Feature_Selection_in_Python

January 29, 2024

0.1 Feature Selection in Machine Learning: Python code

Lets learn about feature selection in machine learning in detail with proper implementation in python.

The predictive accuracy of a machine learning model largely depends on its fed data. Hence, cleaning the data and selecting relevant features are the foremost important steps in model designing.

As a Data Scientist, one of the core practices you will be following is selecting value-adding features from a given data. By value-adding, we mean those features which will play a role in ensuring optimum performance of your machine learning model.

You must know that a model trains better on sizeable training data. While this holds for the number of records (rows) in a dataset, a higher number of features (columns) are often undesirable.

This article will discuss the concept of feature selection and the different techniques to select the best features for training a robust ML model. But before reading this article, I suggest you learn the basics of this article by reading Feature selection techniques. Beginners guide.

We will be covering the following section 1. What is Feature Selection?

2. Why Feature Selection?
3. Methods for Feature Selection

   Filter Methods Wrapper Methods Embedded Methods
4. Performing Feature Selection Using Python
5. Endnotes

0.1.1 What is Feature Selection?

Feature Selection is a significant step in data pre-processing. It is also one of the main techniques in dimensionality reduction. Higher-dimensional data contains many redundant features that may negatively impact the performance of your model. Hence, it is important to identify 'principal' features from a dataset and filter out the irrelevant or unimportant features that will not contribute much to your target/prediction variable.

0.1.2 Why Feature Selection?

1. Reduce model complexity

Data with fewer dimensions is computationally inexpensive and less complex. Also, the computational time reduces significantly as well.

2. Eliminate noise

Unrelated features add to the noise in data. Noisy data wreaks havoc on the entire ML pipeline. So, through feature selection, we minimize redundancy and maximize relevance to the target variable.

3. Improve model performance

As stated above, feature selection helps the model learn better. Properly trained models are more generalized as they alleviate the problem of overfitting.

0.1.3 Methods for Feature Selection

There are various methods for performing feature selection on a dataset. We will discuss here the most important supervised feature selection methods that make use of output class labels. These methods use the target variable to identify relevant features that improve model accuracy.

0.1.4 Filter Methods

A feature can be regarded as irrelevant and discarded if it is conditionally independent of the class labels.

Filter methods are generally used as a pre-processing step. These methods filter and select a subset of the data that contains only the relevant features.

All features are ranked from best to worst based on the intrinsic properties of the data, such as correlation to the target variable, etc.

Different filter methods, including the Chi-Square Test, ANOVA Test, Linear Discriminant Analysis (LDA), etc., use other criteria to measure the relevance of features.

These methods are not dependent on the learning algorithm.

0.1.5 Wrapper Methods

Simple methods have the same objective as filter methods but use an ML algorithm as their evaluation criterion. Data is divided into a feature subset that is fed to the learning algorithm. Based on how the model performs, we decide whether to add or remove features from the subgroup and train the model again to increase its accuracy. These methods produce more accurate models than filtering but consume a lot of computational resources and are usually slow to run. Famous examples include Forward Selection, Backwards Elimination, Recursive Feature Elimination (RFE), etc. Note: Wrapper and Filter Methods are discrete processes, meaning the features are either kept or discarded. This can often cause high variance.

0.1.6 Embedded Methods

Embedded methods fuse the advantages of both filter and wrapper methods.

These methods perform feature selection and algorithm training in parallel. They are implemented by algorithms that have their integral feature selection process.

They are continuous methods and thus, don't suffer much from high variability.

Examples of these methods are RIDGE and LASSO regression, which have built-in functions to help reduce overfitting.

0.2 Performing Feature Selection Using Python

For demonstration, we are going to make use of the Breast Cancer dataset from Kaggle to try and predict if the tumor is cancerous or not by looking at the given features. While doing so, we will use different feature selection techniques to see how it affects the training time and overall accuracy of a Random Forest Classifier model.

Let's get started!

0.2.1 Dataset Description:

ID number

Diagnosis – Diagnosis of breast cancer (M = malignant, B = benign)

radius (mean of distances from the center to points on the perimeter)

texture (standard deviation of gray-scale values)

perimeter

area

smoothness (local variation in radius lengths)

compactness (perimeter^2 / area – 1.0)

concavity (severity of concave portions of the contour)

concave points (number of concave portions of the contour)

symmetry

fractal dimension ("coastline approximation" – 1)

The mean, standard error, and "worst" (mean of the three largest values) of the above features were computed for each image, resulting in 30 features.

All feature values are recoded with four significant digits.

Missing attribute values: none.

Target variable class distribution: 357 benign, 212 malignant.

0.2.2 Tasks to be performed:

1) Load the data
2) Get the list of features
3) Find correlation between features
4) Feature Selection with Correlation and Random Forest Classification

5) Recursive Feature Elimination (RFE) and Random Forest Classification

6) RFE with Cross-Validation and Random Forest Classification

7) Tree-Based Feature Selection and Random Forest Classification

0.2.3 Step 1 – Load the data

[4]:
```
#Import required libraries
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns

#Load the data
data = pd.read_csv('data.csv')
data.head()
```

[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	…	radius_worst	texture_worst	perimeter_worst	area_worst	\
0	…	25.38	17.33	184.60	2019.0	
1	…	24.99	23.41	158.80	1956.0	
2	…	23.57	25.53	152.50	1709.0	
3	…	14.91	26.50	98.87	567.7	
4	…	22.54	16.67	152.20	1575.0	

	smoothness_worst	compactness_worst	concavity_worst	concave_points_worst	\
0	0.1622	0.6656	0.7119	0.2654	
1	0.1238	0.1866	0.2416	0.1860	
2	0.1444	0.4245	0.4504	0.2430	
3	0.2098	0.8663	0.6869	0.2575	
4	0.1374	0.2050	0.4000	0.1625	

```
   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

From our dataset displayed above, we can remove a few irrelevant features right away:

Ø The target variable 'diagnosis' should be separated from the feature set.

Ø The 'id' column is unnecessary for classification.

Ø The 'Unnamed: 32' column includes NaN values so we do not need it.

0.2.4 Step 2 – Get the list of features

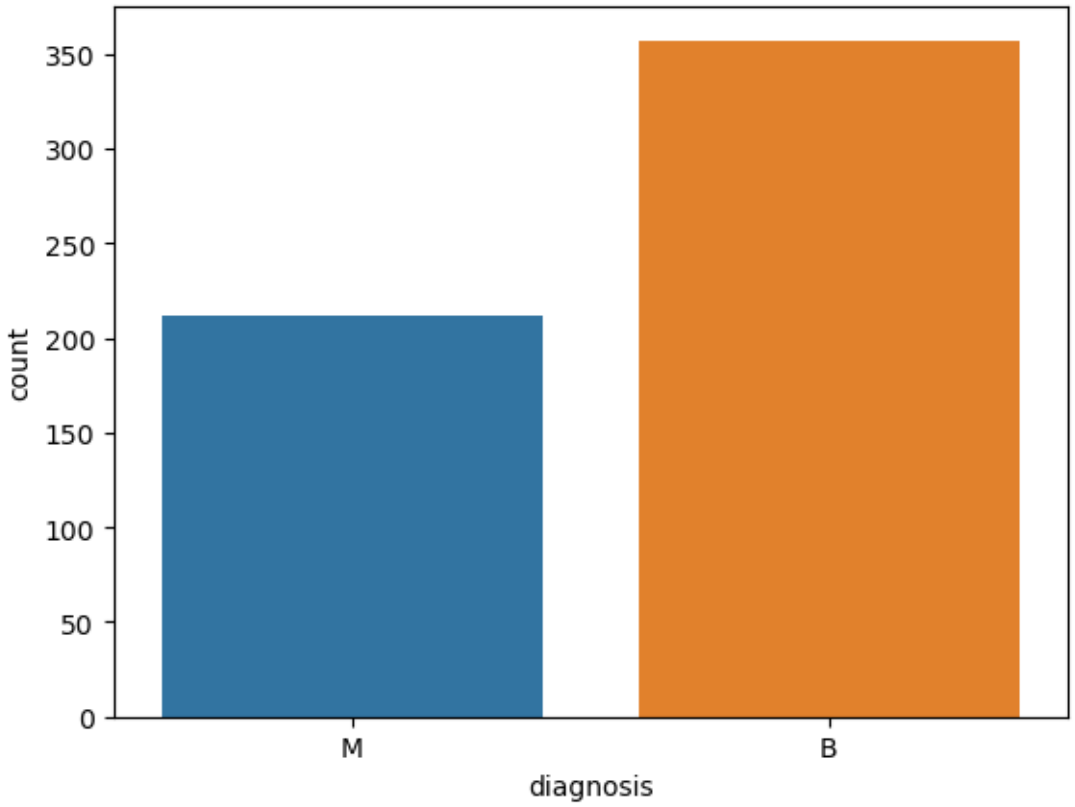
[7]:
```
#Get feature names
col = data.columns
print(col)

 #Target variable
y = data.diagnosis                          # M or B

#Features
list = ['id','diagnosis']
x = data.drop(list,axis = 1 )
x.head()

#Visualize the class labels
ax = sns.countplot(y,label="Count")       # M = 212, B = 357
B, M = y.value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

```
/home/kim/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
```

```
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

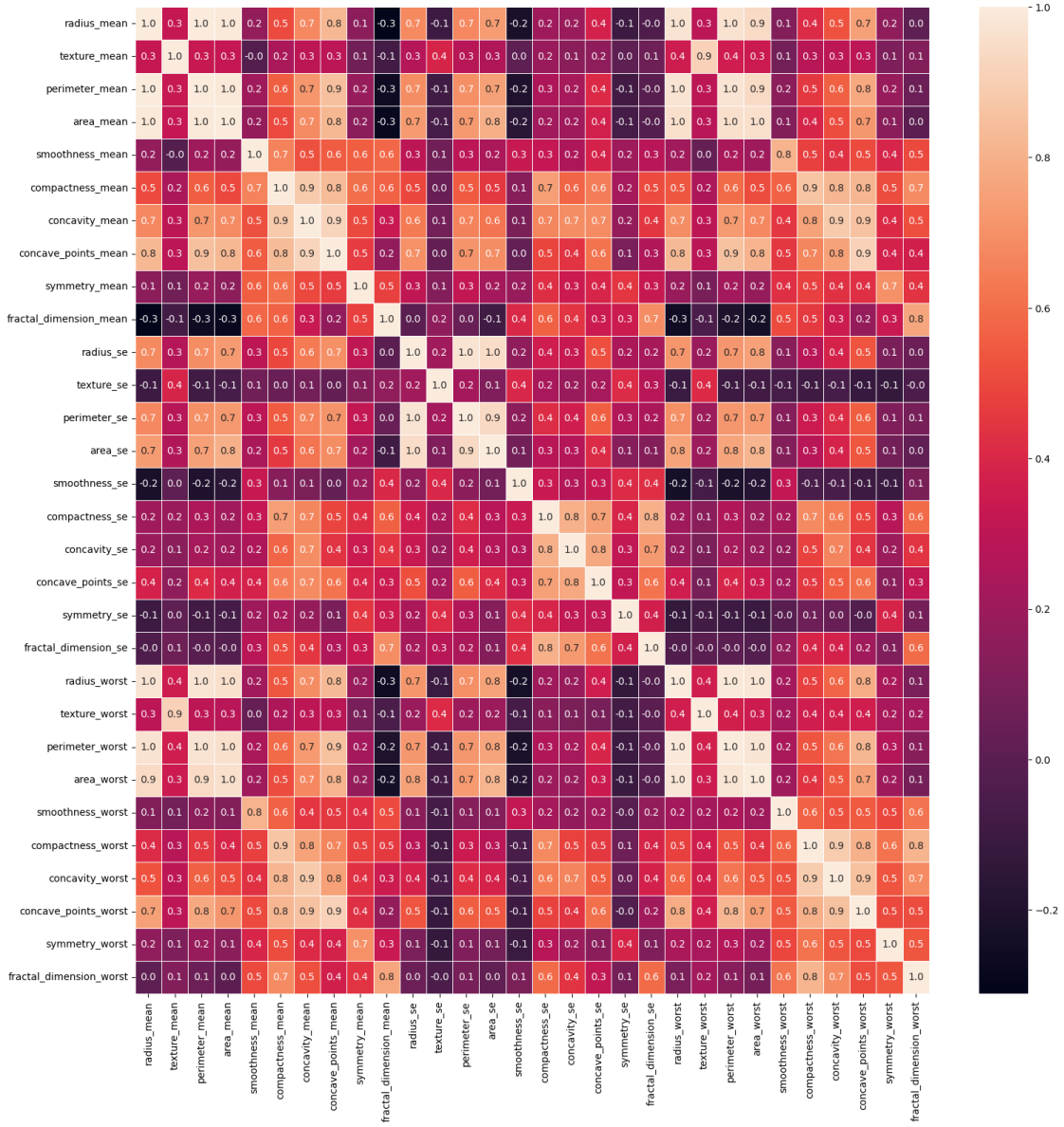
```
Number of Benign:  357
Number of Malignant :  212
```



0.2.5 Step 3 – Find correlation between features

[8]:
```
#Correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

[8]:
```
<AxesSubplot:>
```



0.2.6 Step 4 – Feature Selection with Correlation and Random Forest Classification

According to the heat map we created above, we can infer the following:

The features radius_mean, perimeter_mean, and area_mean are highly correlated with each other, so we will use only the area_mean feature.

Similarly, the features compactness_mean, concavity_mean, and concave points_mean are correlated with each other. Therefore, we will choose only concavity_mean.

The features radius_se, perimeter_se, and area_se are correlated, so we will use area_se.

The features radius_worst, perimeter_worst, and area_worst are correlated, so we will use area_worst.

The features compactness_worst, concavity_worst, and concave points_worst are correlated. So, we will use concavity_worst.

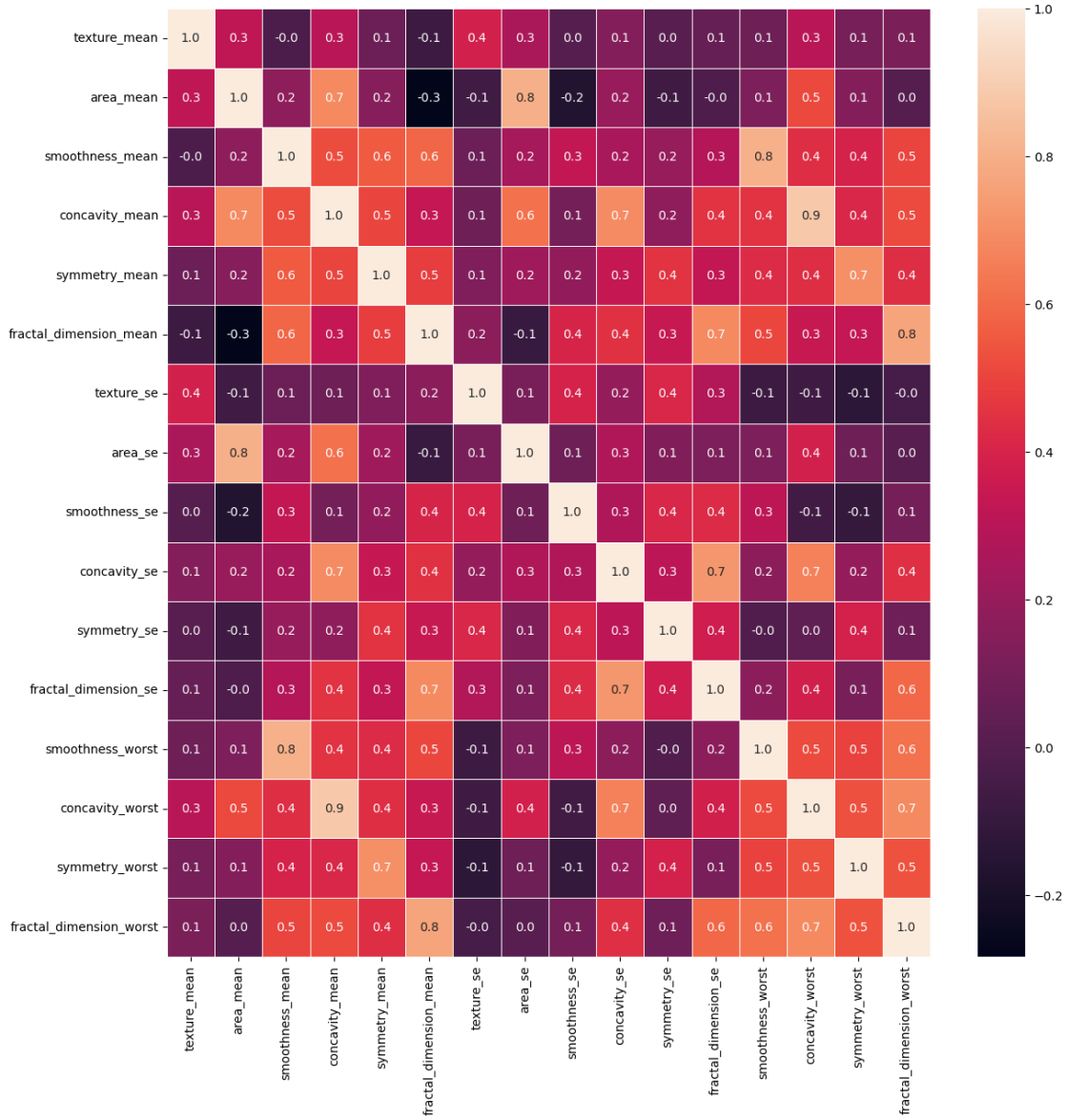
The features compactness_se, concavity_se, and concave points_se are correlated. So, we will use concavity_se.

The features texture_mean and texture_worst are correlated. So, we will use texture_mean.

The features area_worst and area_mean are correlated, we will use area_mean.

[12]:
```
drop_list1 =␣
 ↪['perimeter_mean','radius_mean','compactness_mean','concave_points_mean','radius_se','perim
 ↪'compactness_worst','concave_points_worst','compactness_se','concave_points_se','texture_wo

x_1 = x.drop(drop_list1,axis = 1 )
x_1.head()


# After dropping features, we will create a correlation matrix again as shown␣
 ↪below:
#Correlation heatmap
f,ax = plt.subplots(figsize=(14, 14))
sns.heatmap(x_1.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

[12]: <AxesSubplot:>



As it can be seen in the above heatmap, no more highly correlated features. Actually, there is a correlation value of 0.9 but let's see together what happens if we do not drop it.

So, we have chosen our features, but did we choose correctly? This will be answered by the performance of our Random Forest classifier.

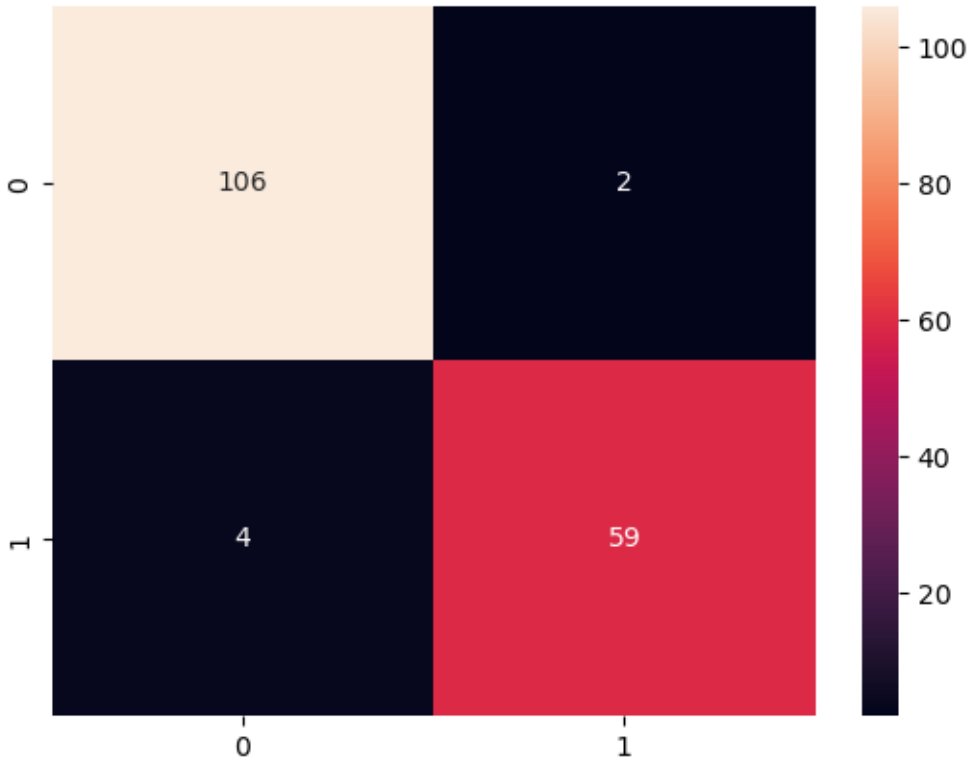
Let's split our data into 70% training and 30% testing set:

[14]:
```
from sklearn.model_selection import train_test_split

#Split the data
```

```
x_train,x_test,y_train,y_test = train_test_split(x_1,y,test_size=0.
↪3,random_state=42)

# Now let's train our classifier and find its accuarcy score:
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score,confusion_matrix
from sklearn.metrics import accuracy_score

# Build a random classifier with n_estimators=10 (default)
clf_rf=RandomForestClassifier(random_state=43)
clf_rf=clf_rf.fit(x_train,y_train)

ac = accuracy_score(y_test,clf_rf.predict(x_test))
print("Accuracy is: ",ac)

cm = confusion_matrix(y_test,clf_rf.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Accuracy is:  0.9649122807017544
```

[14]: <AxesSubplot:>



The accuracy is almost 96% and as can be seen in the confusion matrix, we do make a few wrong

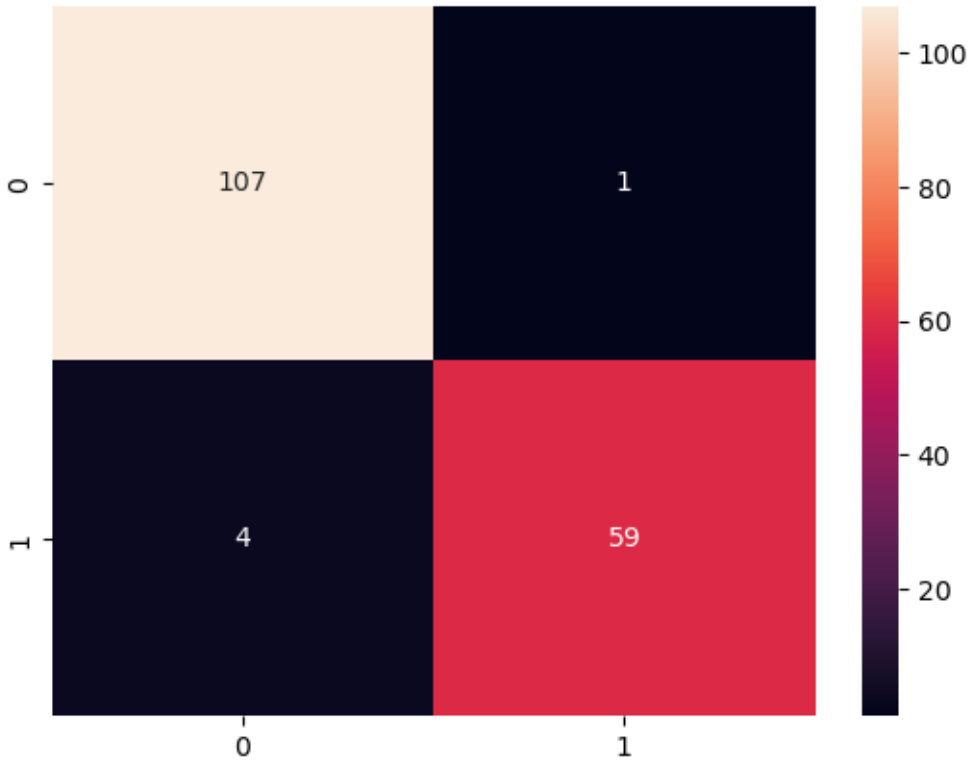
predictions. So, let's try other feature selection methods to see if we find more accurate results.

0.2.7 Step 5 – Recursive Feature Elimination (RFE) and Random Forest Classification

[23]:
```
from sklearn.feature_selection import RFE


# Create the RFE object
clf_rf_2 = RandomForestClassifier()
rfe = RFE(estimator=clf_rf_2,n_features_to_select=5,step=1)
rfe = rfe.fit(x_train,y_train)

print("Choosen best 5 feature by rfe: ",x_train.columns[rfe.support_])

# Let's calculate the accuracy score of the Random Forest classifier when we␣
 ↪use only the 5 select features:
#x_train_2 = feature_selection.transform(x_train)
#x_test_2 = feature_selection.transform(x_test)

#Random forest classifier with n_estimator=10 (default)
clf_rf_2 = RandomForestClassifier()
clr_rf_2 = clf_rf_2.fit(x_train,y_train)
ac_2 = accuracy_score(y_test,clf_rf_2.predict(x_test))
print('Accuracy is: ',ac_2)
cm_2 = confusion_matrix(y_test,clf_rf_2.predict(x_test))
sns.heatmap(cm_2,annot=True,fmt="d")
```

```
Choosen best 5 feature by rfe:  Index(['area_mean', 'concavity_mean', 'area_se',
'concavity_worst',
       'symmetry_worst'],
      dtype='object')
Accuracy is:  0.9707602339181286
```

[23]: <AxesSubplot:>



In this technique, we need to intuitively choose the number of features (k) we will use. Let's have the value of k=5. Now, which 5 features are to be used would be chosen by the RFE method:

The accuracy is almost 97% which is greater than the previous feature selection method we used.

However, this might also be because of our chosen value of k. Maybe if we use the best 2 or best 15 features, we might get much better accuracy. Therefore. Let's determine the optimal number of features we need:

0.2.8 Step 6 – RFE with Cross-Validation and Random Forest Classification

[20]:
```
from sklearn.feature_selection import RFE

#Create the RFE object
clf_rf_2 = RandomForestClassifier()
rfe = RFE(estimator=clf_rf_2, n_features_to_select=5, step=1)
rfe = rfe.fit(x_train, y_train)

print('Chosen best 5 feature by rfe:',x_train.columns[rfe.support_])

#Let's calculate the accuracy score of the Random Forest classifier when we use␣
 ↪only the 5 selected features:
x_train_2 = select_feature.transform(x_train)
```

```
x_test_2 = select_feature.transform(x_test)

#Random forest classifier with n_estimators=10 (default)
clf_rf_2 = RandomForestClassifier()
clr_rf_2 = clf_rf_2.fit(x_train_2,y_train)
ac_2 = accuracy_score(y_test,clf_rf_2.predict(x_test_2))
print('Accuracy is: ',ac_2)
cm_2 = confusion_matrix(y_test,clf_rf_2.predict(x_test_2))
sns.heatmap(cm_2,annot=True,fmt="d")
```

```
Chosen best 5 feature by rfe: Index(['area_mean', 'concavity_mean', 'area_se',
'concavity_worst',
       'symmetry_worst'],
      dtype='object')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_16049/1946359634.py in <module>
      9
     10 #Let's calculate the accuracy score of the Random Forest classifier whe
  ↪we use only the 5 selected features:
---> 11 x_train_2 = select_feature.transform(x_train)
     12 x_test_2 = select_feature.transform(x_test)
     13

NameError: name 'select_feature' is not defined
```

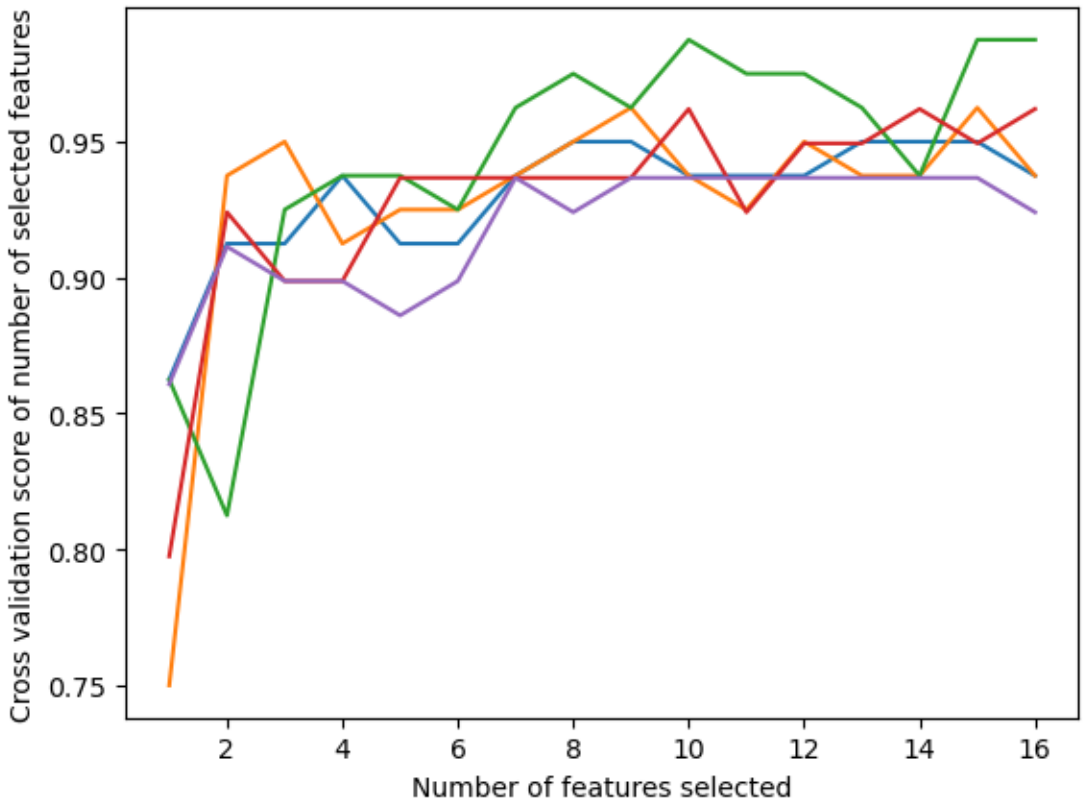
The accuracy score is proportional to the number of correct classifications:

[24]:
```
from sklearn.feature_selection import RFECV

clf_rf_3 = RandomForestClassifier()
rfecv = RFECV(estimator=clf_rf_3,step=1,cv=5,scoring='accuracy') #5-fold
  ↪cross-validation
rfecv = rfecv.fit(x_train,y_train)

print('Optimal number of features: ',rfecv.n_features_)
print('Best features :',x_train.columns[rfecv.support_])

# We now have a list of 15 best features to get the best accuracy score for our
  ↪model.
# Let's visualize the accuracy through a plot:

# Plot number of features VS cross-validation scores

plt.figure()
plt.xlabel("Number of features selected")
```

```
plt.ylabel("Cross validation score of number of selected features")
plt.plot(range(1,len(rfecv.grid_scores_)+1),rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features:  15
Best features : Index(['texture_mean', 'area_mean', 'smoothness_mean',
'concavity_mean',
       'symmetry_mean', 'fractal_dimension_mean', 'area_se', 'smoothness_se',
       'concavity_se', 'symmetry_se', 'fractal_dimension_se',
       'smoothness_worst', 'concavity_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

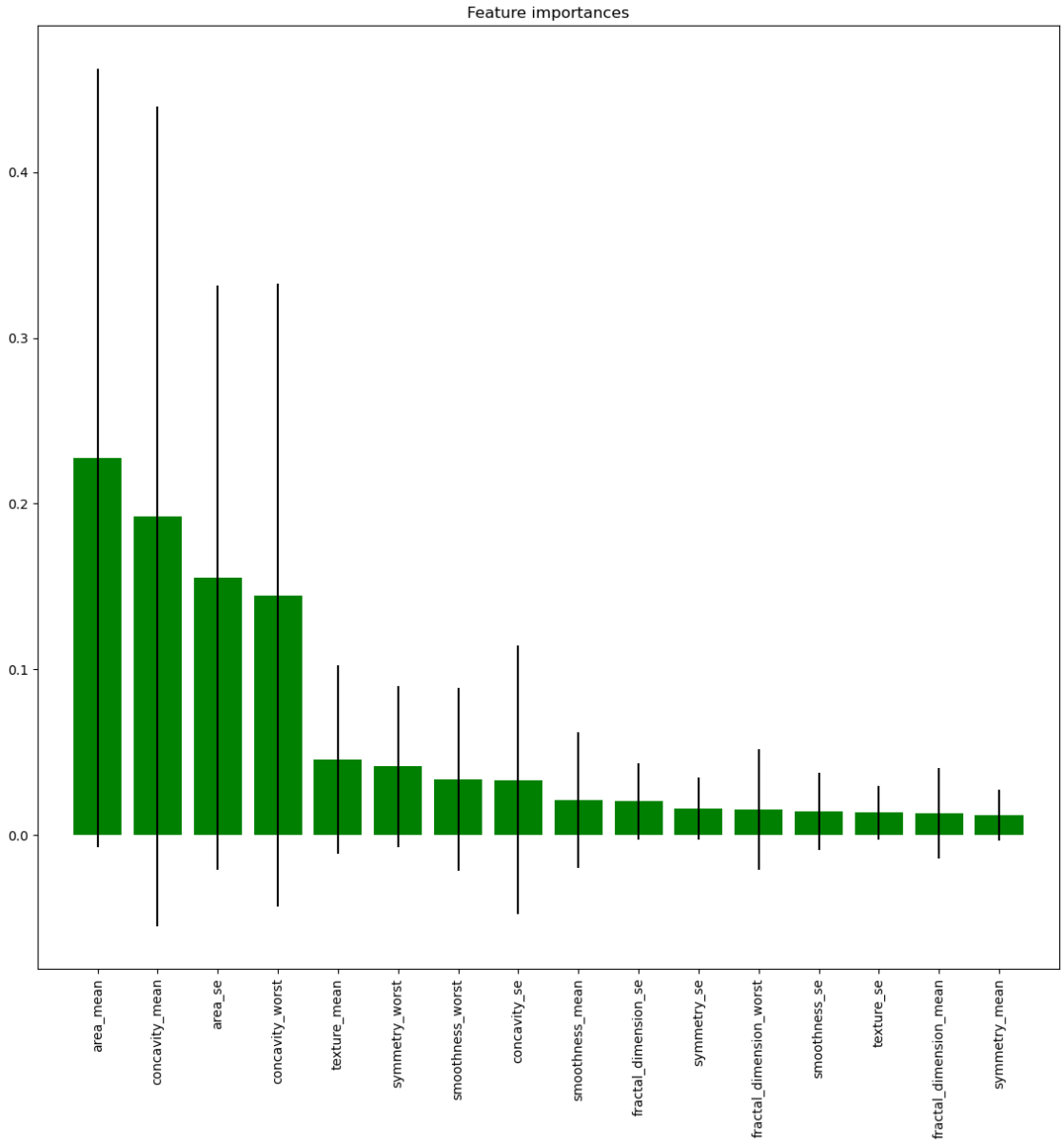
```
/home/kim/anaconda3/lib/python3.9/site-
packages/sklearn/utils/deprecation.py:103: FutureWarning: The `grid_scores_`
attribute is deprecated in version 1.0 in favor of `cv_results_` and will be
removed in version 1.2.
  warnings.warn(msg, category=FutureWarning)
```



0.2.9 Step 7 – Tree-Based Feature Selection in Random Forest Classification

[27]:
```
clf_rf_4 = RandomForestClassifier()
clr_rf_4 = clf_rf_4.fit(x_train,y_train)

importances = clr_rf_4.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf_rf.estimators_],axis=0)
indices = np.argsort(importances)[::-1]

#Print the feature ranking
print("Feature ranking:")

for f in range(x_train.shape[1]):
    print("%d. feature %d (%f)" % (f+1,indices[f],importances[indices[f]]))

# Plot the feature importances list:
plt.figure(1,figsize=(14,13))
plt.title("Feature importances")
plt.bar(range(x_train.
  ↪shape[1]),importances[indices],color="g",yerr=std[indices],align="center")
plt.xticks(range(x_train.shape[1]),x_train.columns[indices],rotation=90)
plt.xlim([-1,x_train.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.227667)
2. feature 3 (0.192364)
3. feature 7 (0.155491)
4. feature 13 (0.144740)
5. feature 0 (0.045427)
6. feature 14 (0.041383)
7. feature 12 (0.033757)
8. feature 9 (0.033253)
9. feature 2 (0.020941)
10. feature 11 (0.020524)
11. feature 10 (0.015968)
12. feature 15 (0.015280)
13. feature 8 (0.014219)
14. feature 6 (0.013596)
15. feature 5 (0.013286)
16. feature 4 (0.012104)
```



In the random forest classification method, there is a feature_importances attribute that defines the importance of the features. To use it, the features in the training data should not be correlated. Random Forest chooses randomly at each iteration; therefore, the sequence of feature importances list can change.

As you can see in the above plot, after the 6 best features, the importance of features decreases. Therefore, we can focus on these 6 features.

0.2.10 Endnotes

Finding the best features from a given data can help us extract valuable information and improve model performance in machine learning hence, feature selection is a must-do step during any model

building process. Artificial Intelligence & Machine Learning is an increasingly growing domain that has hugely impacted big businesses worldwide.

[ ]: