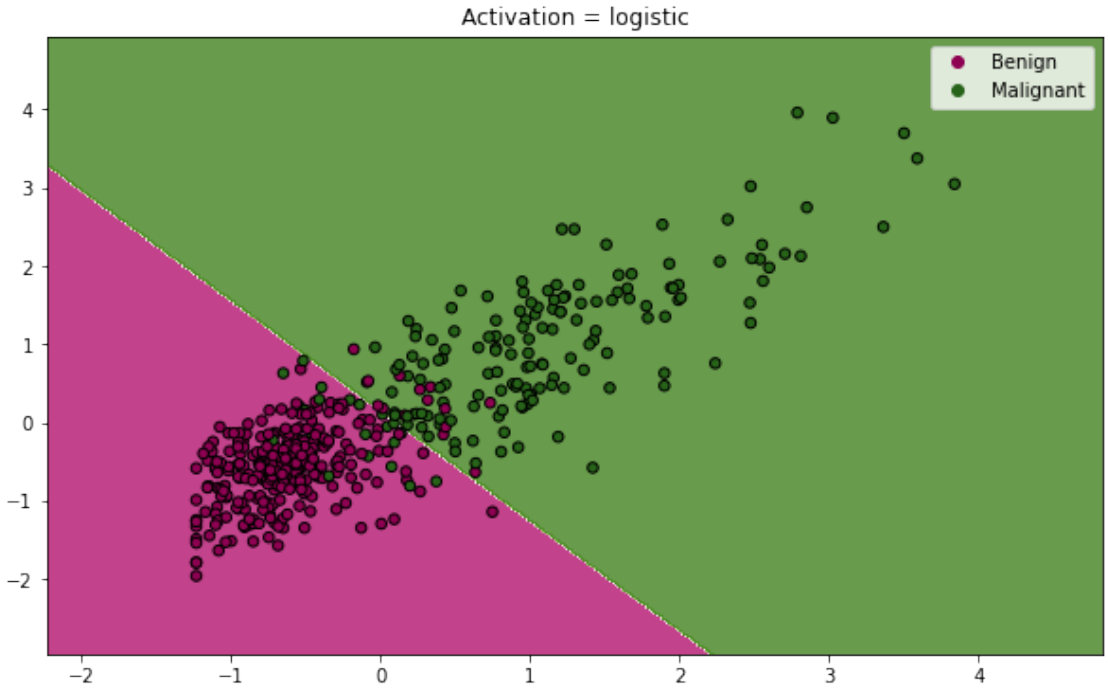

Hidden Layer Sizes =(100, 100)
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

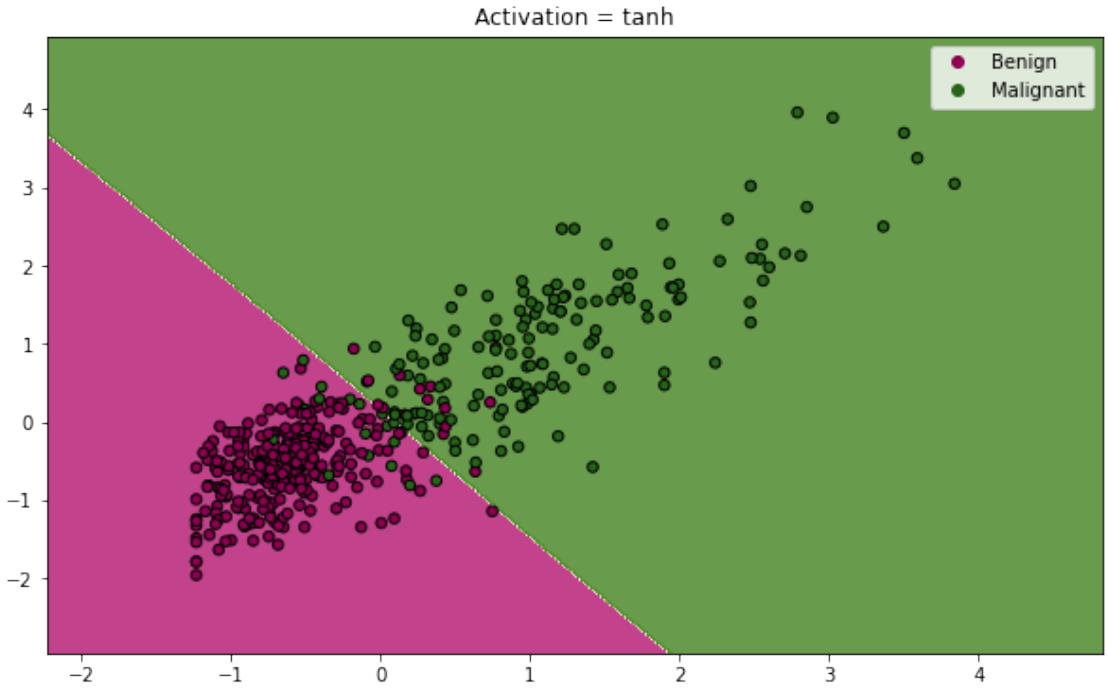

Hidden Layer Sizes =(150, 150)
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

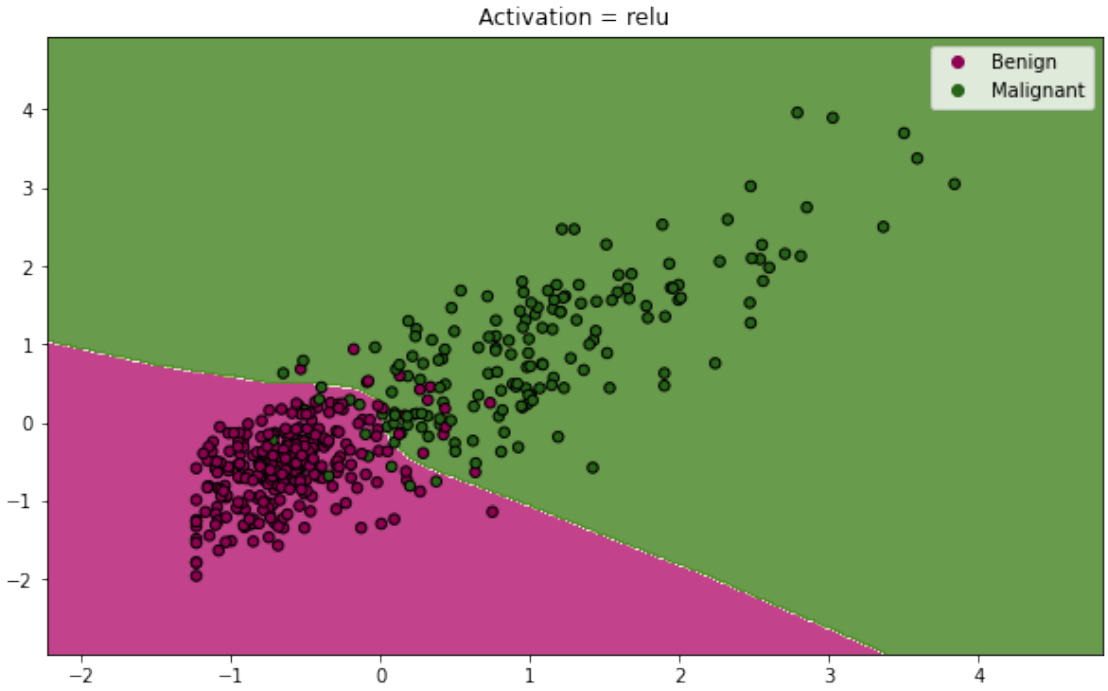

Hidden Layer Sizes =(200, 200)
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

[16]:
```
#Example of using 2 hidden layers of 100 units each with varying activations
train_2 = StandardScaler().fit_transform(train_raw[['concave␣
 ↪points_mean','perimeter_mean']])

acts  = ['identity', 'logistic', 'tanh', 'relu']
for act in acts:
    mlp = MLPClassifier(hidden_layer_sizes=(100,100), activation = act,␣
 ↪max_iter = 400)
    mlp.fit(train_2, target)
    draw_contour(train_2,target,mlp,class_labels = ['Benign', 'Malignant'])

    plt.title(f"Activation = {act}")
```




Activation = logistic
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4


Activation = tanh
Benign
Malignant
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4



2.4 Unsupervised learning: PCA

As shown in lecture, PCA is a valuable dimensionality reduction tool that can extract a small subset of valuable features. In this section, we shall demonstrate how PCA can extract important visual features from pictures of subjects faces. We shall use the AT&T Database of Faces. This dataset contains 40 different subjects with 10 samples per subject which means we a dataset of 400 samples.

We extract the images from the scikit-learn dataset library. The library imports the images (faces.data), the flatten array of images (faces.images), and which subject eacj image belongs to (faces.target). Each image is a 64 by 64 image with pixels converted to floating point values in [0,1].

2.4.1 Eigenfaces

The following codes downloads and loads the face data.

[17]:
```
#Import faces from scikit library
faces = datasets.fetch_olivetti_faces()
print("Flattened Face Data shape:", faces.data.shape)
print("Face Image Data Shape:", faces.images.shape)
print("Shape of target data:", faces.target.shape)
```

```
Flattened Face Data shape: (400, 4096)
Face Image Data Shape: (400, 64, 64)
Shape of target data: (400,)
```

[18]:
```
#Extract image shape for future use
im_shape = faces.images[0].shape
```

[19]:
```
#Prints some example faces
faceimages = faces.images[np.random.choice(len(faces.images),size= 16, replace
 ↪= False)] # take random 16 images

fig, axes = plt.subplots(4,4,sharex=True,sharey=True,figsize=(8,10))
for i in range(16):
    axes[i%4][i//4].imshow(faceimages[i], cmap="gray")
plt.show()
```

Now, let us see what features we can extract from these face images.

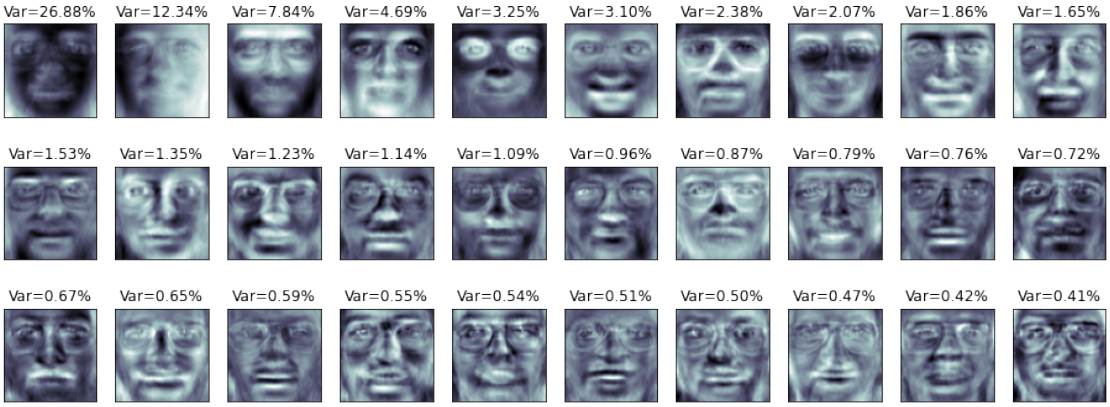
[20]:
```
#Perform PCA
from sklearn.decomposition import PCA

pca = PCA()
pca_pipe = Pipeline([("scaler",StandardScaler()), #Scikit learn PCA does not␣
 ↪standardize so we need to add
```

```
                ("pca",pca)])
pca_pipe.fit(faces.data)
```

[20]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('pca', PCA())])
```

The following plots the top 30 PCA components with how much variance does this feature explain.

[21]:
```
fig = plt.figure(figsize=(16, 6))
for i in range(30):
    ax = fig.add_subplot(3, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(pca.components_[i].reshape(im_shape),
              cmap=plt.cm.bone)
    ax.set_title(f"Var={pca.explained_variance_ratio_[i]:.2%}")
```



Amazing! We can see that the model has learned to focus on many features that we as humans also look at when trying to identify a face such as the nose,eyes, eyebrows, etc.

With this feature extraction, we can perform much more powerful learning.

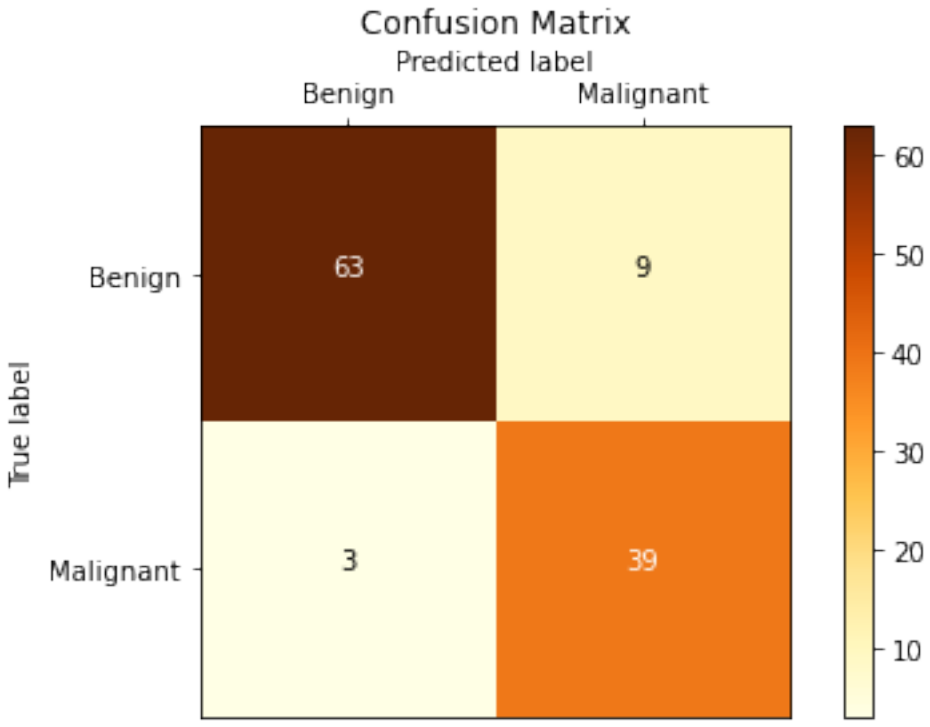
2.4.2 Feature Extraction for Classification

Lets see if we can use PCA to improve the accuracy of the decision tree classifier.

[22]:
```
#Without PCA
clf = DecisionTreeClassifier(random_state=0)
clf.fit(train, target)
predicted = clf.predict(test)

print("Accuracy without PCA")
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,predicted))
draw_confusion_matrix(target_test, predicted, ['Benign', 'Malignant'])

#With PCA
```

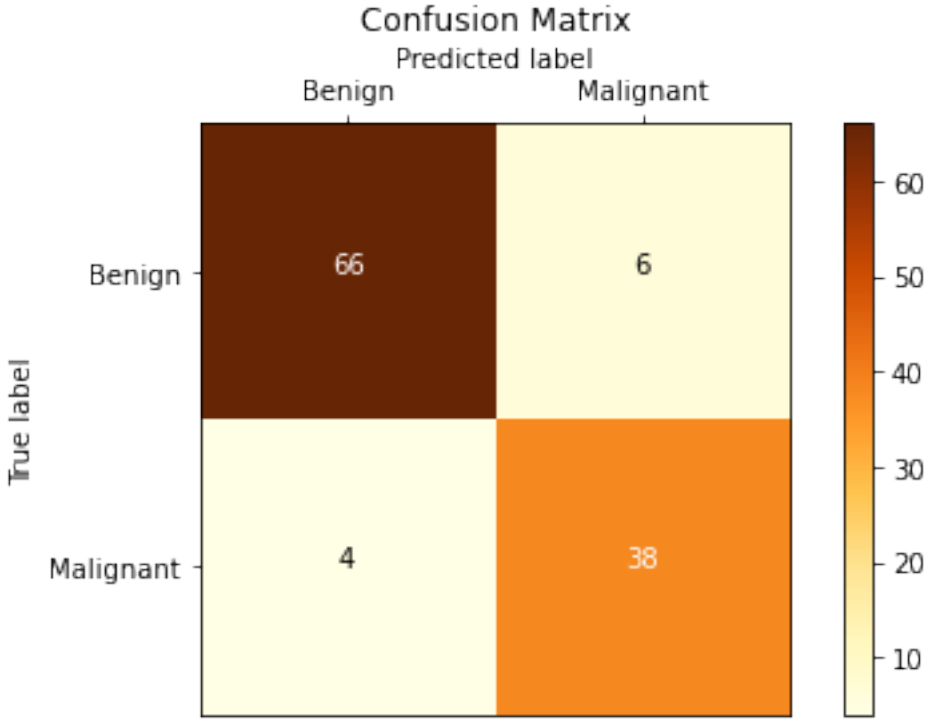
```
pca = PCA(n_components = 0.9) #Take components that explain at lest 90% variance

train_new = pca.fit_transform(train)
test_new = pca.transform(test)

clf_pca = DecisionTreeClassifier(random_state=0)
clf_pca.fit(train_new, target)
predicted = clf_pca.predict(test_new)

print("Accuracy with PCA")
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,predicted))
draw_confusion_matrix(target_test, predicted, ['Benign', 'Malignant'])
```

```
Accuracy without PCA
Accuracy:    0.894737
Confusion Matrix:
 [[63  9]
 [ 3 39]]
```




```
Accuracy with PCA
Accuracy:    0.912281
Confusion Matrix:
 [[66  6]
```

```
[ 4 38]]
```



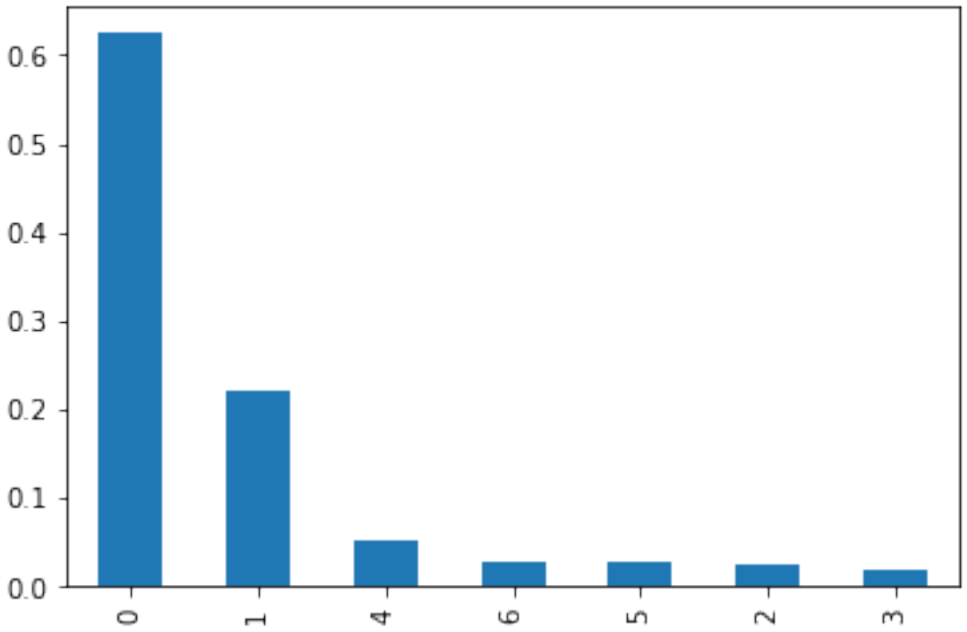
[23]:
```
print("Number of Features without PCA: ", train.shape[1])
print("Number of Features with PCA: ", train_new.shape[1])
```

```
Number of Features without PCA:  20
Number of Features with PCA:  7
```

Clearly, we get a much better accuracy for the model while using fewer features. But does the features the PCA thought were important the same features that the decision tree used. Lets look at the feature importance of the tree. The following plot numbers the first principal component as 0, the second as 1, and so forth.

[24]:
```
feature_names_new = list(range(train_new.shape[1]))
imp_pd = pd.Series(data = clf_pca.feature_importances_ ,index =␣
 ↪feature_names_new)
imp_pd= imp_pd.sort_values(ascending=False)
imp_pd.plot.bar()
```

[24]:
```
<AxesSubplot:>
```



Amazingly, the first and second components were the most important features in the decision tree. Thus, we can claim that PCA has significantly improved the performance of our model.

2.5 Unsupervised learning: Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups. One major algorithm we learned in class is the K-Means algorithm.

2.5.1 Evaluating K-Means performance

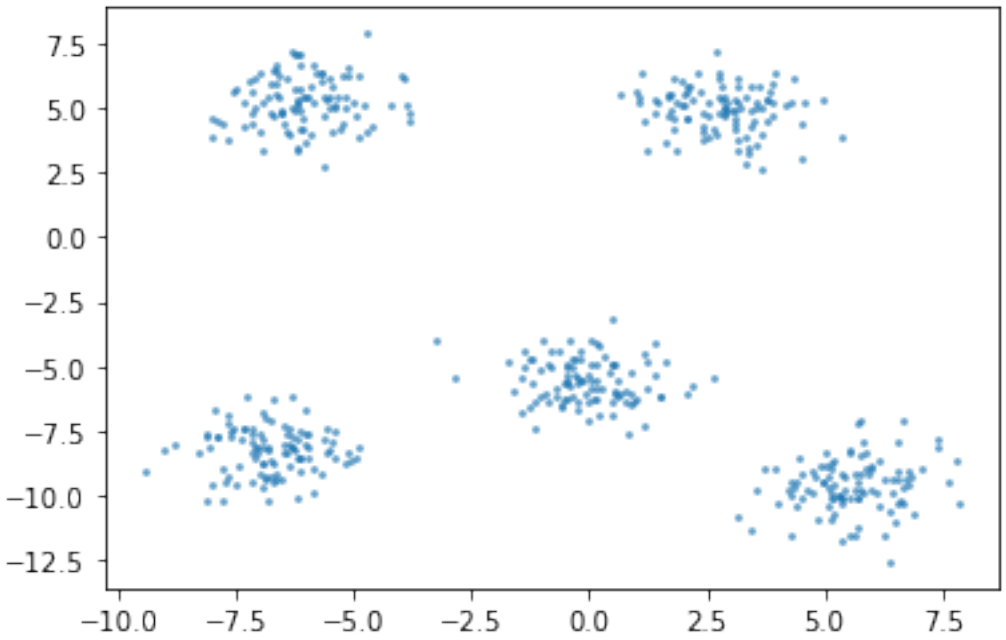
While there are many ways to evaluate the performance measure of clustering algorithsm, we will focus on the inertia score of the K-Means model. Inertia is another term for the sum of squared distances of samples to their closest cluster center.

Let us look at how the Inertia changes as a function of the number of clusters for an artificial dataset.

[25]:
```
#Artifical Dataset
X, y = make_blobs(
    n_samples=500,
    n_features=2,
    centers=5,
    cluster_std=1,
    center_box=(-10.0, 10.0),
    shuffle=True,
    random_state=10,
```

```
)  # For reproducibility

plt.scatter(X[:, 0], X[:, 1], marker=".", s=30, lw=0, alpha=0.7,edgecolor="k")
```

[25]: <matplotlib.collections.PathCollection at 0x7fcadf1862b0>

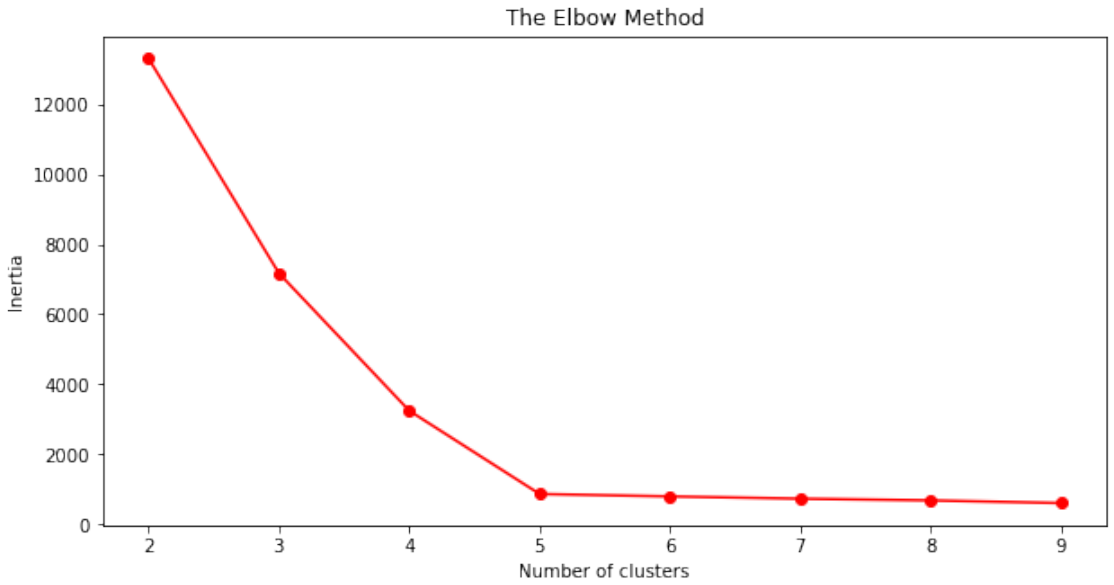


[26]:
```
ks = list(range(2,10))
inertia = []
for k in ks:
    kmeans = KMeans(n_clusters = k, init = 'k-means++', random_state = 0)
    kmeans.fit(X)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
    print(f"Inertia for K = {k}: {kmeans.inertia_}")
```

```
Inertia for K = 2: 13293.99746096155
Inertia for K = 3: 7169.578996856774
Inertia for K = 4: 3247.8674040695832
Inertia for K = 5: 872.8554968701876
Inertia for K = 6: 803.8466864258228
Inertia for K = 7: 739.5236191503766
Inertia for K = 8: 690.2530283275607
Inertia for K = 9: 614.5138307338652
```

[27]:
```
plt.figure(figsize=(10,5))
plt.plot(ks, inertia,marker='o',color='red')
```

```
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



From the plot, we can see that when the number of clusters of K-means is the correct number of clusters, Inertia starts decreasing at a much slower rate. This creates a kind of elbow shape in the graph. For K-means clustering, the elbow method selects the number of clusters where the elbow shape is formed. In this case, we see that this method would produce the correct number of clusters.

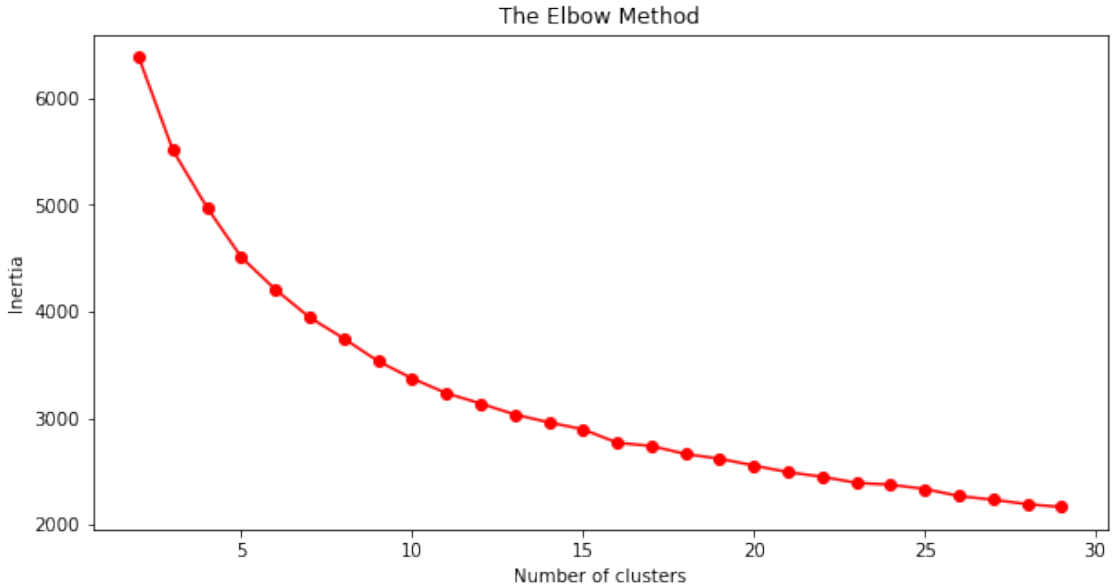
Lets try it on the cancer dataset.

[28]:
```
ks = list(range(2,30))
inertia = []
for k in ks:
    kmeans = KMeans(n_clusters = k, init = 'k-means++', random_state = 0)
    kmeans.fit(train)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
    print(f"Inertia for K = {k}: {kmeans.inertia_}")
```

```
Inertia for K = 2: 6381.27832595592
Inertia for K = 3: 5508.621446593708
Inertia for K = 4: 4972.231721973119
Inertia for K = 5: 4507.2671373660705
Inertia for K = 6: 4203.777246823878
Inertia for K = 7: 3942.659550896411
Inertia for K = 8: 3745.1124228292683
```

```
Inertia for K = 9: 3532.7225156022064
Inertia for K = 10: 3371.033467027838
Inertia for K = 11: 3232.4727580707367
Inertia for K = 12: 3135.1944201924525
Inertia for K = 13: 3033.3838427786477
Inertia for K = 14: 2958.3200036360377
Inertia for K = 15: 2893.7987635119034
Inertia for K = 16: 2767.804761705547
Inertia for K = 17: 2737.474710179063
Inertia for K = 18: 2662.1203080706655
Inertia for K = 19: 2617.9089069400507
Inertia for K = 20: 2553.961378449725
Inertia for K = 21: 2491.9133737078346
Inertia for K = 22: 2448.7776236009963
Inertia for K = 23: 2391.6445885404164
Inertia for K = 24: 2374.134578719017
Inertia for K = 25: 2334.7940109810734
Inertia for K = 26: 2267.9935217066172
Inertia for K = 27: 2233.585453239128
Inertia for K = 28: 2191.73940269357
Inertia for K = 29: 2165.2542076413133
```

[29]:
```
plt.figure(figsize=(10,5))
plt.plot(ks, inertia,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



Here we see that the elbow is not as cleanly defined. This may be due to the dataset not being a good fit for K-means. Regardless, we can still apply the elbow method by noting that the slow down happens around 7~14.

2.5.2 Kmeans on Eigenfaces

Now, lets see how K-means performs in clustering the face data with PCA.

[30]:
```
from sklearn.cluster import KMeans

n_clusters = 10 #We know there are 10 subjects
km = KMeans(n_clusters  = n_clusters,random_state=0)

pipe= Pipeline([("scaler",StandardScaler()), #First standardize
                ("pca",PCA()), #Transform using pca
          ("kmeans", km )]) #Then apply k means
```

[31]:
```
clusters = pipe.fit_predict(faces.data)
print(clusters)
```

```
[3 6 3 4 6 4 3 3 3 6 5 5 5 5 5 5 5 5 5 5 1 1 5 1 4 1 4 4 6 6 5 5 5 3 6 4 3
 5 5 6 4 1 1 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 3 3 7 7 3 4 7 3 7 7 3 7 0 6 3 6
 3 3 6 3 3 6 1 1 1 4 4 4 4 4 1 6 6 6 6 6 6 6 6 6 4 3 0 0 0 0 0 0 0 0 0 0 4
 1 1 1 1 4 1 6 6 4 5 5 4 4 5 5 4 4 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 8 3 3 3 3 8 6 8 3 3 4 4 1 1 4 4 4 4 4 4 3 6 4 6 3 3 3 3 3 3 7 7 7 7 7
 7 7 7 7 7 9 9 9 4 4 4 4 4 4 9 9 9 9 9 9 9 4 8 9 4 2 2 2 2 2 2 2 2 2 3 6
 1 4 1 4 1 6 4 4 8 8 8 8 5 8 8 8 8 8 6 5 6 5 5 5 6 4 5 6 1 1 1 1 1 1 3 1 1
 5 5 5 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 1 4 2 2 2 9 4 4 9 8 2 2 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 7 7 7 7 7 7 5 7 7 7 9 9 9 9 9 9 9 9 9 9 2 2 2
 2 2 2 2 2 2 2 9 9 9 9 4 6 6 1 4 4 3 8 8 8 7 8 8 8 8 8 1 1 1 1 1 1 1 1 1 1
 4 1 1 6 1 4 6 6 4 1 2 2 2 2 2 2 2 2 2 2 6 4 3 4 3 1 4 1 4 4]
```

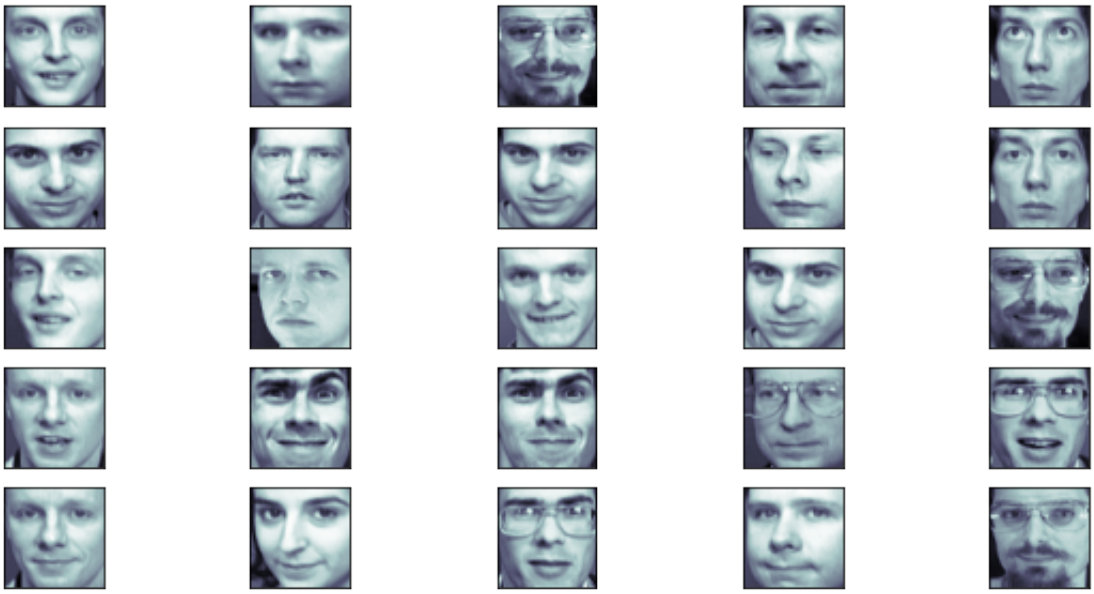
[32]:
```
for labelID in range(n_clusters):
    # find all indexes into the `data` array that belong to the
    # current label ID, then randomly sample a maximum of 25 indexes
    # from the set
    idxs = np.where(clusters == labelID)[0]
    idxs = np.random.choice(idxs, size=min(25, len(idxs)),
        replace=False)

    # Extract the sampled indexes
    id_face = faces.images[idxs]

    #Plots sampled faces
    fig = plt.figure(figsize=(10,5))
    for i in range(min(25,len(idxs))):
        ax = fig.add_subplot(5, 5, i + 1, xticks=[], yticks=[])
```

```
        ax.imshow(id_face[i],
                  cmap=plt.cm.bone)
    fig.suptitle(f"Id={labelID}")
```

Id=2

Id=3

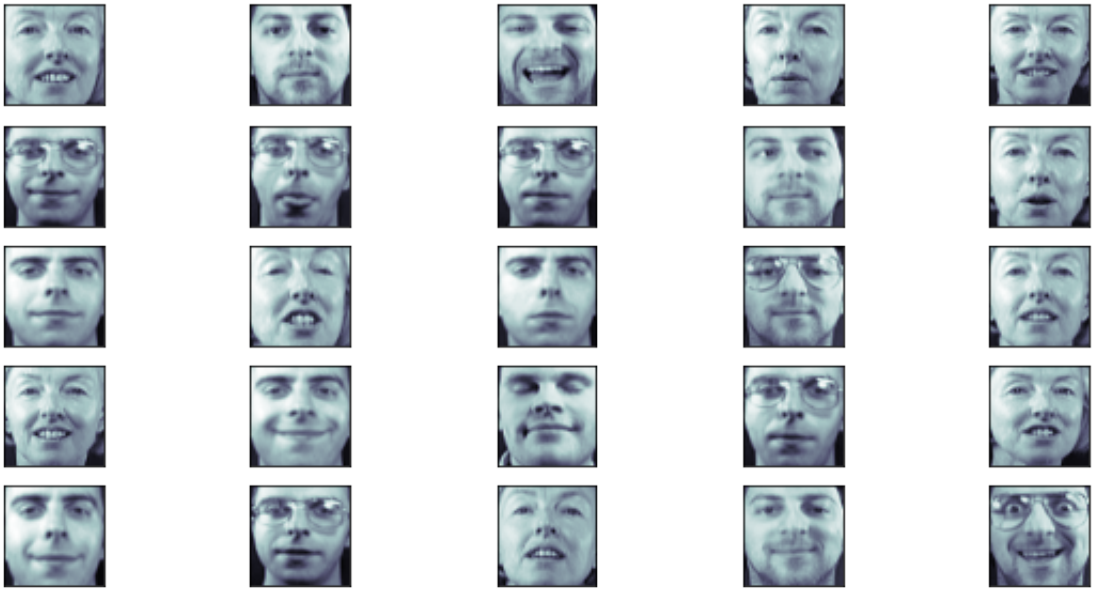
Id=4

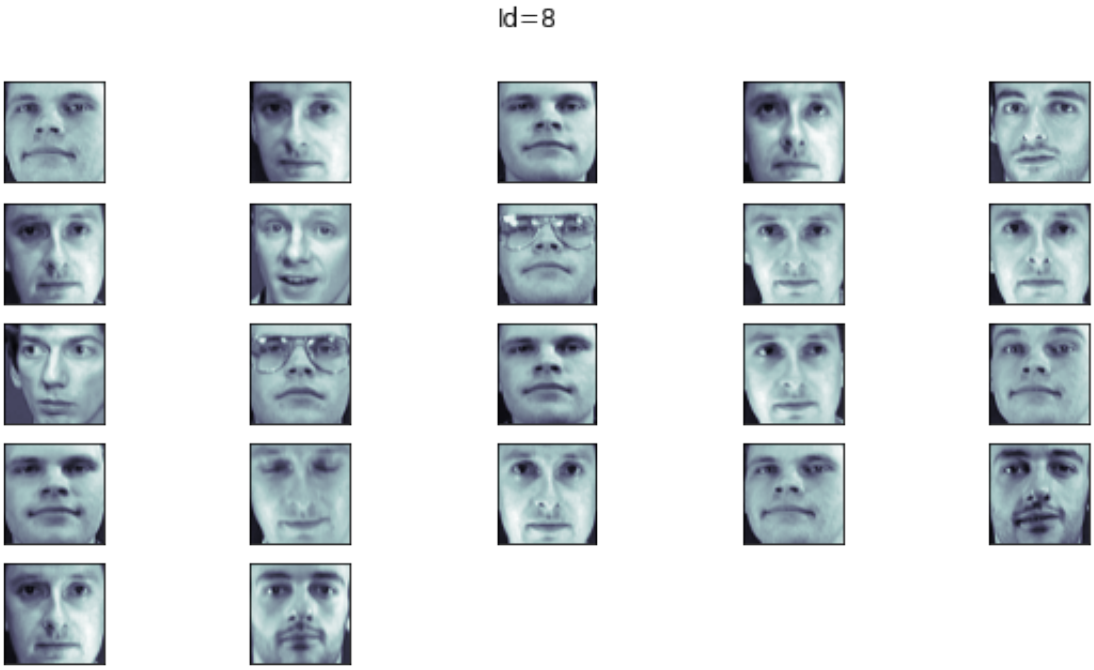


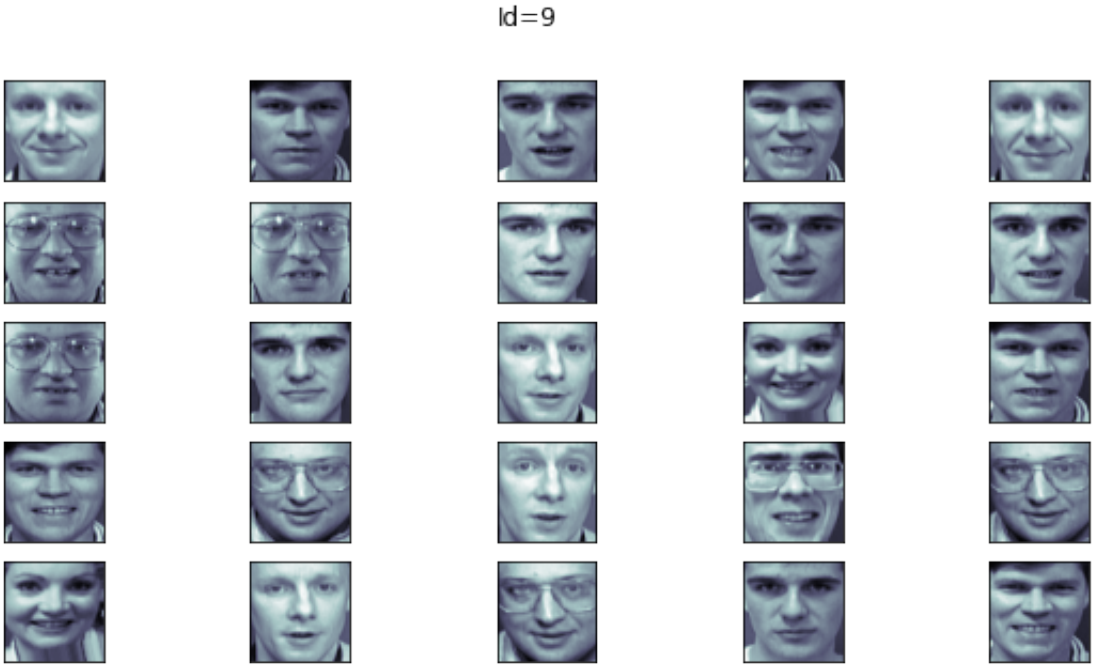
Id=5

Id=6

Id=7







While the algorithm isn't perfect, we can see that K-means with PCA is picking up on some facial similarity or similar expressions.

3 (100 pts) Todo: Use new methods to classify heart disease

To compare how these new models perform with the other models discussed in the course, we will apply these new models on the heart disease dataset that was used in project 2.

3.1 Background: The Dataset (Recap)

For this exercise we will be using a subset of the UCI Heart Disease dataset, leveraging the fourteen most commonly used attributes. All identifying information about the patient has been scrubbed. You will be asked to classify whether a patient is suffering from heart disease based on a host of potential medical factors.

The dataset includes 14 columns. The information provided by each column is as follows:

age: Age in years

sex: (1 = male; 0 = female)

cp: Chest pain type (0 = asymptomatic; 1 = atypical angina; 2 = non-anginal pain; 3 = typical angina)

trestbps: Resting blood pressure (in mm Hg on admission to the hospital)

chol: cholesterol in mg/dl

fbs Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)

restecg: Resting electrocardiographic results (0= showing probable or definite left ventricular hypertrophy by Estes' criteria; 1 = normal; 2 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV))

thalach: Maximum heart rate achieved

exang: Exercise induced angina (1 = yes; 0 = no)

oldpeak: Depression induced by exercise relative to rest

slope: The slope of the peak exercise ST segment (0 = downsloping; 1 = flat; 2 = upsloping)

ca: Number of major vessels (0-3) colored by flourosopy

thal: 1 = normal; 2 = fixed defect; 7 = reversable defect

sick: Indicates the presence of Heart disease (True = Disease; False = No disease)

3.2 Preprocess Data

This part is done for you since you would have already completed it in project 2. Use the train, target, test, and target_test for all future parts. We also provide the column names for each transformed column for future use.

[33]:
```
#Preprocess Data

#Load Data
data = pd.read_csv('datasets/heartdisease.csv')
```

```
#Transform target feature into numerical
le = LabelEncoder()
data['target'] = le.fit_transform(data['sick'])
data = data.drop(["sick"], axis =1)

#Split target and data
y = data["target"]
x = data.drop(["target"],axis = 1)

#Train test split
#40% in test data as was in project 2
train_raw, test_raw, target, target_test = train_test_split(x,y, test_size=0.4,␣
 ↪stratify= y, random_state=0)

#Feature Transformation
#This is the only change from project 2 since we replaced standard scaler to␣
 ↪minmax
#This was done to ensure that the numerical features were still of the same␣
 ↪scale
#as the one hot encoded features
num_pipeline = Pipeline([
    ('minmax', MinMaxScaler())
])

heart_num = train_raw.drop(['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',␣
 ↪'ca','thal'], axis=1)
numerical_features = list(heart_num)
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',␣
 ↪'ca','thal']

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", OneHotEncoder(categories='auto'), categorical_features),
])

#Transform raw data
train = full_pipeline.fit_transform(train_raw)
test = full_pipeline.transform(test_raw) #Note that there is no fit calls

#Extracts features names for each transformed column
feature_names = full_pipeline.get_feature_names_out(list(x.columns))
```

[34]:
```
print("Column names after transformation by pipeline: ", feature_names)
```

```
Column names after transformation by pipeline:  ['num__age' 'num__trestbps'
'num__chol' 'num__thalach' 'num__oldpeak'
 'cat__sex_0' 'cat__sex_1' 'cat__cp_0' 'cat__cp_1' 'cat__cp_2' 'cat__cp_3'
```

```
'cat__fbs_0' 'cat__fbs_1' 'cat__restecg_0' 'cat__restecg_1'
'cat__restecg_2' 'cat__exang_0' 'cat__exang_1' 'cat__slope_0'
'cat__slope_1' 'cat__slope_2' 'cat__ca_0' 'cat__ca_1' 'cat__ca_2'
'cat__ca_3' 'cat__ca_4' 'cat__thal_0' 'cat__thal_1' 'cat__thal_2'
'cat__thal_3']
```

The following shows the baseline accuracy of simply classifying every sample as the majority class.

[35]:
```
#Baseline accuracy of using the majority class
ct = target_test.value_counts()
print("Counts of each class in target_test: ")
print(ct)
print("Baseline Accuraccy of using Majority Class: ", np.max(ct)/np.sum(ct))
```

```
Counts of each class in target_test:
0    66
1    56
Name: target, dtype: int64
Baseline Accuraccy of using Majority Class:  0.5409836065573771
```

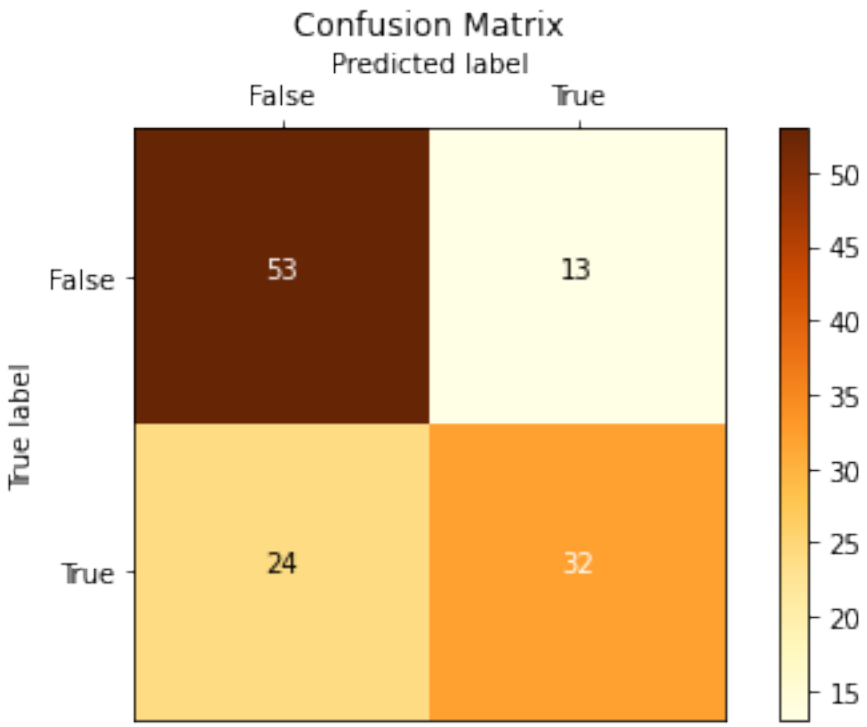
3.3 (25 pts) Decision Trees

3.3.1 [5 pts] Apply Decision Tree on Train Data

Apply the decision tree on the **train data** with default parameters of the DecisionTreeClassifier. **Report the accuracy and print the confusion matrix**. Make sure to use random_state = 0 so that your results match ours.

[37]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state = 0)
dt.fit(train, target)
dtpredicted = dt.predict(test)
print("%-12s %f" % ('Accuracy:', metrics.
  ↪accuracy_score(target_test,dtpredicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,dtpredicted))
draw_confusion_matrix(target_test, dtpredicted, ['False', 'True'])
```

```
Accuracy:    0.696721
Confusion Matrix:
 [[53 13]
 [24 32]]
```



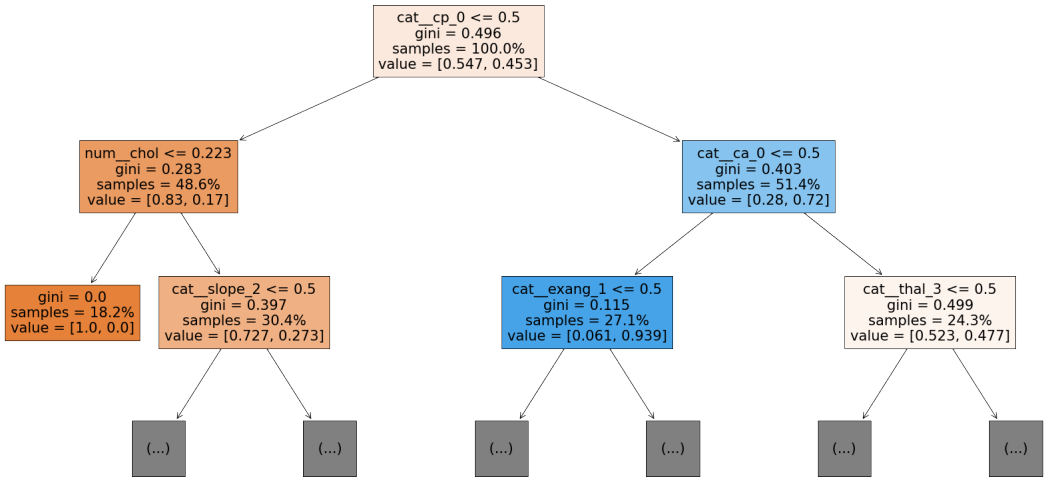
3.3.2 [5 pts] Visualize the Decision Tree

Visualize the first two layers of the decision tree that you trained.

[38]:
```
plt.figure(figsize = (30,15))
tree.plot_tree(dt,max_depth=2, proportion=True,feature_names=feature_names,␣
↪filled=True)
```

[38]:
```
[Text(0.4230769230769231, 0.875, 'cat__cp_0 <= 0.5\ngini = 0.496\nsamples =
100.0%\nvalue = [0.547, 0.453]'),
 Text(0.15384615384615385, 0.625, 'num__chol <= 0.223\ngini = 0.283\nsamples =
48.6%\nvalue = [0.83, 0.17]'),
 Text(0.07692307692307693, 0.375, 'gini = 0.0\nsamples = 18.2%\nvalue = [1.0,
0.0]'),
 Text(0.23076923076923078, 0.375, 'cat__slope_2 <= 0.5\ngini = 0.397\nsamples =
30.4%\nvalue = [0.727, 0.273]'),
 Text(0.15384615384615385, 0.125, '\n  (…)  \n'),
 Text(0.3076923076923077, 0.125, '\n  (…)  \n'),
 Text(0.6923076923076923, 0.625, 'cat__ca_0 <= 0.5\ngini = 0.403\nsamples =
51.4%\nvalue = [0.28, 0.72]'),
 Text(0.5384615384615384, 0.375, 'cat__exang_1 <= 0.5\ngini = 0.115\nsamples =
27.1%\nvalue = [0.061, 0.939]'),
 Text(0.46153846153846156, 0.125, '\n  (…)  \n'),
 Text(0.6153846153846154, 0.125, '\n  (…)  \n'),
```

```
 Text(0.8461538461538461, 0.375, 'cat__thal_3 <= 0.5\ngini = 0.499\nsamples =
24.3%\nvalue = [0.523, 0.477]'),
 Text(0.7692307692307693, 0.125, '\n  (…)  \n'),
 Text(0.9230769230769231, 0.125, '\n  (…)  \n')]
```



What is the gini index improvement of the first split?

[39]:
```
0.496-0.283*0.486-0.403*0.514
```

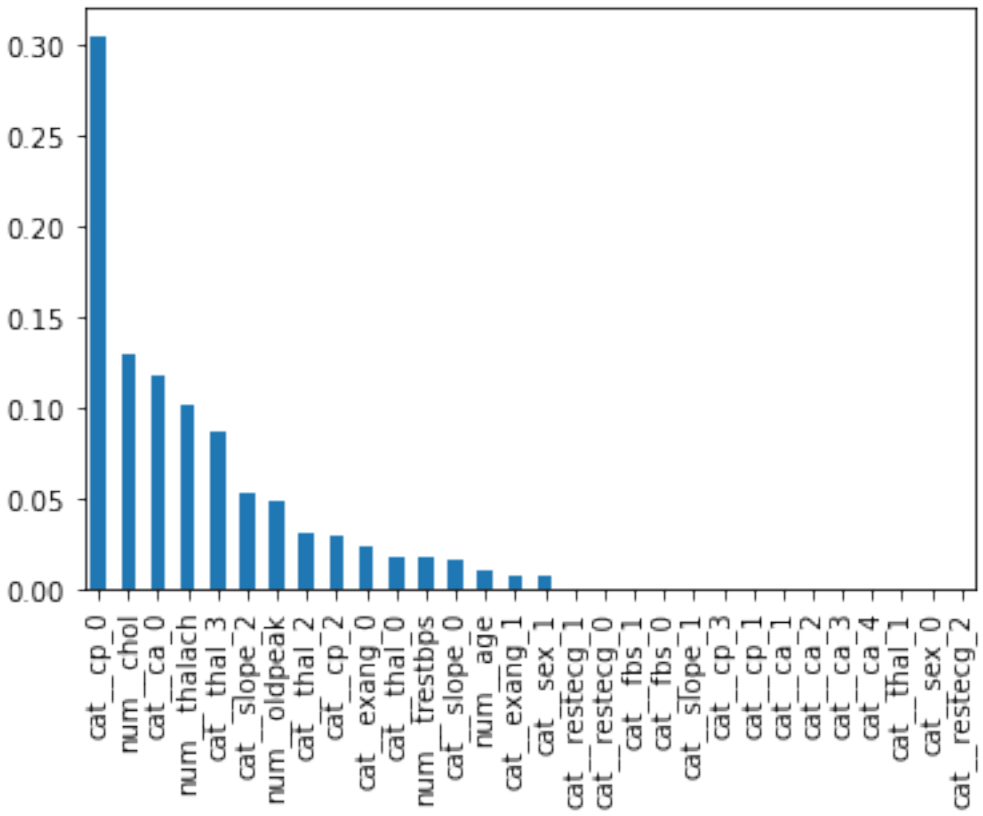
[39]:
```
0.15131999999999998
```

Response: gini index improvement of the first split is around 0.15

3.3.3 [5 pts] Plot the importance of each feature for the Decision Tree

[40]:
```
imp_pd = pd.Series(data = dt.feature_importances_ ,index = feature_names)
imp_pd= imp_pd.sort_values(ascending=False)
imp_pd.plot.bar()
```

[40]:
```
<AxesSubplot:>
```



How many features have non-zero importance for the Decision Tree? If we remove the features with zero importance, will it change the decision tree for the same sampled dataset?

Response: **16 non-zero importance**. It will **not change** decision tree for same sampled dataset because they have zero importance which means they're not contribute to the decision making.

3.3.4 [10 pts] Optimize Decision Tree

While the default Decision Tree performs fairly well on the data, lets see if we can improve performance by optimizing the parameters.

Run a GridSearchCV with 3-Fold Cross Validation for the Decision Tree. Find the best model parameters amongst the following:

- max_depth = [1,2,3,4,5,10,15]
- min_samples_split = [2,4,6,8]
- criterion = ["gini", "entropy"]

After using GridSearchCV, print the best model parameters and the best score.

[41]:
```
parameters = [
    {"max_depth": [1,2,3,4,5,10,15],
     "min_samples_split" : [2,4,6,8],
```

```
    "criterion" : ["gini", "entropy"]}
]

k = 3
kf = KFold(n_splits=k, random_state=None)

grid = GridSearchCV(dt , parameters, cv = kf, scoring = "accuracy")
grid.fit(train,target)
```

[41]:
```
GridSearchCV(cv=KFold(n_splits=3, random_state=None, shuffle=False),
             estimator=DecisionTreeClassifier(random_state=0),
             param_grid=[{'criterion': ['gini', 'entropy'],
                          'max_depth': [1, 2, 3, 4, 5, 10, 15],
                          'min_samples_split': [2, 4, 6, 8]}],
             scoring='accuracy')
```

[42]:
```
res= pd.DataFrame(grid.cv_results_)
res.loc[res["rank_test_score"]==1]
```

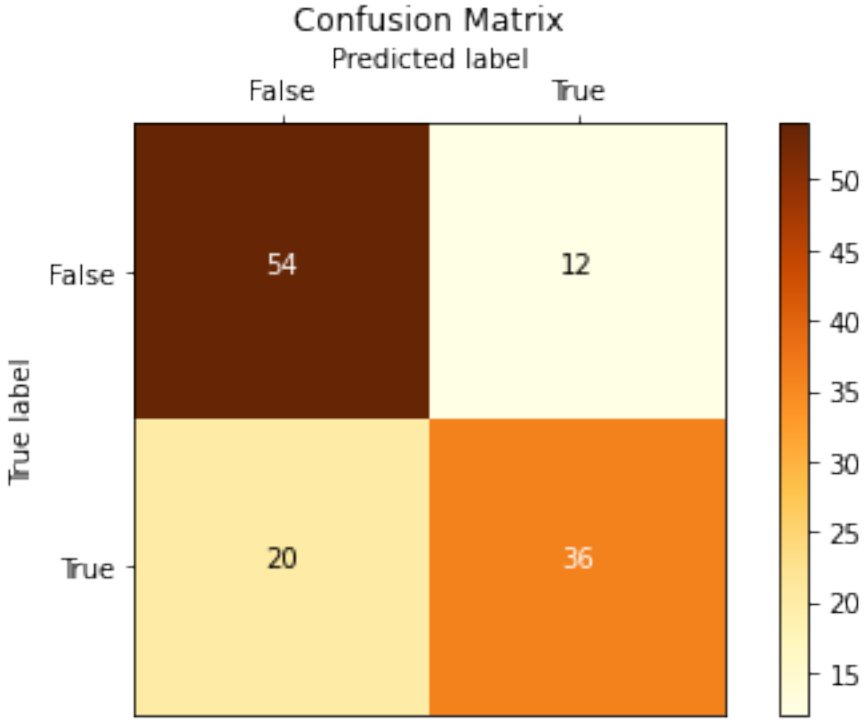
[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_split	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
43	0.001451	0.000132	0.000414	0.000024	entropy	4	8	{'criterion': 'entropy', 'max_depth': 4, 'min_…	0.737705	0.75	0.816667	0.768124	0.03469	1
51	0.001664	0.000160	0.000435	0.000027	entropy	10	8	{'criterion': 'entropy', 'max_depth': 10, 'min…	0.737705	0.75	0.816667	0.768124	0.03469	1
55	0.001581	0.000154	0.000475	0.000102	entropy	15	8	{'criterion': 'entropy', 'max_depth': 15, 'min…	0.737705	0.75	0.816667	0.768124	0.03469	1

Using the best model you have, report the test accuracy and print out the confusion matrix

[43]:
```
best = grid.predict(test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,best)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test,best))
draw_confusion_matrix(target_test, best, ['False', 'True'])
```

```
Accuracy:    0.737705
Confusion Matrix:
 [[54 12]
 [20 36]]
```



3.4 (20 pts) Multi-Layer Perceptron

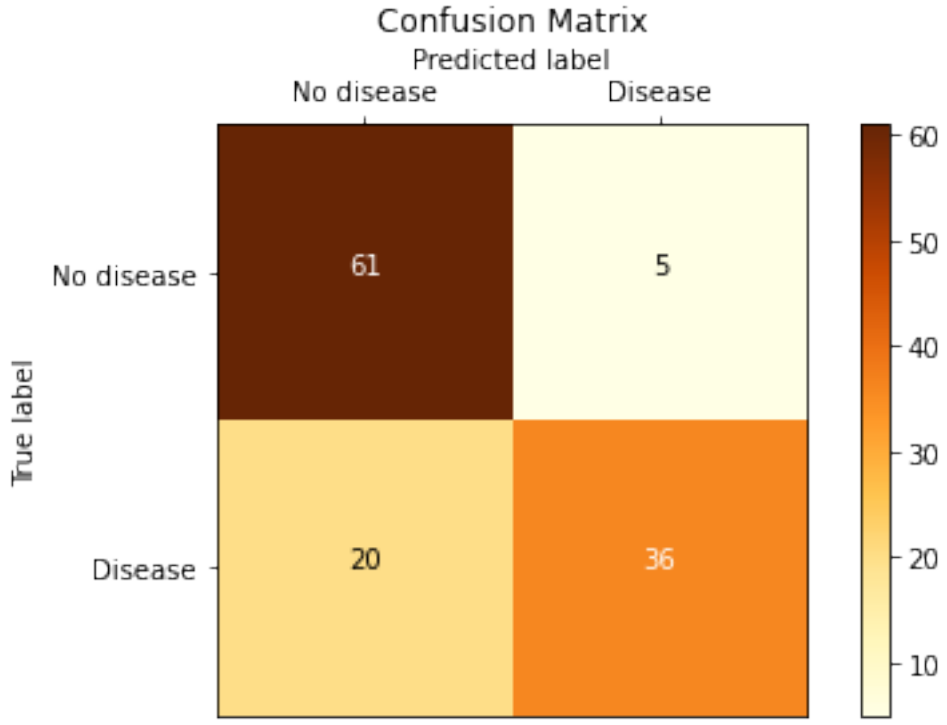
3.4.1 [5 pts] Applying a Multi-Layer Perceptron

Apply the MLP on the **train data** with hidden_layer_sizes=(100,100) and max_iter = 800. **Report the accuracy and print the confusion matrix**. Make sure to set random_state=0.

[44]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100,100), max_iter = 800)
mlp.fit(train, target)
mlppredicted = mlp.predict(test)
print("%-12s %f" % ('Accuracy:', metrics.
 ↪accuracy_score(target_test,mlppredicted)))
```

```
print("Confusion Matrix: \n", metrics.
  ↪confusion_matrix(target_test,mlppredicted))
draw_confusion_matrix(target_test, mlppredicted, ['No disease', 'Disease'])
```

```
Accuracy:    0.795082
Confusion Matrix:
 [[61  5]
 [20 36]]
```



3.4.2 [10 pts] Speedtest between Decision Tree and MLP

Let us compare the training times and prediction times of a Decision Tree and an MLP. **Time how long it takes for a Decision Tree and an MLP to perform a .fit operation (i.e. training the model). Then, time how long it takes for a Decision Tree and an MLP to perform a .predict operation (i.e. predicting the testing data). Print out the timings and specify which model was quicker for each operation.** We recommend using the time python module to time your code. An example of the time module was shown in project 2. Use the default Decision Tree Classifier and the MLP with the previously mentioned parameters.

[45]:
```
t0 = time.time()
dt.fit(train, target)
t1 = time.time()
t00 = time.time()
dt.predict(test)
```

```
t11 = time.time()
print("Decision Tree Performing Time : ", t1-t0)
print("Decision Tree Predicting Time : ", t11-t00)
```

```
Decision Tree Performing Time :  0.005759000778198242
Decision Tree Predicting Time :  0.0007650852203369141
```

[46]:
```
t2 = time.time()
mlp.fit(train, target)
t3 = time.time()
t22 = time.time()
mlp.predict(test)
t33 = time.time()
print("Multi-Layer Perceptron Performing Time : ", t3-t2)
print("Multi-Layer Perceptron Predicting Time : ", t33-t22)
```

```
Multi-Layer Perceptron Performing Time :  0.9729678630828857
Multi-Layer Perceptron Predicting Time :  0.0005841255187988281
```

a. fit operation: Decision Tree is quicker

b. predict operation: Decision Tree is quicker

3.4.3 [5 pts] Compare and contrast Decision Trees and MLPs.

Describe at least one advantage and disadvantage of using an MLP over a Decision Tree.

Response:

MLP:

Advantage: More accurate because it's universal approximation and using ensemble methods.

Disadvanatge: Take more time because MLPs involve complex computations and optimization algorithms during both the forward and backward passes of training.

Decision Tree:

Advantage: Time efficiency because it has less parameters than MLP.

Disadvanatge: Not that accurate compare with MLP because it not only sensitive to small change in training data but also replying on feature split. If the decision tree fails to find the most informative splits or makes suboptimal splitting decisions, its accuracy can be lower. MLPs, being universal function approximators, can learn complex decision boundaries without relying solely on predefined feature splits.

3.5 (35 pts) PCA

3.5.1 [5 pts] Transform the train data using PCA

Train a PCA model to project the train data on the top 10 components. **Print out the 10 principal components**. Look at the documentation of PCA for reference.

[47]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=10)
pca.fit(train)
print(pca.components_)
```

```
[[ 6.09946635e-02  4.03486405e-02  1.92458119e-02 -1.01732201e-01
   1.10715411e-01 -1.23314342e-01  1.23314342e-01  3.42653325e-01
  -1.34589184e-01 -2.09361216e-01  1.29707586e-03 -1.28849271e-02
   1.28849271e-02  1.96937289e-01 -1.98556306e-01  1.61901735e-03
  -3.50544191e-01  3.50544191e-01  4.59558653e-02  2.94123242e-01
  -3.40079107e-01 -2.05535178e-01  7.46326307e-02  8.34805264e-02
   6.75833376e-02 -2.01613171e-02 -3.90382012e-04  4.43852832e-02
  -3.14081117e-01  2.70086216e-01]
 [ 5.23178867e-02  2.89025067e-02  3.82650356e-02 -7.33246302e-03
  -3.72850287e-03  4.44422145e-01 -4.44422145e-01  7.36224643e-02
  -3.17147806e-02 -2.86078655e-02 -1.32998182e-02 -2.10653488e-02
   2.10653488e-02  4.25893739e-01 -4.44476340e-01  1.85826015e-02
   1.84907981e-02 -1.84907981e-02 -2.18311052e-02  1.43545701e-01
  -1.21714596e-01  2.01177502e-02 -3.19965459e-02  3.69926555e-02
   3.82437776e-03 -2.89382375e-02  3.36806325e-03  3.01278536e-03
   2.90071072e-01 -2.96451921e-01]
 [-4.27615964e-02 -3.74214268e-02  3.54062787e-03 -4.73356567e-02
   1.80128286e-02  3.06998751e-01 -3.06998751e-01  9.34724674e-02
   3.29171966e-02 -9.89118027e-02 -2.74778613e-02  1.99901512e-01
  -1.99901512e-01 -4.31180483e-01  4.09965787e-01  2.12146963e-02
  -1.32234559e-01  1.32234559e-01 -2.51484162e-02  3.20908646e-01
  -2.95760230e-01  2.91494765e-01 -1.81653124e-01 -5.23591628e-02
  -4.72251996e-02 -1.02572786e-02  3.55496902e-03 -3.31268376e-02
   2.75485826e-02  2.02328599e-03]
 [-1.08526684e-02  5.12885516e-02  2.04370575e-02  4.68524214e-02
   3.20774050e-02 -2.89931779e-02  2.89931779e-02 -3.49991818e-02
   7.51886952e-02 -1.47222546e-01  1.07033033e-01  7.44448341e-02
  -7.44448341e-02  1.91691186e-01 -1.93986022e-01  2.29483659e-03
   3.73813856e-01 -3.73813856e-01  4.15962863e-02  9.05417530e-02
  -1.32138039e-01  3.84693769e-01 -4.11618177e-01  1.35048835e-03
   4.40451383e-02 -1.84712183e-02  2.25007917e-03  4.15374728e-02
  -3.66278328e-01  3.22490776e-01]
 [ 4.62793806e-02  1.97044763e-02 -3.81961161e-03 -3.50990198e-02
   2.13481873e-03 -6.41167044e-02  6.41167044e-02 -4.03420284e-01
  -3.78482985e-02  3.29558992e-01  1.11709591e-01 -2.54225373e-01
   2.54225373e-01 -6.93684511e-02  6.17493498e-02  7.61910128e-03
   1.79472704e-01 -1.79472704e-01 -6.91121011e-02  4.99776798e-01
  -4.30664697e-01 -1.75364478e-01  1.54506279e-01 -1.61821481e-03
  -4.09490841e-03  2.65713222e-02  3.76025899e-03  4.67297745e-02
   2.27974035e-03 -5.27697739e-02]
 [-6.83684670e-02 -2.10631304e-02 -3.64071029e-02  5.76781333e-03
  -4.37663939e-02 -3.59388014e-01  3.59388014e-01 -5.08255318e-03
```

```
  5.58430800e-02 -1.22172796e-01  7.14122688e-02 -1.00496367e-01
  1.00496367e-01  8.79697953e-02 -9.14477532e-02  3.47795793e-03
 -1.81198958e-01  1.81198958e-01  8.40745880e-03  1.15245549e-01
 -1.23653008e-01  4.89977935e-01 -2.22291278e-01 -1.55482017e-01
 -9.52207407e-02 -1.69838995e-02  9.97475772e-03  8.20113234e-02
  3.04306327e-01 -3.96292408e-01]
[-1.78126881e-02  7.30869047e-02  2.01669928e-02  1.82617883e-02
  2.83650271e-02  1.89632044e-01 -1.89632044e-01 -2.28914692e-01
 -9.80404345e-02  3.24999333e-01  1.95579325e-03 -3.62677848e-01
  3.62677848e-01  2.76714162e-02 -1.72866529e-02 -1.03847633e-02
 -2.96007862e-01  2.96007862e-01  6.46671292e-02 -1.88716817e-01
  1.24049688e-01  3.12705923e-01 -1.79722278e-01 -8.23815428e-02
 -1.80852435e-02 -3.25168582e-02  3.17733224e-02 -9.35340776e-02
 -2.02189819e-01  2.63950574e-01]
[ 4.33522625e-02  5.42145161e-02 -3.47043818e-02 -1.53334622e-02
  1.85078255e-02  6.96158719e-02 -6.96158719e-02  3.34668254e-01
  1.11230064e-01 -4.14074834e-01 -3.18234845e-02 -4.71838238e-01
  4.71838238e-01 -1.88415072e-01  1.85305199e-01  3.10987336e-03
  1.41716965e-01 -1.41716965e-01  1.93816114e-02 -4.76355708e-02
  2.82539594e-02 -1.17280675e-01 -1.66524382e-01  2.11683037e-01
  9.21390983e-02 -2.00170785e-02  1.10368046e-02  1.24143564e-01
 -3.23015807e-03 -1.31950210e-01]
[-5.97613225e-02 -3.69370641e-02  6.61316204e-03  5.61585865e-02
 -4.80614388e-02  7.17304750e-02 -7.17304750e-02 -3.32520420e-01
  6.47193101e-01 -4.43929000e-01  1.29256319e-01 -5.00439300e-02
  5.00439300e-02  7.30899332e-02 -3.35107082e-02 -3.95792250e-02
 -8.97283451e-02  8.97283451e-02 -4.74533090e-03  1.65062850e-02
 -1.17609541e-02 -7.54476449e-04  3.32370630e-01 -2.23757746e-01
 -7.77035754e-02 -3.01548322e-02 -1.30691596e-02 -1.15565949e-01
 -2.84080137e-02  1.57043123e-01]
[-3.95638796e-02  1.44987850e-02 -1.86389673e-03  7.34567783e-02
  9.64455093e-02 -1.21837970e-02  1.21837970e-02 -3.66301726e-01
  2.07778591e-01 -1.80298647e-02  1.76553000e-01  1.08577675e-01
 -1.08577675e-01  1.55218475e-02 -3.36271181e-03 -1.21591357e-02
 -1.68157144e-01  1.68157144e-01 -3.17943315e-02 -1.03539031e-02
  4.21482346e-02 -2.23925153e-01 -4.10777625e-01  6.63213019e-01
 -8.58159468e-02  5.73057053e-02 -1.30077281e-02  1.12209974e-01
 -5.47823656e-03 -9.37240090e-02]]
```

3.5.2 [5 pts] Percentage of variance explained by top 10 principal components

Using PCA's "explained_variance_ratio_", print the percentage of variance explained by the top 10 principal components.

[48]:
```
pca.explained_variance_ratio_[:10]
```

[48]:
```
array([0.23862094, 0.1360394 , 0.10034179, 0.08239361, 0.07495304,
       0.06591197, 0.05919248, 0.04935616, 0.0404145 , 0.0299425 ])
```

3.5.3 [5 pts] Transform the train and test data into train_pca and test_pca using PCA

Note: Use fit_transform for train and transform for test

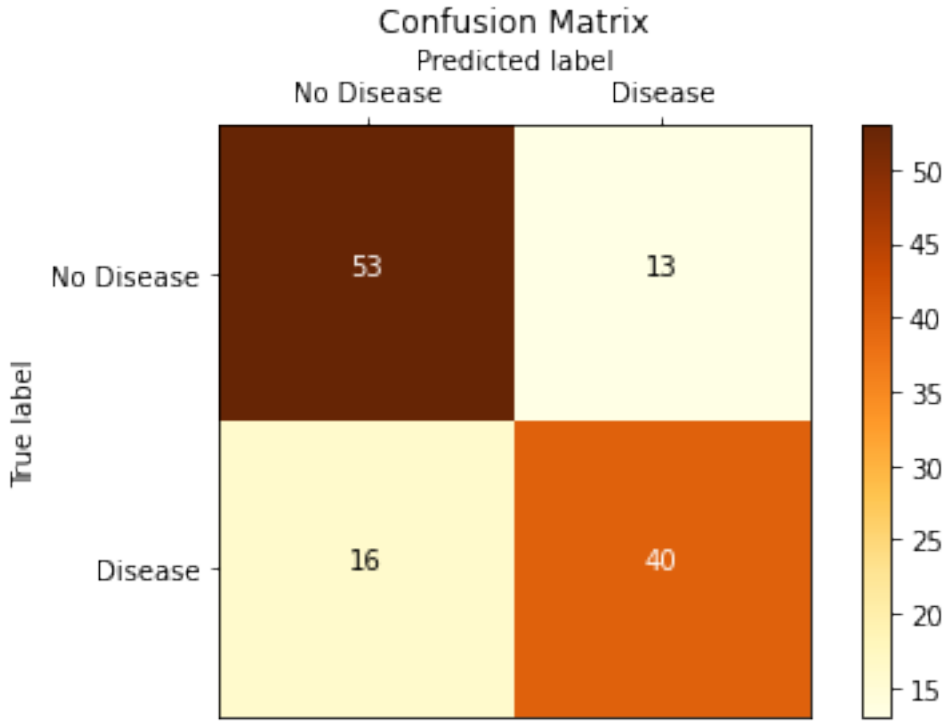
[49]:
```
train_pca = pca.fit_transform(train)
test_pca = pca.transform(test)
```

3.5.4 [5 pts] PCA+Decision Tree

Train the default Decision Tree Classifier using train_pca. **Report the accuracy using test_pca and print the confusion matrix.**

[50]:
```
pcadt=DecisionTreeClassifier(random_state = 0)
pcadt.fit(train_pca, target)
pcadtpredicted = pcadt.predict(test_pca)
print("%-12s %f" % ('Accuracy:', metrics.
 ↪accuracy_score(target_test,pcadtpredicted)))
print("Confusion Matrix: \n", metrics.
 ↪confusion_matrix(target_test,pcadtpredicted))
draw_confusion_matrix(target_test, pcadtpredicted, ['No Disease', 'Disease'])
```

```
Accuracy:    0.762295
Confusion Matrix:
 [[53 13]
 [16 40]]
```



Does the model perform better with or without PCA?

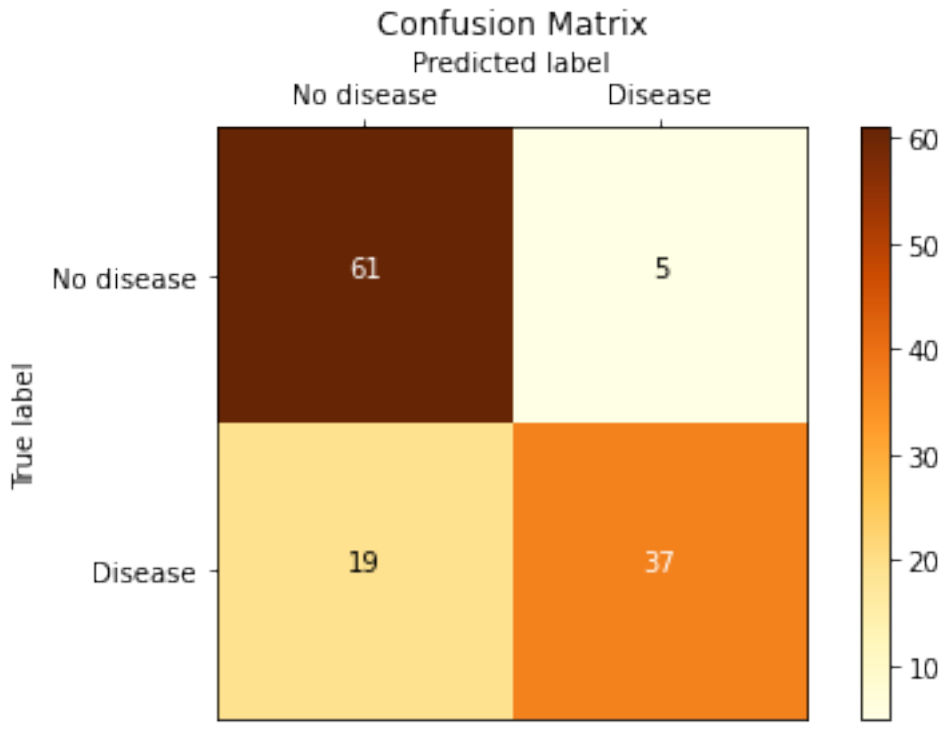
Response: Yes model with PCA perform better than without. It improve accuracy from 70% to 76%.

3.5.5 [5 pts] PCA+MLP

Train the MLP classifier with the same parameters as before using train_pca. **Report the accuracy using test_pca and print the confusion matrix**.

[51]:
```
pcamlp = MLPClassifier(hidden_layer_sizes=(100,100), max_iter = 800)
pcamlp.fit(train_pca, target)
pcamlppredicted = pcamlp.predict(test_pca)
print("%-12s %f" % ('Accuracy:', metrics.
 ↪accuracy_score(target_test,pcamlppredicted)))
print("Confusion Matrix: \n", metrics.
 ↪confusion_matrix(target_test,pcamlppredicted))
draw_confusion_matrix(target_test, pcamlppredicted, ['No disease', 'Disease'])
```

```
Accuracy:    0.803279
Confusion Matrix:
 [[61  5]
 [19 37]]
```



Does the model perform better with or without PCA?

Response: No, MLP with PCA perform worse than without PCA. It drop accuracy from 81% to 79%.

3.5.6 [10 pts] Pros and Cons of PCA

In your own words, provide at least two pros and at least two cons for using PCA

Response:

Pros:

PCA can **reduce dimensionality** of the input feature space by transforming it into a lower-dimensional space while preserving the most important information. Decision trees often struggle with high-dimensional data, so PCA can help alleviate the curse of dimensionality.

PCA creates new features, called principal components, which are orthogonal and capture the most significant variations in the data. By using PCA, the resulting **features are independent** of each other, which can be advantageous for decision trees that assume feature independence.

Cons:

PCA transforms the original features into a new set of features, making it **more challenging to interpret** the results. The principal components do not directly correspond to the original features. Decision trees are known for their interpretability, but using PCA can reduce their interpretability since the transformed features are not directly meaningful or easily understandable.

The transformation inherently involves **some loss of information**. Depending on the amount of variance retained in the selected principal components, some less significant but potentially useful information may be discarded.

3.6 (20 pts) K-Means Clustering

3.6.1 [5 pts] Apply K-means to the train data and print out the Inertia score

Use n_cluster = 5 and random_state = 0.

[52]:
```
kmeans = KMeans(n_clusters = 5, random_state = 0)
kmeans.fit(train)
print(kmeans.inertia_)
```

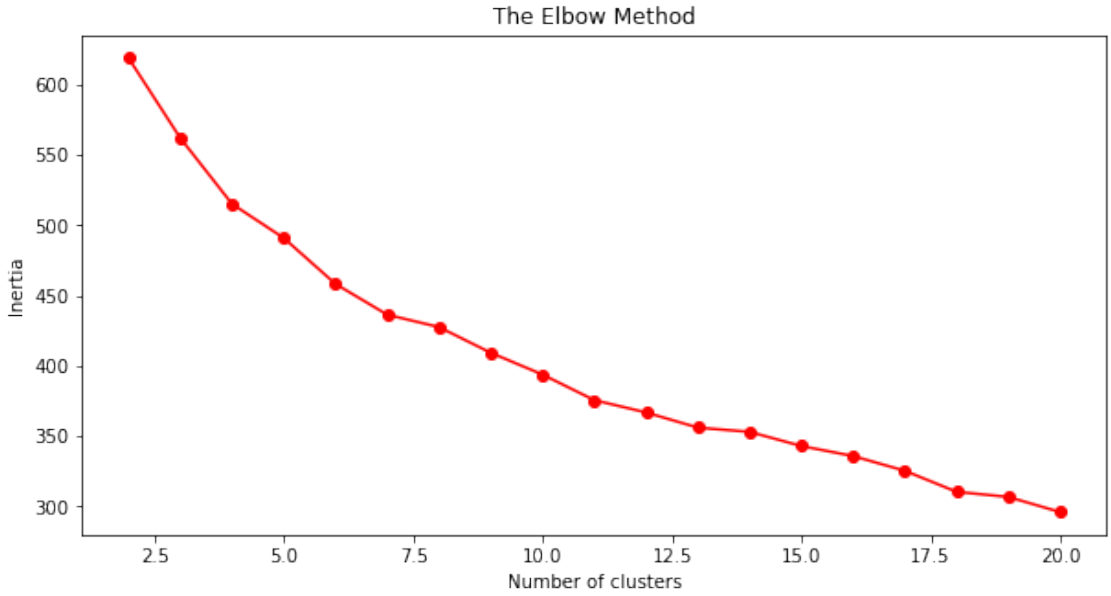
```
491.06656636125916
```

3.6.2 [10 pts] Find the optimal cluster size using the elbow method.

Use the elbow method to find the best cluster size or range of best cluster sizes for the train data. Check the cluster sizes from 2 to 20. Make sure to plot the Inertia and state where you think the elbow starts. Make sure to use random_state = 0.

[53]:
```
ks = list(range(2,21))
inertia = []
for k in ks:
    kmeans = KMeans(n_clusters = k, init = 'k-means++', random_state = 0)
```

```
    kmeans.fit(train)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(10,5))
plt.plot(ks, inertia,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



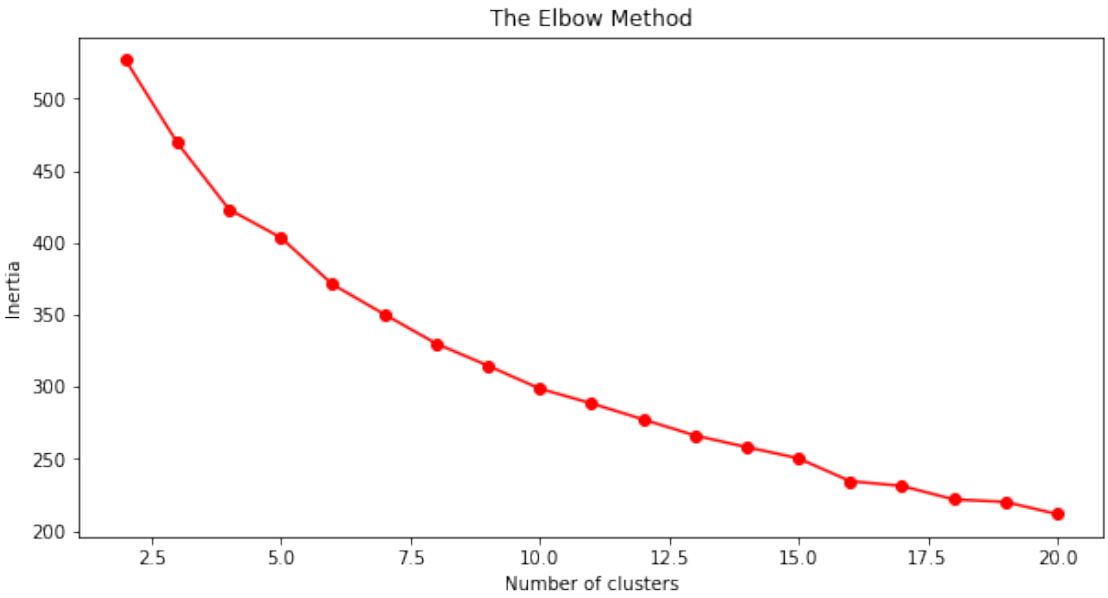
Elbow should start at **k=6** because after k=6, inertia starts to drop consistently.

3.6.3 [5 pts] Find the optimal cluster size for the train_pca data

Repeat the same experiment but use train_pca instead of train.

[54]:
```
ks = list(range(2,21))
inertia = []
for k in ks:
    kmeans = KMeans(n_clusters = k, init = 'k-means++', random_state = 0)
    kmeans.fit(train_pca)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(10,5))
plt.plot(ks, inertia,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
plt.show()
```



Notice that the inertia is much smaller for every cluster size when using PCA features. Why do you think this is happening? Hint: Think about what Inertia is calculating and consider the number of features that PCA outputs.

Response: After PCA transformation, when using kmean to calculate the distance, the distance between each feature already be closer compare with original data.

4 (100 pts) Putting it all together

Through all the homeworks and projects, you have learned how to apply many different models to perform a supervised learning task. We are now asking you to take everything that you learned to create a model that can predict whether a hotel reservation will be canceled or not.

Context

Hotels see millions of people every year and always wants to keep rooms occupied and payed for. Cancellations make the business lose money since it may make it difficult to reserve to another customer on such short notice. As such, it is useful for a hotel to know whether a reservation is likely to cancel or not. The following dataset will provide a variety of information about a booking that you will use to predict whether that booking will cancel or not.

Property Management System - PMS

Attribute Information

(C) is for Categorical

(D) is for Numeric

1) is_canceled (C) : Value indicating if the booking was canceled (1) or not (0).
2) hotel (C) : The datasets contains the booking information of two hotel. One of the hotels is a resort hotel and the other is a city hotel.
3) arrival_date_month (C): Month of arrival date with 12 categories: "January" to "December"
4) stays_in_weekend_nights (N): Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
5) stays_in_week_nights (N): Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel BO and BL/Calculated by counting the number of week nights
6) adults (N): Number of adults
7) children (N): Number of children
8) babies (N): Number of babies
9) meal (C): Type of meal
10) country (C): Country of origin.
11) previous_cancellations (N): Number of previous bookings that were canceled by the customer prior to the current booking
12) previous_bookings_not_canceled (N) : Number of previous bookings not canceled by the customer prior to the current booking
13) reserved_room_type (C): Code of room type reserved. Code is presented instead of designation for anonymity reasons
14) booking_changes (N) : Number of changes/amendments made to the booking from the moment the booking was entered on the PMS until the moment of check-in or cancellation
15) deposit_type (C) : No Deposit – no deposit was made; Non Refund – a deposit was made in the value of the total stay cost; Refundable – a deposit was made with a value under the total cost of stay
16) days_in_waiting_list (N): Number of days the booking was in the waiting list before it was confirmed to the customer
17) customer_type (C): Group – when the booking is associated to a group; Transient – when the booking is not part of a group or contract, and is not associated to other transient booking; Transient-party – when the booking is transient, but is associated to at least other transient booking
18) adr (N): Average Daily Rate (Calculated by dividing the sum of all lodging transactions by the total number of staying nights)
19) required_car_parking_spaces (N): Number of car parking spaces required by the customer
20) total_of_special_requests (N): Number of special requests made by the customer (e.g. twin bed or high floor)
21) name (C): Name of the Guest (Not Real)
22) email (C): Email (Not Real)
23) phone-number (C): Phone number (not real)

This dataset is quite large with 86989 samples. This makes it difficult to just brute force running a lot of models. As such, you have to be thoughtful when designing your models.

The file name for the training data is "hotel_booking.csv".

Challenge

This project is about being able to predict whether a reservation is likely to cancel based on the input parameters available to us. We will ask you to perform some specific instructions to lead you in the right direction but you are given free reign on which models to use and the preprocessing steps you make. We will ask you to **write out a description of what models you choose and why you choose them**.

4.1 (50 pts) Preprocessing

Preprocessing: For the dataset, the following are mandatory pre-processing steps for your data:

- Use One-Hot Encoding on all categorical features (specify whether you keep the extra feature or not for features with multiple values)
- Determine which fields need to be dropped
- Handle missing values (Specify your strategy)
- Rescale the real valued features using any strategy you choose (StandardScaler, MinMaxScaler, Normalizer, etc)
- Augment at least one feature
- Implement a train-test split with 20% of the data going to the test data. Make sure that the test and train data are balanced in terms of the desired class.

After writing your preprocessing code, write out a description of what you did for each step and provide a justification for your choices. All descriptions should be written in the markdown cells of the jupyter notebook. Make sure your writing is clear and professional.

We highly recommend reading through the scikit-learn documentation to make this part easier.

[55]:
```
df = pd.read_csv("datasets/hotel_booking.csv")
df.head()
```

[55]:

	is_canceled	hotel	lead_time	arrival_date_month	\
0	0	Resort Hotel	4	February	
1	1	City Hotel	172	June	
2	0	City Hotel	4	November	
3	1	City Hotel	68	September	
4	1	City Hotel	149	July	

	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	\
0	1	2	2	0.0	0	
1	0	2	1	0.0	0	
2	2	1	1	0.0	0	
3	0	2	2	0.0	0	
4	2	4	3	0.0	0	

	meal	…	booking_changes	deposit_type	days_in_waiting_list	\
0	FB	…	0	No Deposit	0	
1	BB	…	0	No Deposit	0	
2	BB	…	0	No Deposit	0	
3	HB	…	0	No Deposit	0	
4	BB	…	0	No Deposit	0	

```
     customer_type    adr required_car_parking_spaces  \
0        Transient   75.0                            0
1  Transient-Party   95.0                            0
2        Transient   65.0                            0
3  Transient-Party    0.0                            0
4        Transient  167.7                            0

   total_of_special_requests             name                         email  \
0                          1      Linda Moore                LMoore@att.com
1                          0   Madison Greene  Greene_Madison56@verizon.com
2                          0  Alicia Richards   Richards.Alicia@comcast.net
3                          0    Gregory Smith      GregorySmith@outlook.com
4                          0  Rachel Martinez          Rachel.M@outlook.com

   phone-number
0  217-602-3707
1  791-162-2669
2  442-385-2754
3  670-687-2703
4  692-194-2274

[5 rows x 24 columns]
```

[56]:
```
df=df.drop(['name', 'email', 'phone-number'], axis=1)
```

There is no need for name, email, phone-number features because they do not help a lot in prediction. Personal information are unique and will not provide correlation between whether cancel hotel or not.

[57]:
```
print(df.isnull().sum())
print(len(df))
df=df.dropna()
```

```
is_canceled                     0
hotel                           0
lead_time                       0
arrival_date_month              0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        3
babies                          0
meal                            0
country                         0
previous_cancellations          0
previous_bookings_not_canceled  0
reserved_room_type              0
```

```
booking_changes                 0
deposit_type                    0
days_in_waiting_list            0
customer_type                   0
adr                             0
required_car_parking_spaces     0
total_of_special_requests       0
dtype: int64
78290
```

[57]:

	is_canceled	hotel	lead_time	arrival_date_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	…	previous_cancellations	previous_bookings_not_canceled	reserved_room_type	booking_changes	deposit_type	days_in_waiting_list	customer_type	adr	required_car_parking_spaces	total_of_special_requests
0	0	Resort Hotel	4	February	1	2	2	0.0	0	FB	…	0	0	A	0	No Deposit	0	Transient	75.0	0	1
1	1	City Hotel	172	June	0	2	1	0.0	0	BB	…	0	0	A	0	No Deposit	0	Transient-Party	95.0	0	
2	0	City Hotel	4	November	2	1	1	0.0	0	BB	…	0	0	A	0	No Deposit	0	Transient	65.0	0	
3	1	City Hotel	68	September	0	2	2	0.0	0	HB	…	0	0	A	0	No Deposit	0	Transient-Party	0.0	0	
4	1	City Hotel	149	July	2	4	3	0.0	0	BB	…	0	0	D	0	No Deposit	0	Transient	167.7	0	

```
1                           0
2                           0
3                           0
4                           0

[5 rows x 21 columns]
```

As we have big enough observations, so we can drop three missing value which will not impact the performance a lot.

[58]:
```
df.describe(include='object')
```

[58]:
```
              hotel arrival_date_month   meal country reserved_room_type  \
count         78287              78287  78287   78287              78287
unique            2                 12      5       5                 10
top     City Hotel             August     BB     PRT                  A
freq          50200               9188  60443   43644              58568

       deposit_type customer_type
count         78287         78287
unique            3             4
top      No Deposit     Transient
freq          65161         57284
```

After check description of categorical variables, we could encode month of arrival date to seasons so that we can have smaller dataset after One-Hot Encoding.

[59]:
```
season_month_north = {
    'December':'Winter', 'January':'Winter', 'February':'Winter',
    'March':'Spring', 'April':'Spring', 'May':'Spring',
    'June':'Summer', 'July':'Summer', 'August':'Summer',
    'September':'Autumn', 'October':'Autumn', 'November':'Autumn'}
```

[60]:
```
df=df.replace({'arrival_date_month': season_month_north})
df.head()
```

[60]:
```
   is_canceled         hotel  lead_time arrival_date_month  \
0            0  Resort Hotel          4             Winter
1            1    City Hotel        172             Summer
2            0    City Hotel          4             Autumn
3            1    City Hotel         68             Autumn
4            1    City Hotel        149             Summer

   stays_in_weekend_nights  stays_in_week_nights  adults  children  babies  \
0                        1                     2       2       0.0       0
1                        0                     2       1       0.0       0
2                        2                     1       1       0.0       0
3                        0                     2       2       0.0       0
```

```
4                       2                     4       3       0.0       0

  meal  … previous_cancellations  previous_bookings_not_canceled  \
0   FB  …                       0                               0
1   BB  …                       0                               0
2   BB  …                       0                               0
3   HB  …                       0                               0
4   BB  …                       0                               0

   reserved_room_type booking_changes  deposit_type days_in_waiting_list  \
0                   A               0    No Deposit                    0
1                   A               0    No Deposit                    0
2                   A               0    No Deposit                    0
3                   A               0    No Deposit                    0
4                   D               0    No Deposit                    0

     customer_type    adr  required_car_parking_spaces  \
0        Transient   75.0                            0
1  Transient-Party   95.0                            0
2        Transient   65.0                            0
3  Transient-Party    0.0                            0
4        Transient  167.7                            0

   total_of_special_requests
0                          1
1                          0
2                          0
3                          0
4                          0

[5 rows x 21 columns]
```

[61]:
```
correlations = df.corr()
correlations
```

[61]:
```
                                is_canceled  lead_time  \
is_canceled                        1.000000   0.326865
lead_time                          0.326865   1.000000
stays_in_weekend_nights           -0.012625   0.080942
stays_in_week_nights               0.019326   0.162367
adults                             0.057672   0.123466
children                          -0.018874  -0.053794
babies                            -0.035104  -0.023262
previous_cancellations             0.119779   0.088407
previous_bookings_not_canceled    -0.067608  -0.080490
booking_changes                   -0.167776  -0.017826
days_in_waiting_list               0.070788   0.175964
```

```
adr                              0.035499  -0.056880
required_car_parking_spaces     -0.216897  -0.129994
total_of_special_requests       -0.249011  -0.125441

                                stays_in_weekend_nights  stays_in_week_nights  \
is_canceled                                   -0.012625              0.019326
lead_time                                      0.080942              0.162367
stays_in_weekend_nights                        1.000000              0.502492
stays_in_week_nights                           0.502492              1.000000
adults                                         0.105297              0.106137
children                                       0.046651              0.047172
babies                                         0.022213              0.024342
previous_cancellations                        -0.018852             -0.024699
previous_bookings_not_canceled                -0.042012             -0.051351
booking_changes                                0.062640              0.094069
days_in_waiting_list                          -0.047322             -0.001174
adr                                            0.065647              0.090391
required_car_parking_spaces                   -0.012911             -0.019037
total_of_special_requests                      0.085814              0.077960

                                  adults  children    babies  \
is_canceled                     0.057672 -0.018874 -0.035104
lead_time                       0.123466 -0.053794 -0.023262
stays_in_weekend_nights         0.105297  0.046651  0.022213
stays_in_week_nights            0.106137  0.047172  0.024342
adults                          1.000000  0.039043  0.019844
children                        0.039043  1.000000  0.032727
babies                          0.019844  0.032727  1.000000
previous_cancellations         -0.006569 -0.026697 -0.008918
previous_bookings_not_canceled -0.111317 -0.020634 -0.007259
booking_changes                -0.051013  0.059031  0.088605
days_in_waiting_list           -0.004776 -0.034557 -0.012031
adr                             0.200782  0.302619  0.039143
required_car_parking_spaces     0.009859  0.062359  0.040160
total_of_special_requests       0.113022  0.101342  0.105878

                                previous_cancellations  \
is_canceled                                   0.119779
lead_time                                     0.088407
stays_in_weekend_nights                      -0.018852
stays_in_week_nights                         -0.024699
adults                                       -0.006569
children                                     -0.026697
babies                                       -0.008918
previous_cancellations                        1.000000
previous_bookings_not_canceled                0.170992
booking_changes                              -0.030638
```

```
days_in_waiting_list                          0.005041
adr                                          -0.060357
required_car_parking_spaces                  -0.022866
total_of_special_requests                    -0.047774

                                 previous_bookings_not_canceled  \
is_canceled                                           -0.067608
lead_time                                             -0.080490
stays_in_weekend_nights                               -0.042012
stays_in_week_nights                                  -0.051351
adults                                                -0.111317
children                                              -0.020634
babies                                                -0.007259
previous_cancellations                                 0.170992
previous_bookings_not_canceled                         1.000000
booking_changes                                        0.017298
days_in_waiting_list                                  -0.012047
adr                                                   -0.070682
required_car_parking_spaces                            0.051663
total_of_special_requests                              0.052482

                                 booking_changes  days_in_waiting_list  \
is_canceled                            -0.167776              0.070788
lead_time                              -0.017826              0.175964
stays_in_weekend_nights                 0.062640             -0.047322
stays_in_week_nights                    0.094069             -0.001174
adults                                 -0.051013             -0.004776
children                                0.059031             -0.034557
babies                                  0.088605             -0.012031
previous_cancellations                 -0.030638              0.005041
previous_bookings_not_canceled          0.017298             -0.012047
booking_changes                         1.000000             -0.017132
days_in_waiting_list                   -0.017132              1.000000
adr                                     0.028307             -0.033147
required_car_parking_spaces             0.083790             -0.036686
total_of_special_requests               0.065804             -0.087617

                                       adr  required_car_parking_spaces  \
is_canceled                       0.035499                    -0.216897
lead_time                        -0.056880                    -0.129994
stays_in_weekend_nights           0.065647                    -0.012911
stays_in_week_nights              0.090391                    -0.019037
adults                            0.200782                     0.009859
children                          0.302619                     0.062359
babies                            0.039143                     0.040160
previous_cancellations           -0.060357                    -0.022866
previous_bookings_not_canceled   -0.070682                     0.051663
```

```
booking_changes                  0.028307                    0.083790
days_in_waiting_list            -0.033147                   -0.036686
adr                              1.000000                    0.069751
required_car_parking_spaces      0.069751                    1.000000
total_of_special_requests        0.179531                    0.103825

                                 total_of_special_requests
is_canceled                                      -0.249011
lead_time                                        -0.125441
stays_in_weekend_nights                           0.085814
stays_in_week_nights                              0.077960
adults                                            0.113022
children                                          0.101342
babies                                            0.105878
previous_cancellations                           -0.047774
previous_bookings_not_canceled                    0.052482
booking_changes                                   0.065804
days_in_waiting_list                             -0.087617
adr                                               0.179531
required_car_parking_spaces                       0.103825
total_of_special_requests                         1.000000
```

After check the correlation table, we should drop "adr" because it has lowest correlation with response variable.

[62]:
```
df=df.drop(['adr'], axis=1)
```

Augment Features based on variables we have and transform:

[63]:
```
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin

######Processing Real Valued Features

class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    total night = stays_in_weekend_nights+stays_in_week_nights
    total booking = previous_cancellations+previous_bookings_not_canceled
    total people = adults+children+babies
    '''
    def __init__(self):
        self.week = 'stays_in_week_nights'
        self.weekend = 'stays_in_weekend_nights'
```

```
        self.cancel='previous_cancellations'
        self.nocancel='previous_bookings_not_canceled'
        self.adult='adults'
        self.child='children'
        self.baby='babies'
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        X['total_night']=X[self.week]+X[self.weekend]
        X['total_book']=X[self.cancel]+X[self.nocancel]
        X['total_people']=X[self.adult]+X[self.child]+X[self.baby]
        return X
```

[64]:
```
#Split target and data
y = df["is_canceled"]
x = df.drop(["is_canceled"],axis = 1)

#Train test split
train_raw, test_raw, target, target_test = train_test_split(x,y, test_size=0.2,␣
 ↪stratify= y, random_state=0)
```

[65]:
```
#Pipiline for real valued features
num_pipeline = Pipeline([
        ('attribs_adder', AugmentFeatures()), #
        ('std_scaler', StandardScaler()),
    ])

#Full Pipeline

#Splits names into numerical and categorical features
categorical_features = ['hotel','arrival_date_month','meal','country',
                        'reserved_room_type', 'deposit_type', 'customer_type']
numerical_features =␣
 ↪['lead_time','stays_in_weekend_nights','stays_in_week_nights','adults',
                    ␣
 ↪'children','babies','previous_cancellations','previous_bookings_not_canceled',
                    ␣
 ↪'booking_changes','days_in_waiting_list','required_car_parking_spaces',
                      'total_of_special_requests']

#Applies different transformations on numerical columns vs categorial columns
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(categories='auto'), categorical_features),
    ])

train = full_pipeline.fit_transform(train_raw)
```

```
test = full_pipeline.transform(test_raw)
```

4.2 (50 pts) Try out a few models

Now that you have pre-processed your data, you are ready to try out different models.

For this part of the project, we want you to experiment with all the different models demonstrated in the course to determine which one performs best on the dataset.

You must perform classification using at least 3 of the following models: - Logistic Regression - K-nearest neighbors - SVM - Decision Tree - Multi-Layer Perceptron

Due to the size of the dataset, be careful which models you use and look at their documentation to see how you should tackle this size issue for each model.

For full credit, you must perform some hyperparameter optimization on your models of choice. You may find the following scikit-learn library on hyperparameter optimization useful.

For each model chosen, write a description of which models were chosen, which parameters you optimized, and which parameters you choose for your best model. While the previous part of the project asked you to pre-process the data in a specific manner, you may alter pre-processing step as you wish to adjust for your chosen classification models.

[91]:
```
from sklearn.linear_model import LogisticRegression
logparameters = [
    {"penalty": ['l1', 'l2'],
    "C": [0.01,1,100]}
]

k = 3
kf = KFold(n_splits=k, random_state=None)

log_reg = LogisticRegression(penalty = "none",max_iter = 1000, solver = "lbfgs")
loggrid = GridSearchCV(log_reg , logparameters, cv = kf, scoring = "accuracy")
loggrid.fit(train,target)
loggrid.best_params_
loggrid.best_score_
```

```
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
9 fits failed out of a total of 18.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
9 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/mac/opt/anaconda3/lib/python3.9/site-
```

```
packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py", line 447, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/model_selection/_search.py:969: UserWarning: One or more of the
test scores are non-finite: [       nan 0.80721387        nan 0.80724579
nan 0.80732563]
  warnings.warn(
```

[91]:
```
0.8073256289651152
```

[89]:
```
from sklearn.neighbors import KNeighborsClassifier
knnparameters = [
    {"n_neighbors" :[1,3,5,7],
    "metric":["euclidean","manhattan"]}
]

knn = KNeighborsClassifier()
knngrid = GridSearchCV(knn , knnparameters, cv = kf, scoring = "accuracy")
knngrid.fit(train,target)
knngrid.best_params_
knngrid.best_score_
```

[89]:
```
0.830334124369707
```

[90]:
```
dtparameters = [
    {"max_depth":[1,2,3,4,5,10,15],
    "min_samples_split":[2,4,6,8],
    "criterion": ["gini", "entropy"]}
]

dt = DecisionTreeClassifier()
dtgrid = GridSearchCV(dt , dtparameters, cv = kf, scoring = "accuracy")
dtgrid.fit(train,target)
dtgrid.best_params_
dtgrid.best_score_
```

[90]:
```
0.834102371744975
```

[92]:
```
dtgrid.best_params_
```

[92]:
```
{'criterion': 'gini', 'max_depth': 15, 'min_samples_split': 6}
```

It shows that decision tree has the best accuracy with {'criterion': 'gini', 'max_depth': 15, 'min_samples_split': 6}

[84]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
clf1 = LogisticRegression()
clf2 = KNeighborsClassifier()
clf3 = DecisionTreeClassifier()
params = [{'clf__penalty': ['l1', 'l2'],
           'clf__C': [0.01, 1, 100],
          'clf': [clf1]},
          {'clf__n_neighbors':[1,3,5,7],
           'clf__metric':["euclidean","manhattan"],
           'clf': [clf2]},
          {'lf__max_depth':[1,2,3,4,5,10,15],
           'clf__min_samples_split':[2,4,6,8],
           'clf__criterion': ["gini", "entropy"],
           'clf': [clf3]}]
kf = KFold(n_splits=3)
pipeline = Pipeline([('clf', clf1)])
```

It's too large to run following. So, I seperate them.

[85]:
```
#grid = GridSearchCV(pipeline,params,cv=kf, scoring ="accuracy")
#grid.fit(train,target)
#grid.best_params_
```

```
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/mac/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

KeyboardInterrupt
```

4.3 Extra Credit

We have provided an extra test dataset named "hotel_booking_test.csv" that does not have the target labels. Classify the samples in the dataset with your best model and write them into a csv file. Submit your csv file to our Kaggle contest. The website will specify your classification accuracy on the test set. We will award a bonus point for the project for every percentage point over 75% that you get on your kaggle test accuracy.

To get the bonus points, you must also write out a summary of the model that you submit including any changes you made to the pre-processing steps. The summary must be written in a markdown cell of the jupyter notebook. Note that you should not change earlier parts of the project to complete the extra credit.

Kaggle Submission Instruction Submit a two column csv where the first column is named "ID" and is the row number. The second column is named "target" and is the classification for each sample. Make sure that the sample order is preserved.

[94]:
```
test_hotel = pd.read_csv("datasets/hotel_booking_test.csv")
test_hotel["children"][test_hotel["children"].isna()] = 0
test_hotel=test_hotel.replace({'arrival_date_month': season_month_north})
output = dtgrid.predict(full_pipeline.transform(test_hotel))
```

```
/var/folders/98/vwplyq_x2ddf8y2lpl0r0h7w0000gn/T/ipykernel_23716/1492027790.py:2
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_hotel["children"][test_hotel["children"].isna()] = 0
```

[95]:
```
kaggle = pd.DataFrame(output,columns=["target"])
kaggle.index.name = "ID"
kaggle.to_csv("kaggle.csv")
```

Summary:

Data pre-processing

I removed the name, email, and phone number features first because they are fake and do not provide much help in prediction. Personal information is unique and does not correlate with whether the hotel will be canceled or not.

Secondly, I checked for missing values and found three instances of "NA". Since we have a sufficiently large number of observations, I decided to drop these missing values as they will not significantly impact the performance. During prediction, I replaced the missing values with 0.

Thirdly, I examined the descriptions of the categorical variables we have. After reviewing the descriptions, I converted the month of arrival date into seasons to reduce the size of the dataset after applying One-Hot Encoding. I followed the same processing steps for the test data during prediction.

Next, I examined the correlation matrix of the numerical variables. After analyzing the correlation table, I decided to drop the "adr" variable because it has the lowest correlation with the response variable.

Lastly, I augmented three features based on the available variables and performed transformations on both the numerical and categorical variables. I applied these transformations to both the training data and our own testing data.

Model

I primarily experimented with three models: logistic regression, KNN, and decision tree. Initially, I planned to use a pipeline to automatically search for the best parameters, but due to the large size of the data, it was not feasible. Therefore, I ran each model separately.

For logistic regression, I configured the parameters to determine the type of penalty to use and its magnitude. In the case of KNN, I specified the number of neighbors to consider and the distance metric to be employed. Regarding the decision tree, I set the maximum depth of the tree, the minimum number of samples required to split a node, and the criterion for measuring information loss or entropy.

After comparing the best scores achieved by each model with their respective optimal parameters, I found that the decision tree attained the highest score of 83.4%. This was achieved using the gini index measurement, a maximum depth of 15, and a minimum sample split of 6.

At the end, after I upload prediction csv to Kaggle, I got 84% accuracy.

[ ]: