# 3.2.1 Aplicación de Naive Bayes

December 26, 2020

## 1 Aplicación de Naive Bayes

### 1.1 Conceptos previos

Es uno de los algoritmos más simples y poderosos para la clasificación basado en el Teorema de Bayes con una suposición de **independencia entre los predictores**. Naive Bayes es fácil de construir y particularmente útil para conjuntos de datos muy grandes.

Naive Bayes asume que el efecto de una característica particular en una clase es independiente de otras características. Por ejemplo, un solicitante de préstamo es deseable o no dependiendo de sus ingresos, historial de préstamos y transacciones anteriores, edad y ubicación. Incluso si estas características son interdependientes, estas características se consideran de forma independiente.

Esta suposición simplifica la computación, es por esa razón el nombre del algoritmo **Bayes ingenuo**. Formalmente, esta suposición se denomina **independencia condicional de clase**.

$P(clase|Datos) = \frac{P(Datos|clase)*P(clase)}{P(Datos)}$

Donde: + clase es una dalida en particular, en este caso será benigna

- datos son las características
- P(clase) se conoce como la **probabilidad anterior**. Es la probabilidad que ya se tiene
- P(Datos) se conoce como **probabilidad marginal**.
- P(clase|Datos) se conoce como **probabilidad posterior**. Es el resultado que se quiere encontrar
- P(Datos|clase) se conoce como **verosimilitud**

#### 1.1.1 Cómo funciona

En caso de que se tenga una sola característica, el clasificador Naive Bayes calcula la probabilidad de un evento en los siguientes pasos:

- 1 : calcular la probabilidad previa para las etiquetas de clase dadas.
- 2 : determinar la probabilidad de probabilidad con cada atributo para cada clase.
- 3 : poner estos valores en el teorema de Bayes y calcular la probabilidad posterior.
- 4 : ver qué clase tiene una probabilidad más alta, dado que la variable de entrada pertenece a la clase de probabilidad más alta.

### 1.2 Set de datos

Se usará un set de datos precargado en scikit-learn que tiene relación con datos de muestras de biopsias que pueden ser benignas o malignas.

[6]:
```
## Carga de datos
import pandas as pd
import numpy as np
from sklearn import datasets
dataset = datasets.load_breast_cancer()
data_frame = pd.DataFrame(np.c_[dataset['data'], dataset['target']],
                  columns= np.append(dataset['feature_names'], ['target']))
data_frame
```

[6]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | \ |
|---|---|---|---|---|---|---|
| 0 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| 2 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| 3 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| 4 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |
| .. | … | … | … | … | … | |
| 564 | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | |
| 565 | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | |
| 566 | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | |
| 567 | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | |
| 568 | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | |

| | mean compactness | mean concavity | mean concave points | mean symmetry | \ |
|---|---|---|---|---|---|
| 0 | 0.27760 | 0.30010 | 0.14710 | 0.2419 | |
| 1 | 0.07864 | 0.08690 | 0.07017 | 0.1812 | |
| 2 | 0.15990 | 0.19740 | 0.12790 | 0.2069 | |
| 3 | 0.28390 | 0.24140 | 0.10520 | 0.2597 | |
| 4 | 0.13280 | 0.19800 | 0.10430 | 0.1809 | |
| .. | … | … | … | … | |
| 564 | 0.11590 | 0.24390 | 0.13890 | 0.1726 | |
| 565 | 0.10340 | 0.14400 | 0.09791 | 0.1752 | |
| 566 | 0.10230 | 0.09251 | 0.05302 | 0.1590 | |
| 567 | 0.27700 | 0.35140 | 0.15200 | 0.2397 | |
| 568 | 0.04362 | 0.00000 | 0.00000 | 0.1587 | |

| | mean fractal dimension | … | worst texture | worst perimeter | worst area | \ |
|---|---|---|---|---|---|---|
| 0 | 0.07871 | … | 17.33 | 184.60 | 2019.0 | |
| 1 | 0.05667 | … | 23.41 | 158.80 | 1956.0 | |
| 2 | 0.05999 | … | 25.53 | 152.50 | 1709.0 | |
| 3 | 0.09744 | … | 26.50 | 98.87 | 567.7 | |
| 4 | 0.05883 | … | 16.67 | 152.20 | 1575.0 | |
| .. | … | … | … | … | … | |

```
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                …                  …                …
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890     0.0
1                  0.1860          0.2750                  0.08902     0.0
2                  0.2430          0.3613                  0.08758     0.0
3                  0.2575          0.6638                  0.17300     0.0
4                  0.1625          0.2364                  0.07678     0.0
..                    …               …                        …       …
564                0.2216          0.2060                  0.07115     0.0
565                0.1628          0.2572                  0.06637     0.0
566                0.1418          0.2218                  0.07820     0.0
567                0.2650          0.4087                  0.12400     0.0
568                0.0000          0.2871                  0.07039     1.0

[569 rows x 31 columns]
```

[7]:
```
# Tipos de datos de las columnas
data_frame.dtypes
```

[7]:
```
mean radius               float64
mean texture              float64
mean perimeter            float64
mean area                 float64
mean smoothness           float64
mean compactness          float64
mean concavity            float64
mean concave points       float64
mean symmetry             float64
mean fractal dimension    float64
```

```
radius error                float64
texture error               float64
perimeter error             float64
area error                  float64
smoothness error            float64
compactness error           float64
concavity error             float64
concave points error        float64
symmetry error              float64
fractal dimension error     float64
worst radius                float64
worst texture               float64
worst perimeter             float64
worst area                  float64
worst smoothness            float64
worst compactness           float64
worst concavity             float64
worst concave points        float64
worst symmetry              float64
worst fractal dimension     float64
target                      float64
dtype: object
```

[9]:
```
data_frame.describe()
```

[9]:

| | mean radius | mean texture | mean perimeter | mean area | \ |
|---|---|---|---|---|---|
| count | 569.000000 | 569.000000 | 569.000000 | 569.000000 | |
| mean | 14.127292 | 19.289649 | 91.969033 | 654.889104 | |
| std | 3.524049 | 4.301036 | 24.298981 | 351.914129 | |
| min | 6.981000 | 9.710000 | 43.790000 | 143.500000 | |
| 25% | 11.700000 | 16.170000 | 75.170000 | 420.300000 | |
| 50% | 13.370000 | 18.840000 | 86.240000 | 551.100000 | |
| 75% | 15.780000 | 21.800000 | 104.100000 | 782.700000 | |
| max | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | |

| | mean smoothness | mean compactness | mean concavity | mean concave points | \ |
|---|---|---|---|---|---|
| count | 569.000000 | 569.000000 | 569.000000 | 569.000000 | |
| mean | 0.096360 | 0.104341 | 0.088799 | 0.048919 | |
| std | 0.014064 | 0.052813 | 0.079720 | 0.038803 | |
| min | 0.052630 | 0.019380 | 0.000000 | 0.000000 | |
| 25% | 0.086370 | 0.064920 | 0.029560 | 0.020310 | |
| 50% | 0.095870 | 0.092630 | 0.061540 | 0.033500 | |
| 75% | 0.105300 | 0.130400 | 0.130700 | 0.074000 | |
| max | 0.163400 | 0.345400 | 0.426800 | 0.201200 | |

| | mean symmetry | mean fractal dimension | … | worst texture | \ |
|---|---|---|---|---|---|
| count | 569.000000 | 569.000000 | … | 569.000000 | |

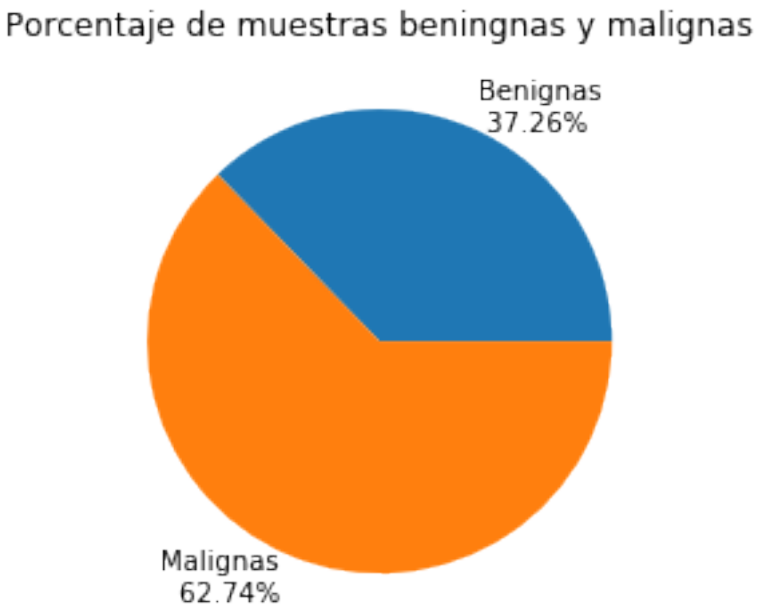
```
mean         0.181162                0.062798  …      25.677223
std          0.027414                0.007060  …       6.146258
min          0.106000                0.049960  …      12.020000
25%          0.161900                0.057700  …      21.080000
50%          0.179200                0.061540  …      25.410000
75%          0.195700                0.066120  …      29.720000
max          0.304000                0.097440  …      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness  \
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

### 1.2.1 Revisa la distribución de las observaciones respecto de la variable que se usará para la clasificación

[42]:
```
print(data_frame.groupby('target').size())
```

```
target
0.0    212
1.0    357
dtype: int64
```

[40]:
```
import matplotlib.pyplot as plt
# Gráfico de tortas del porcentaje de muestras benignas y mañignas
# Contando las benignas
clases = np.array([data_frame[data_frame.target == 0.0].shape[0],␣
 ↪data_frame[data_frame.target == 1.0].shape[0]])

# Creando las leyendas del grafico.
labels = [ str(round(x * 1.0 / clases.sum() * 100.0, 2)) + '%'  for x in clases␣
 ↪]
labels[0] = 'Benignas\n ' + labels[0]
labels[1] = 'Malignas\n ' + labels[1]

plt.pie(clases, labels=labels)
plt.title('Porcentaje de muestras beningnas y malignas')
plt.show()
```



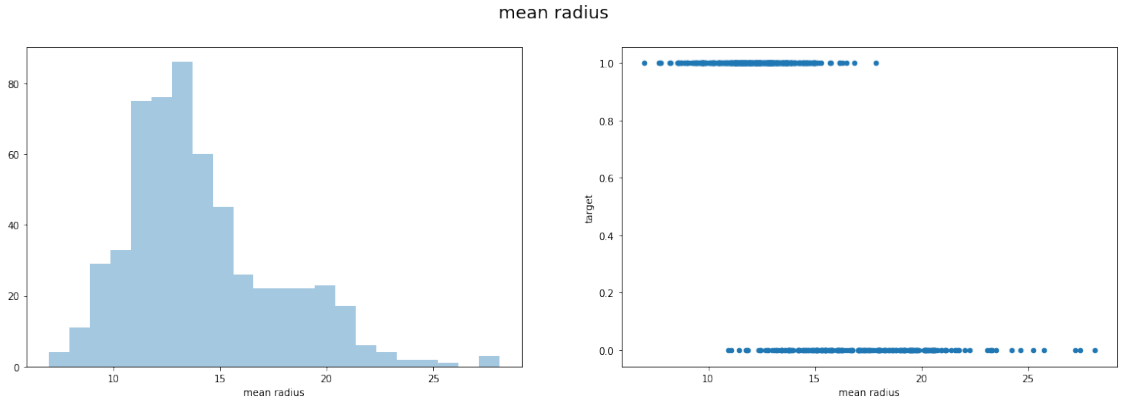

### 1.2.2 Comentarios hasta este punto

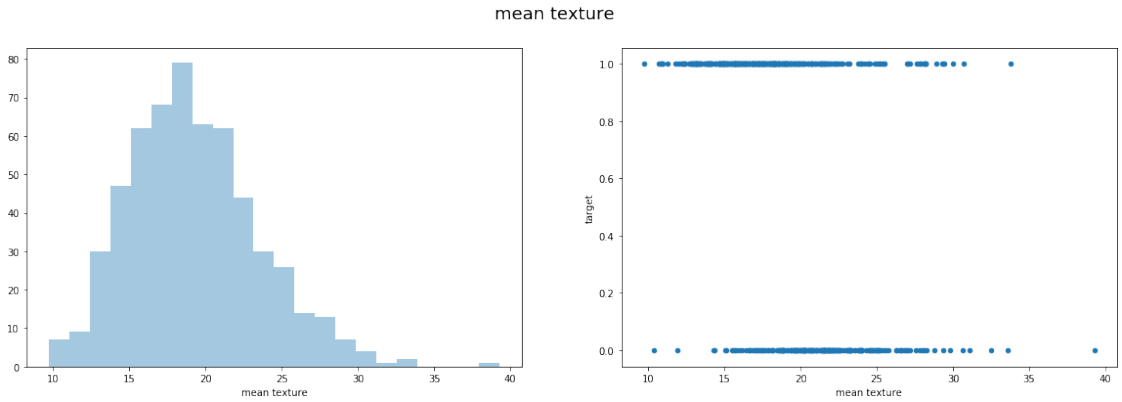
- Hay 569 observaciones con 31 columnas, siendo una de ellas la columna 'target' que es la que indica si es o no es cáncer.

- Se observa que hay una distribución aceptable de muestras malignas y benignas.

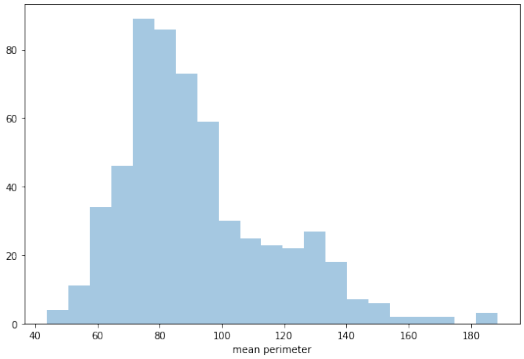
### 1.2.3 Análisis de distribución

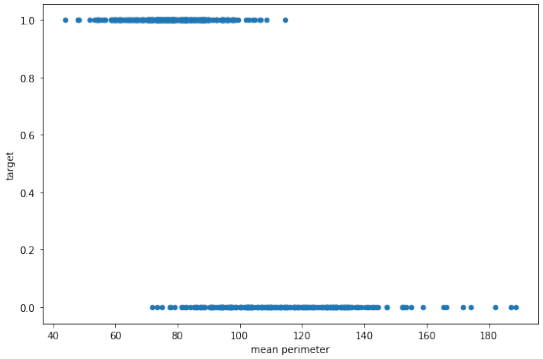
[56]:
```
import seaborn as sns
columnas = np.array(data_frame.columns)

for col in columnas[:-1]:
    fig, ax =plt.subplots(1, 2, figsize=(20, 6))
    fig.suptitle(col, fontsize=18)
    sns.distplot(data_frame[col], ax=ax[0], kde=False)
    data_frame[[col]+['target']].plot.scatter(x=col, y='target', ax=ax[1])
    plt.show()
```



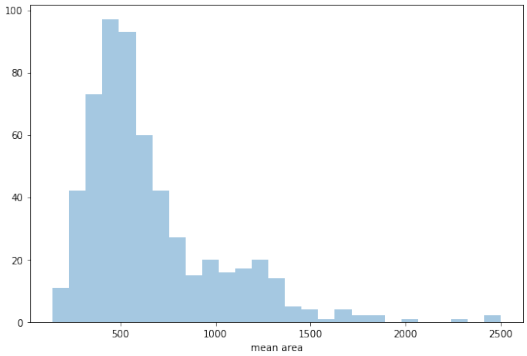



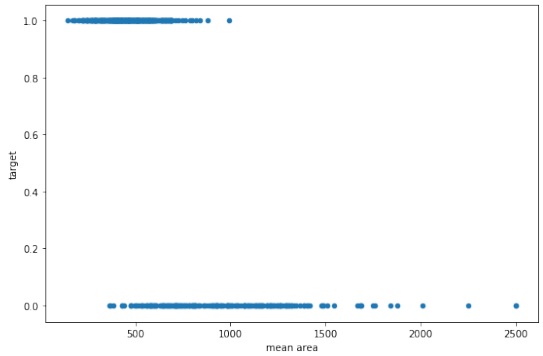
mean perimeter



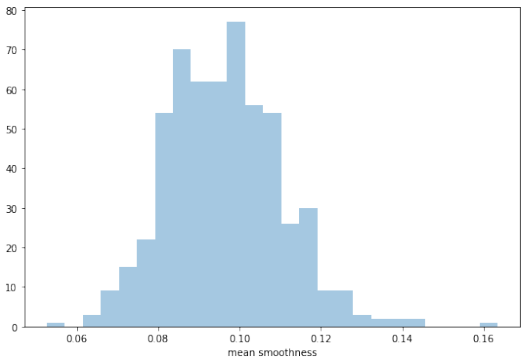



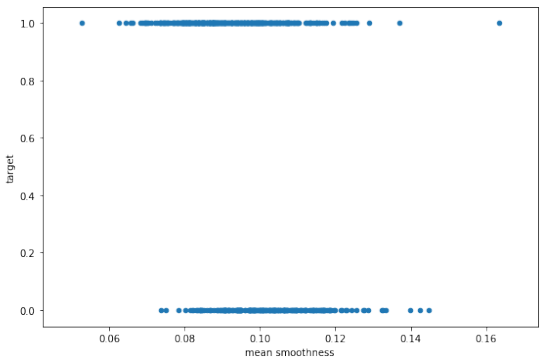

mean area



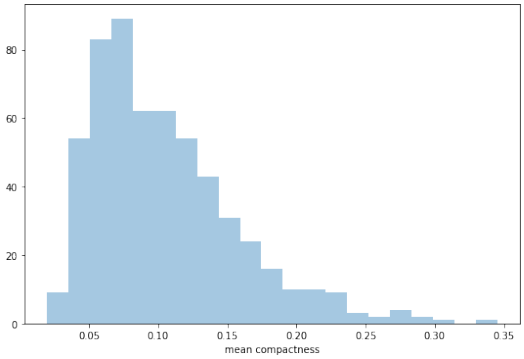



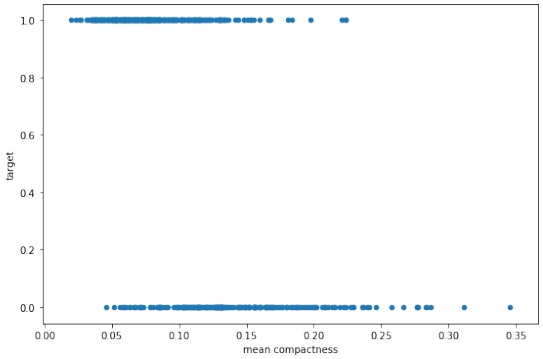

mean smoothness



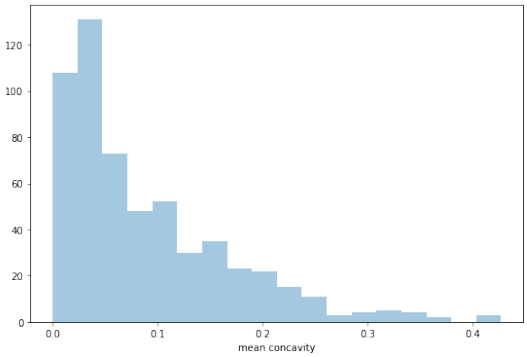



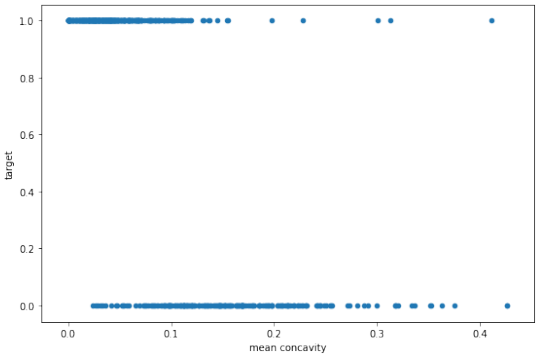
mean compactness



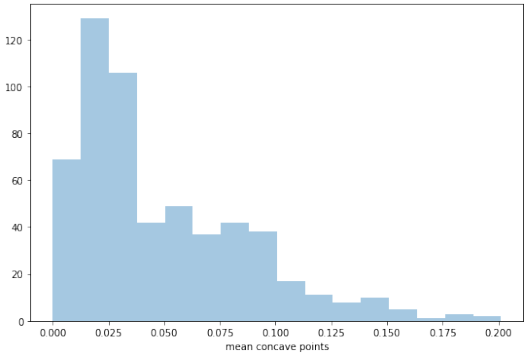



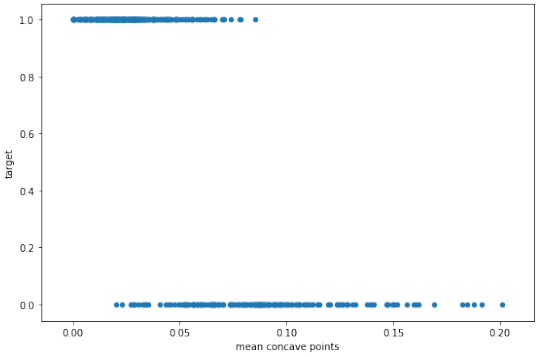

mean concavity



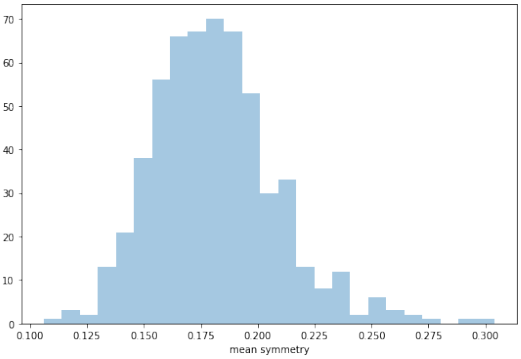



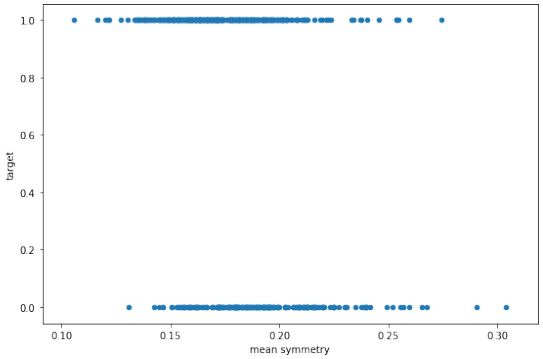

mean concave points



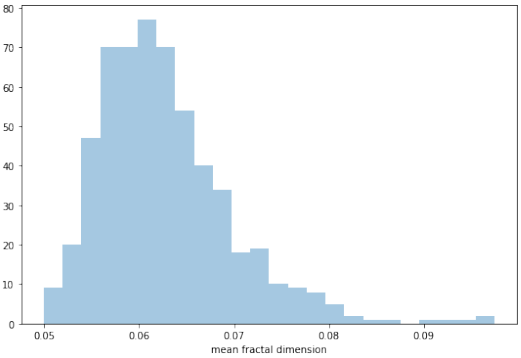



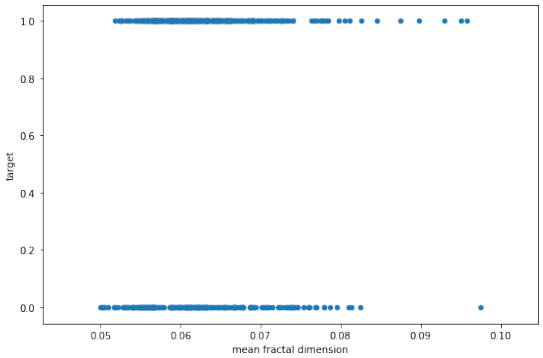
mean symmetry



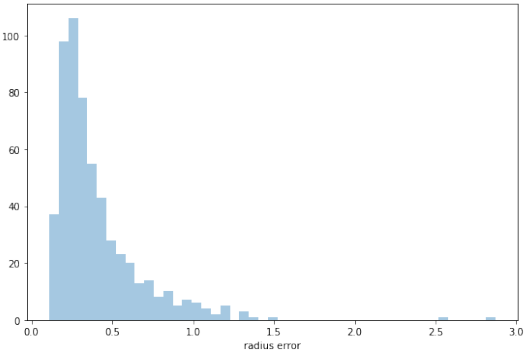



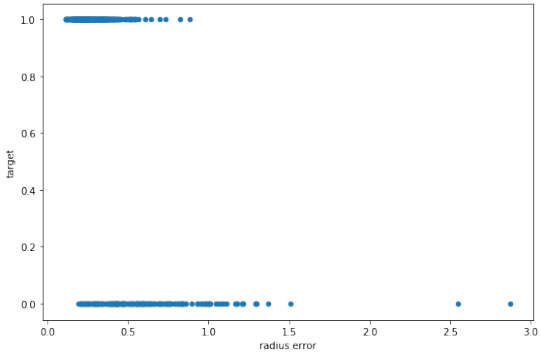

mean fractal dimension



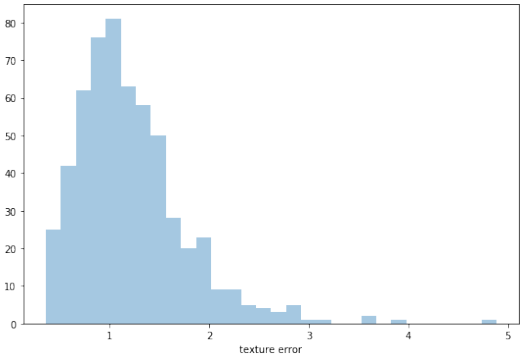



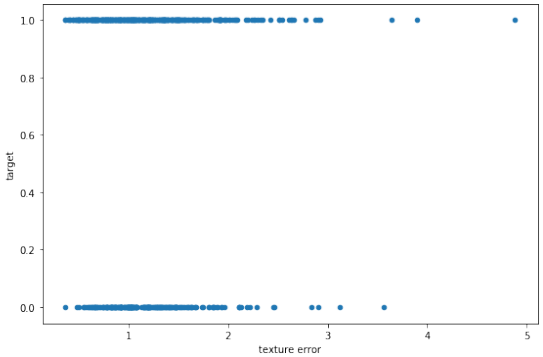

radius error



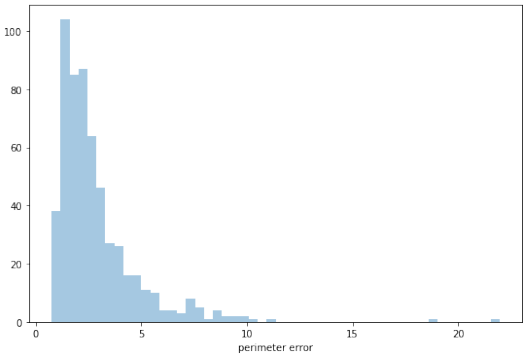



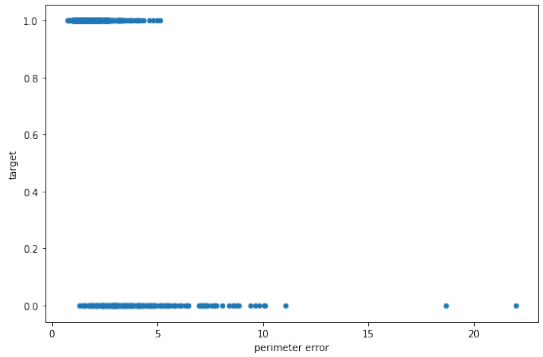
texture error



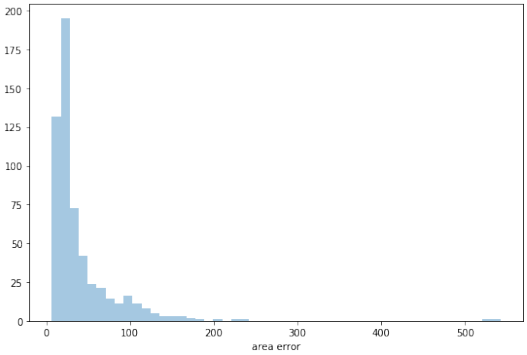



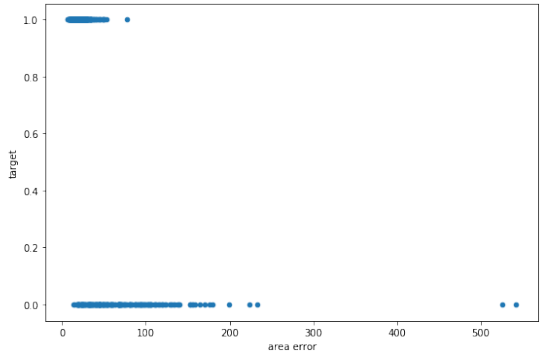

perimeter error



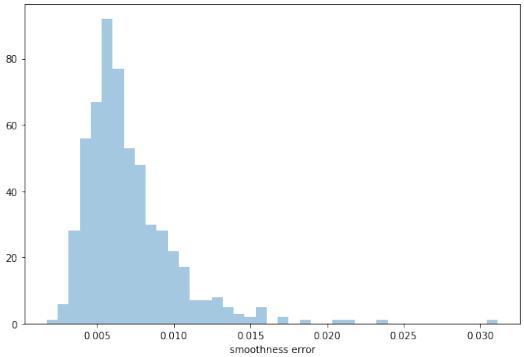



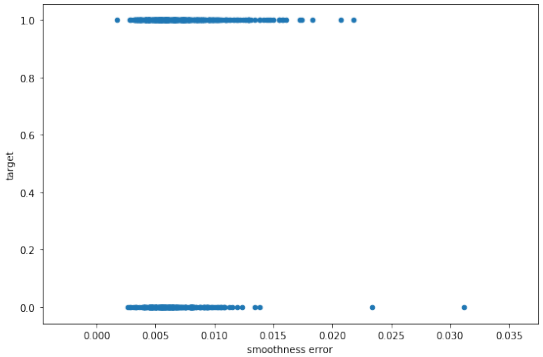

area error



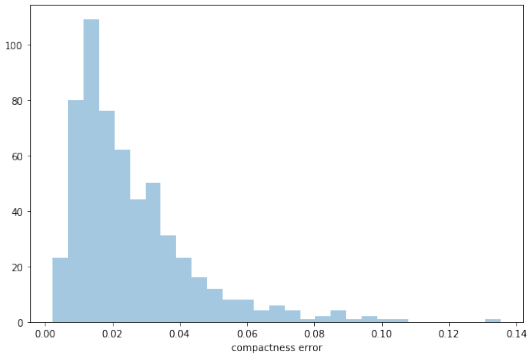



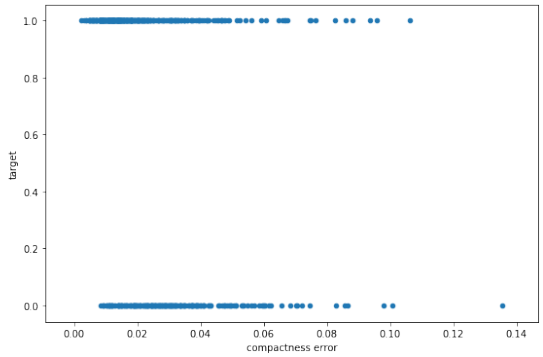
smoothness error



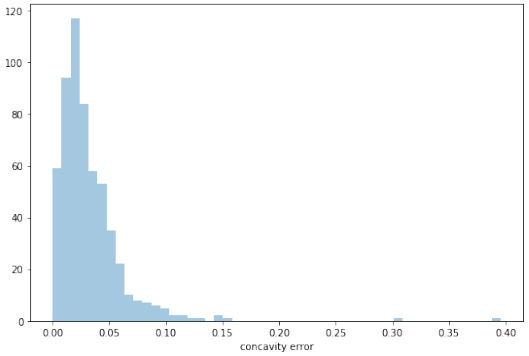



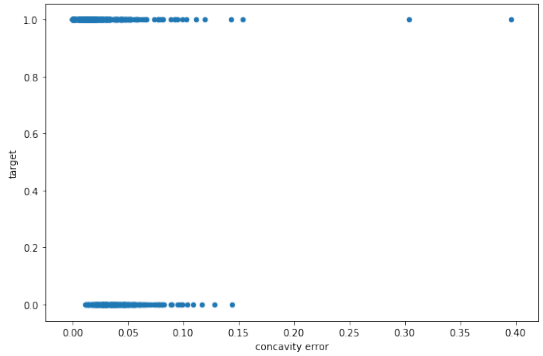

compactness error



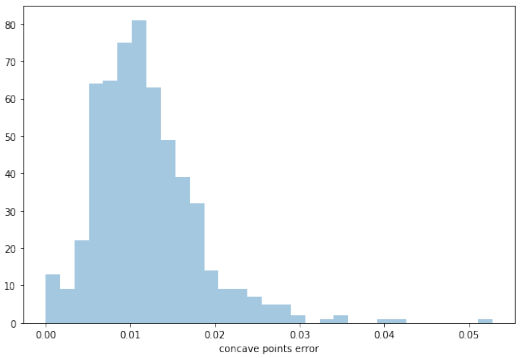



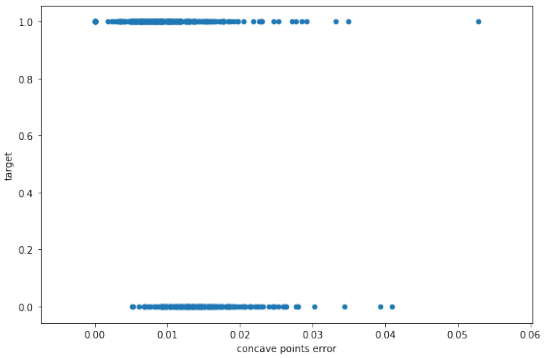

concavity error



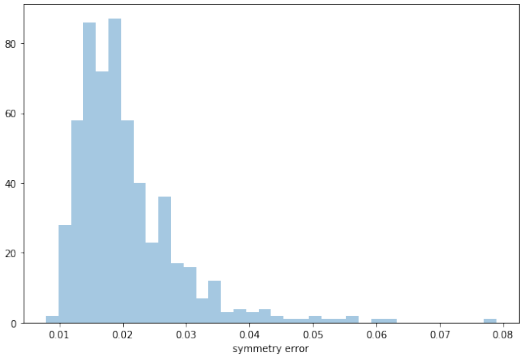



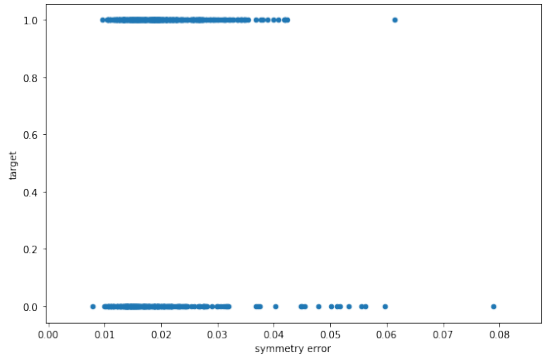
concave points error



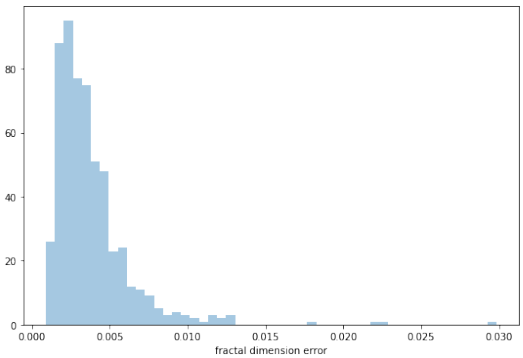



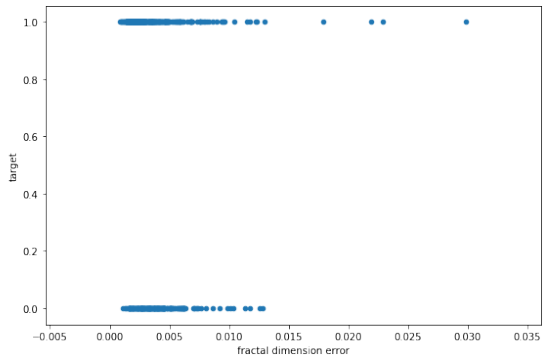

symmetry error



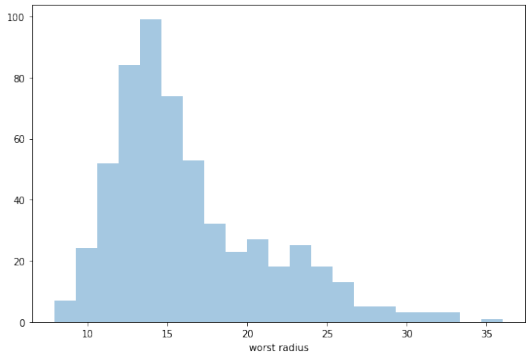



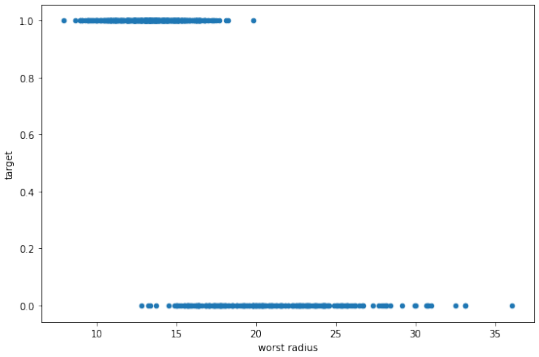

fractal dimension error



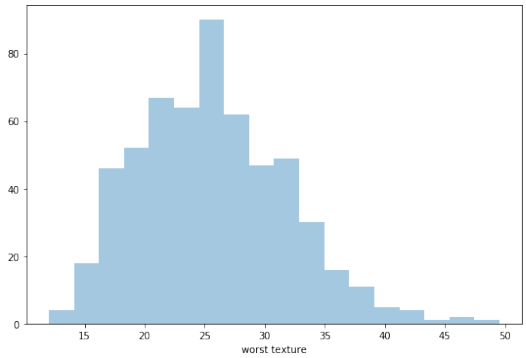



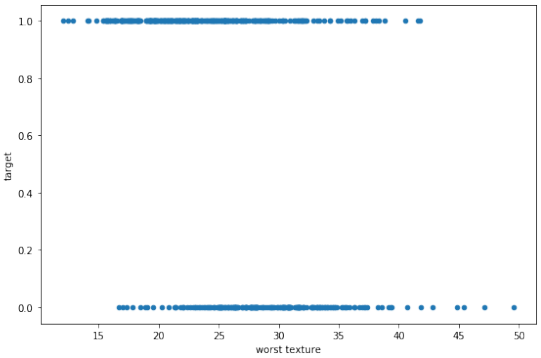
worst radius



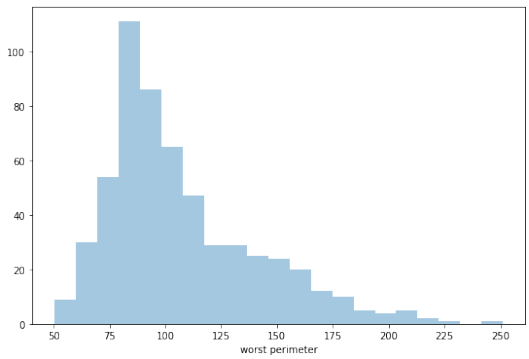



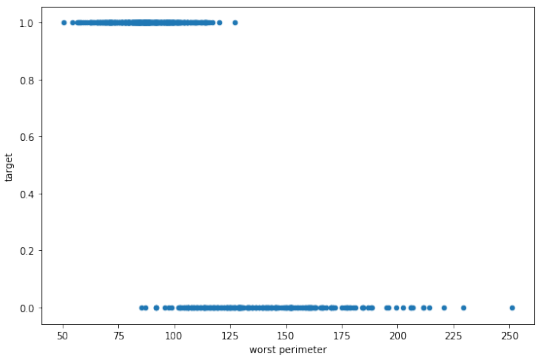

worst texture



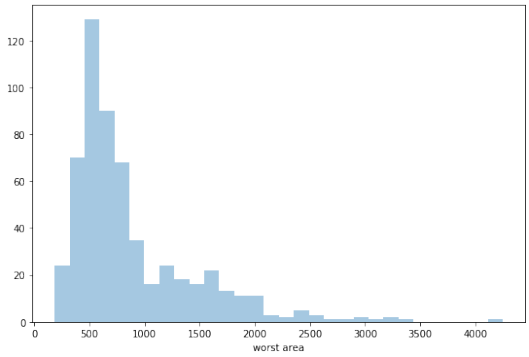



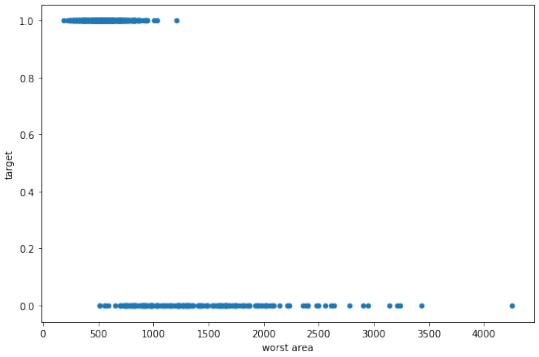

worst perimeter



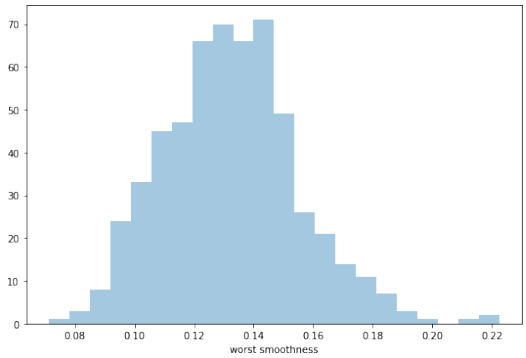



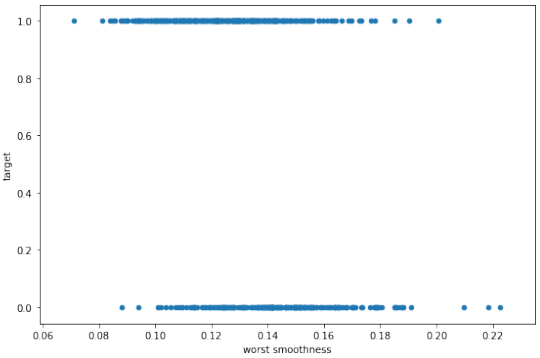
worst area



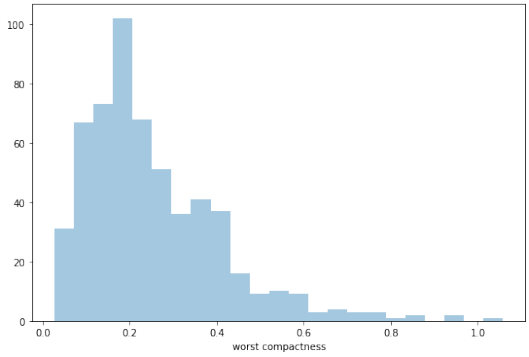



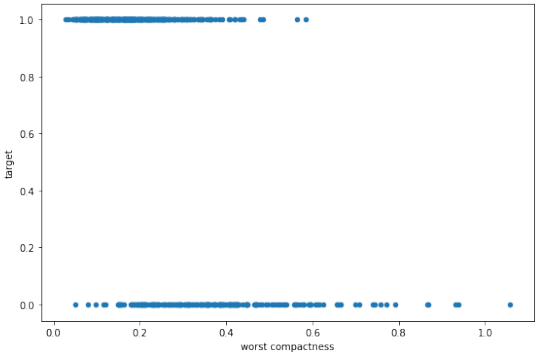

worst smoothness



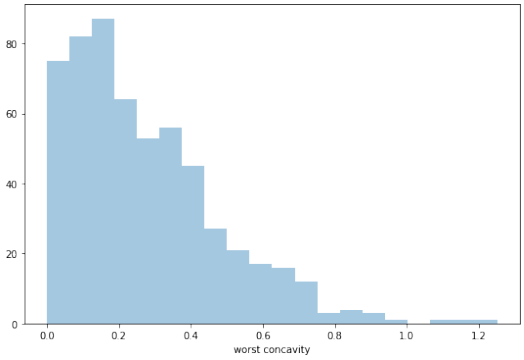



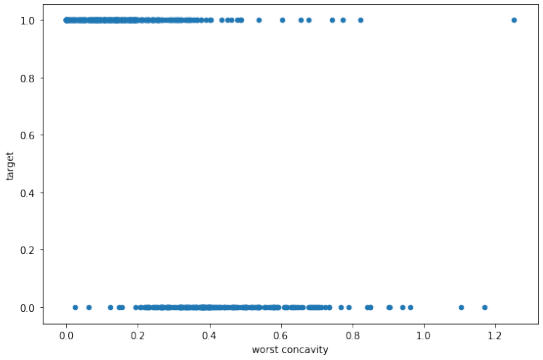

worst compactness



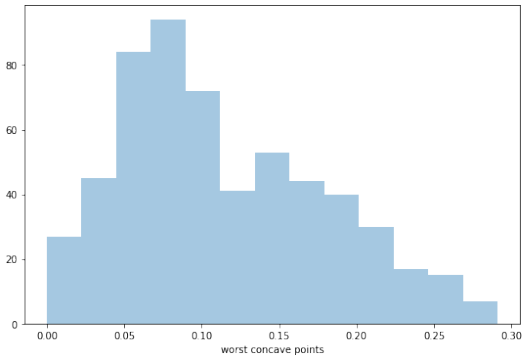



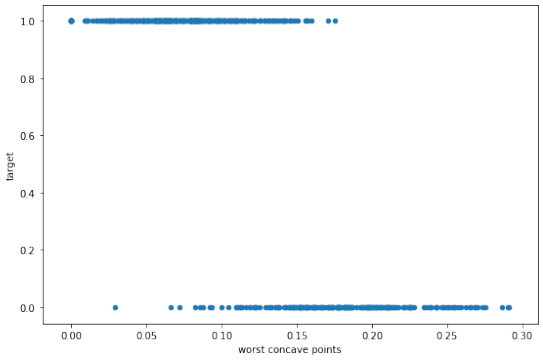
worst concavity



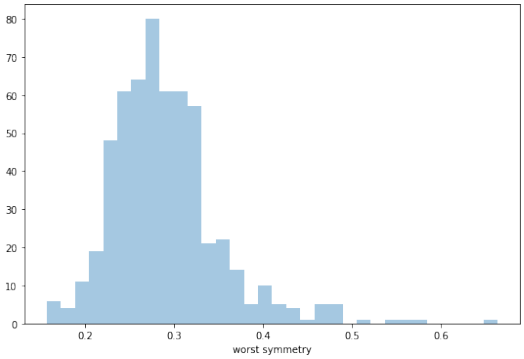



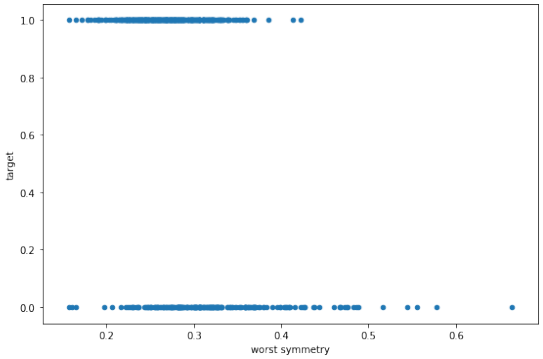

worst concave points

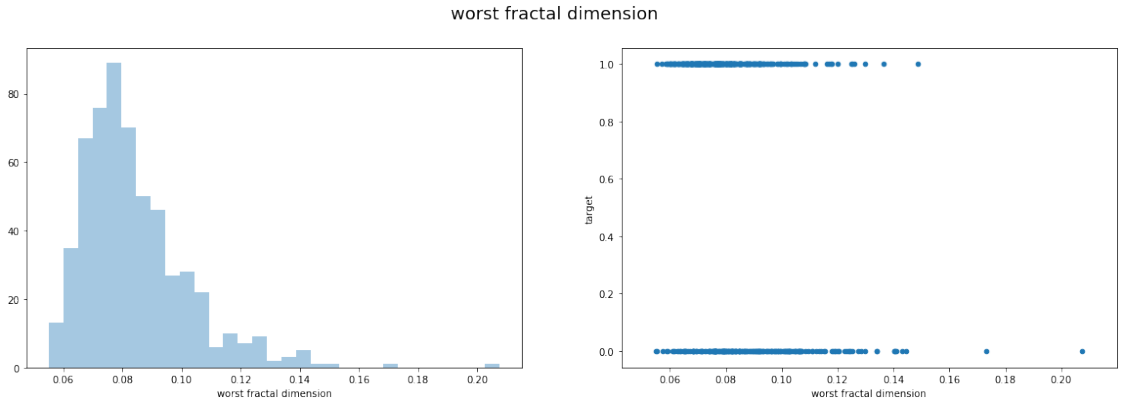



worst symmetry






## 1.3 Preparación de los datos

[15]:
```
# Selecciona las variables
X = data_frame.drop(["target"],axis=1)

# Rescata la etiqueta
y = data_frame.target
```

## 1.4 Creando el modelo

### 1.4.1 Ideas previas

Se está tratando de elegir entre 2 clases: benigna o maligna; entonces una forma de tratar la decisión es calcular la tasa de probabilidades a posterior.

Cuando se trabaja con Naive Bayes se debe escoger un **clasificador**. Uno de los tipos de clasificadores más populares es el **Gaussian Naive Bayes Classifier**.

Hay otros clasificadores Bayesianos (https://scikit-learn.org/stable/modules/naive_bayes.html#).

La fórmula del clasificador, usando 2 clases (benigna, maligna) y 2 características (solo para explicar la fórmula): mean_radius, mean_texture.

$$P(benigna|datos) = \frac{P(benigna)*P(mean\_radius|benigna)*P(mean\_texture|benigna)}{P(datos)}$$

$$P(maligna|datos) = \frac{P(maligna)*P(mean\_radius|maligna)*P(mean\_texture|maligna)}{P(datos)}$$

donde:

- P(benigna) es la probabilidad que ya se tiene. Corresponde al número de veces que el valor de target = 0.0 en el conjunto de datos, dividido el total de observaciones. En este caso (como ya se revisó) es 212/569
- $P(mean_radius|benigna), P(mean_texture|benigna) $ es la verosimilitud.

Los nombres Gaussian y Naive (ingenuo) de algoritmo vienen de dos suposiciones:

- Se asume que las características de la verosimilitud no estan correlacionada entre ellas. Esto seria que el promedio de radio y la etxtura promedio son independienets entre ellos. Dado que eso no siempre puede ser cierto y es una suposición ingenua es que aparece en el nombre **naive bayes**
- Se asume que el valor de las características (mean_radius, mean_texture y todas las que se quieran agregar) tendrán una distribución normal (gaussiana). Esto permite calcular cada parte p(mean_radius|benigna) usando la función de probabilidad de densidad normal.

### 1.5 Obtención de datos de entrenamiento y validación

[21]:
```
from sklearn.model_selection import train_test_split

# Se separan los datos de "train" en entrenamiento y prueba para probar los␣
 ↪algoritmos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state = 29)

# Define el algoritmo a utilizar Naive Bayes
from sklearn.naive_bayes import GaussianNB

modelo = GaussianNB()

# Entrenamiento del modelo
modelo.fit(X_train, y_train)

# Validación del modelo
y_pred = modelo.predict(X_test)
```

[22]:
```
# Evaluación del modelo
from sklearn.metrics import confusion_matrix

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz)

# Se calcula la precisión del modelo
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred)
print('Precisión del modelo:', precision)
```

```
Matriz de Confusión:
[[35  4]
 [ 3 72]]
```

```
Precisión del modelo:
0.9473684210526315
```

### 1.6 Interpretación de resultados

Desde la matriz de confusión:

- Se obtienen 107 datos predichos correctamente
- Se obtienen 7 datos erróneos

#### 1.6.1 Acerca de la matriz de confusión

La imagen muestra la estructura de la matriz de confusión:


Si se considera que una muestra se dice beningna si está etiquetada como no cancerígena y maligna en caso contrario.

Considerando lo anterior esto se interpreta la matriz de la siguiente forma:

- 1. Muestra maligna y el modelo la clasificó como maligna (+) . Esto sería un verdadero positivo o VP .
- 2. Muestra maligna y el modelo lo clasificó como benigna (-) . Este seria un verdadero negativo o sea un VN.
- 3. Muestra benigna y el modelo lo clasificó como benigna (-) . Este seria un error tipo II o falso negativo o FN.

- 4. Muestra benigna y el modelo lo clasificó como maligna (+) . Este es un error tipo I, o falso positivo o FP.

### 1.7 Otras medidas

https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

[67]:
```
from sklearn.metrics import accuracy_score, f1_score, recall_score

accuracy = accuracy_score(y_test, y_pred)
print('Accurancy del modelo:',accuracy)

print('F1 score del modelo:',f1_score(y_test, y_pred))
print('Recall del modelo:',recall_score(y_test, y_pred))
```

```
Accurancy del modelo: 0.9298245614035088
F1 score del modelo: 0.9473684210526316
Recall del modelo: 0.96
```

### 1.8 Usando un selector de columnas

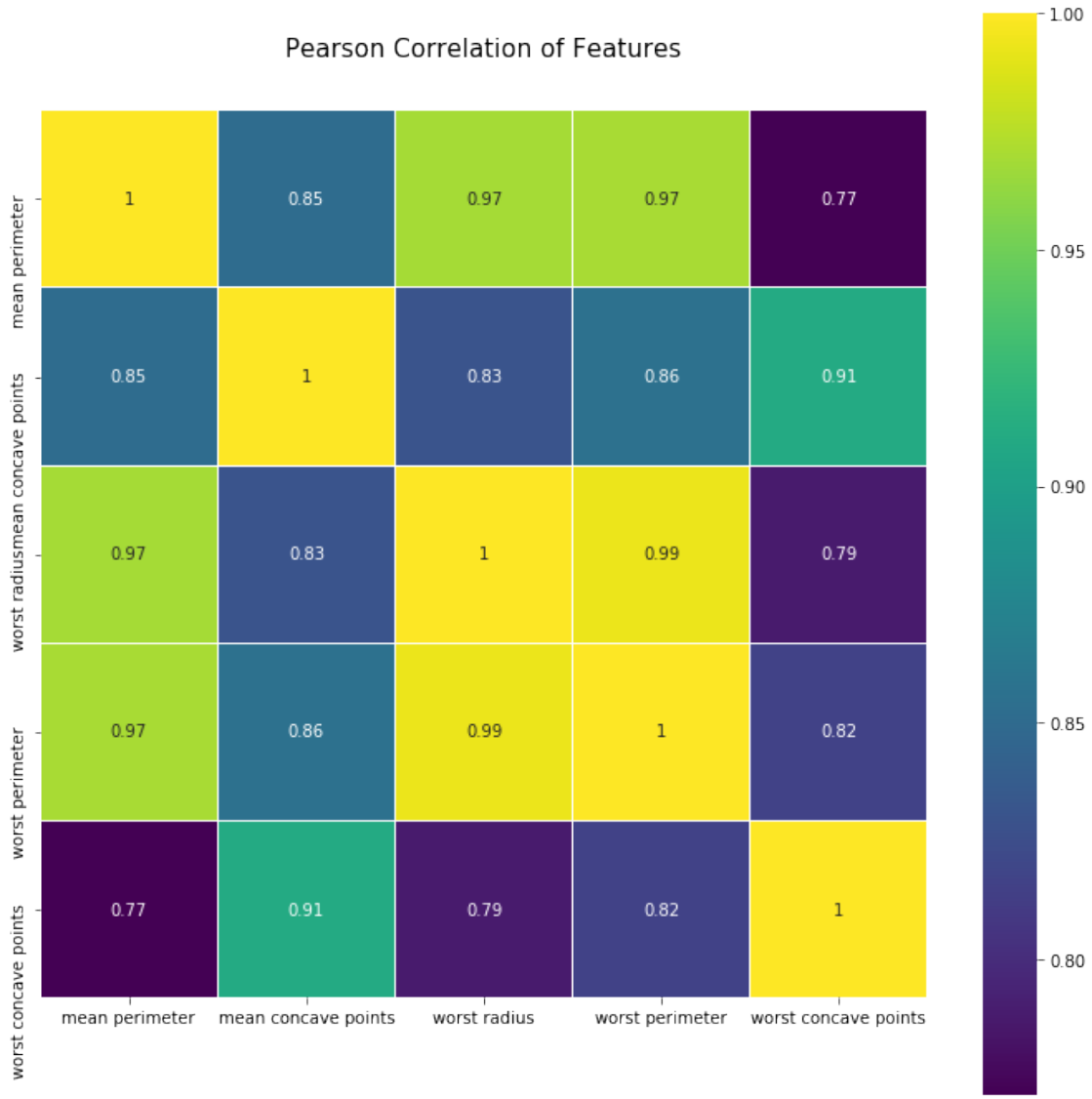
Para mejorar los resultados con este algoritmo. En vez de utilizar las 30 columnas de datos de entrada que se tienen, se va a utilizar una Clase de SkLearn llamada **SelectKBest** con la que seleccionaremos las 5 mejores características y usarán sólo esas.

[58]:
```
from sklearn.feature_selection import SelectKBest
X=data_frame.drop(['target'], axis=1)
y=data_frame['target']

best=SelectKBest(k=5)
X_new = best.fit_transform(X, y)
X_new.shape
selected = best.get_support(indices=True)
print(X.columns[selected])
```

```
Index(['mean perimeter', 'mean concave points', 'worst radius',
       'worst perimeter', 'worst concave points'],
      dtype='object')
```

[60]:
```
import seaborn as sb
used_features =X.columns[selected]

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(data_frame[used_features].astype(float).corr(),linewidths=0.1,vmax=1.
 ↪0, square=True, cmap=colormap, linecolor='white', annot=True)
```

[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7f825fa4f048>




[62]:
```
X = data_frame[used_features]
# Se separan los datos de "train" en entrenamiento y prueba para probar los␣
 ↪algoritmos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state = 29)

modelo_x = GaussianNB()

# Entrenamiento del modelo
modelo_x.fit(X_train, y_train)
```

```
# Validación del modelo
y_pred = modelo_x.predict(X_test)
```

[63]:
```
# Evaluación del modelo
matriz_x = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(matriz_x)

precision = precision_score(y_test, y_pred)
print('Precisión del modelo:')
print(precision)
```

```
Matriz de Confusión:
[[34  5]
 [ 3 72]]
Precisión del modelo:
0.935064935064935
```