# Applicability of basic machine learning algorithms for breast cancer detection

Maximilian Eineder

# Abstract

In this project, a dataset containing geometric characteristics of cell nuclei found in samples of breast mass was examined. Each record of the dataset is classified as benign (sample from healthy tissue) or malignant (sample from cancerous tissue). The research questions I tried to answer using this dataset were:

1. Is it possible to accurately detect breast cancer with basic machine learning algorithms like the ones introduced in the Python for Data Science course with similar data?
2. How do these basic algorithms perform in comparison with more advanced machine learning techniques?
3. Is it possible to determine which of the features are important to the classification, and which ones have little or no contribution?

To answer the first research question, models using four basic machine learning algorithms were created, trained with 2/3 of the records in the dataset. Then, the accuracy of the models was measured using the rest of the data of the dataset. Although the results show a surprisingly high accuracy of ca. 95% for these algorithms, a real application would still require a higher accuracy.

To answer the second research question, existing notebooks on https://www.kaggle.com, in which others have described their results with the same dataset using more advanced techniques like Artificial Neural Networks, were examined and their findings were compared to the findings in this project. This comparison showed that the accuracy of such advanced techniques was not dramatically better (around 98%).

To answer the third research question, I first examined feature distribution plots and tried to predict which features should be the most important ones. Then I compared my predictions with feature importance values calculated by the models used. This showed that it is possible to identify the important features for the classification task.

# Motivation

The research of this project was carried out because if even basic machine learning algorithms can produce prediction results that are close to acceptable for a real application, then it should be possible to build models that produce very accurate results. Such models could be the basis for quicker and less expensive cancer screenings, because only a small number of inconclusive results would have to be reviewed by a pathologist. As a result, the risk of developing cancer for the population could be reduced.

# Dataset

The dataset used in this project is called *Breast Cancer Wisconsin (Diagnostic)* and was found at https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. It contains 569 records. Each record contains 30 attributes of cell nuclei found in samples from breast tissue and a classification as „malignant“ or „benign“.

# Data Preparation and Cleaning

The dataset contains one unnamed column whose purpose was unclear. This column was removed. Apart from this, the dataset contains no null values or other noise, so no other data was removed.

# Research Question(s)

In this project there are three research questions:

1. Is it possible to accurately detect breast cancer with basic machine learning algorithms like the ones introduced in the Python for Data Science course with similar data?
2. How do these basic algorithms perform in comparison with more advanced machine learning techniques?
3. Is it possible to determine which of the features are important to the classification, and which ones have little or no contribution?

# Methods (I)

To answer the first research question, models using four basic machine learning algorithms were created:

- Decision Tree Classifier
- Random Forest Classifier
- Gaussian Naïve Bayes Classifier
- k-Nearest Neighbors Classifier

The models were trained with 2/3 of the records in the dataset. Then, the accuracy of the models was measured using the rest of the data of the dataset.

# Methods (II)

To answer the second research question, existing notebooks on https://www.kaggle.com, in which others have described their results with the same dataset using more advanced techniques were examined and their findings compared to the findings in the project. The examined techniques were

- Artificial Neural Networks
- Gradient Boosting Machine
- Support Vector Machine

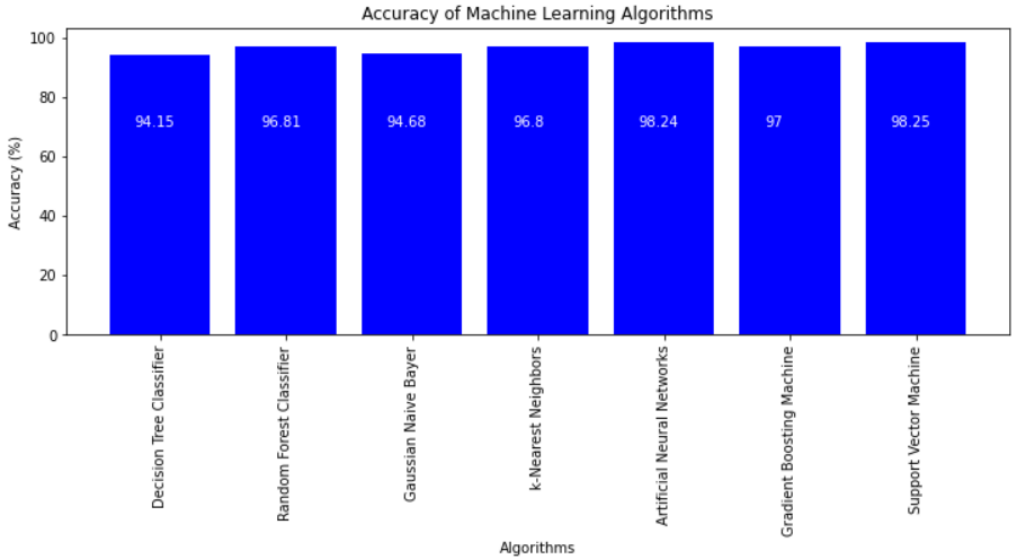
# Methods (III)

To answer the third research question, I first examined feature distribution plots and tried to predict which features should be the most important ones. Then I compared my predictions with feature importance values calculated by the models used.

# Findings (I)

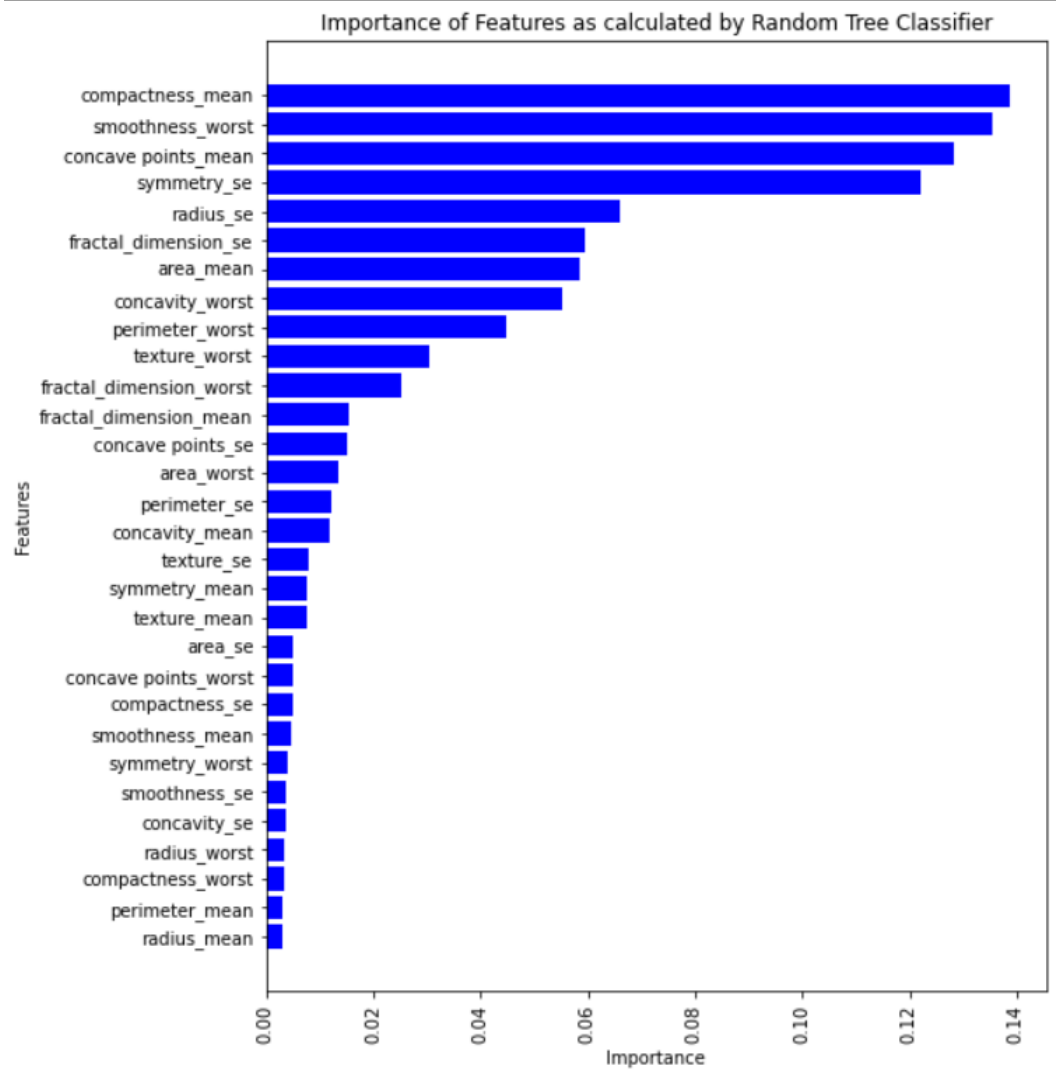
Comparison of basic and advanced machine learning techniques. The results show:

- Even basic algorithms (first four bars) have a remarkable accuracy of ca. 95% (Research question 1)
- More advanced techniques (last three bars) show a higher accuracy, but not dramatically higher (Research question 2)



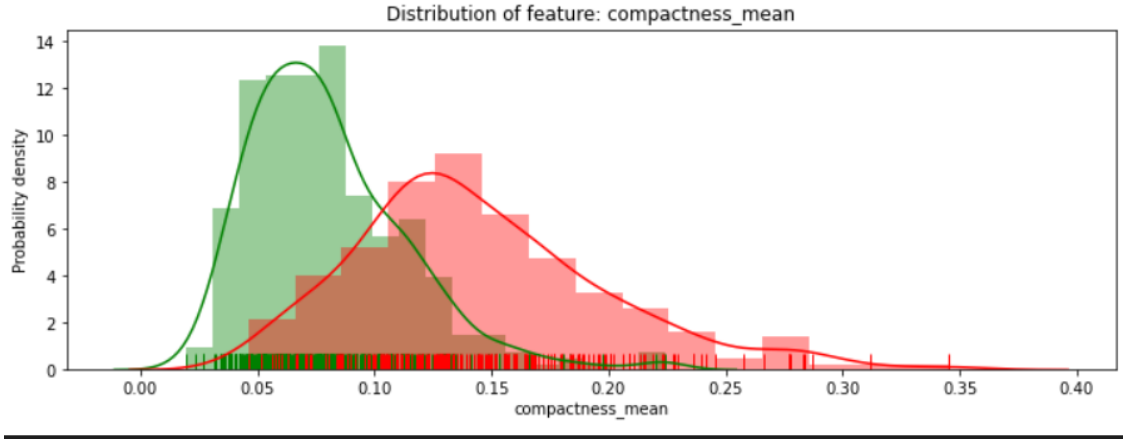
# Findings (II)

It is clearly possible to identify features that highly contribute to classification and others that nearly don‘t contribute („feature importance“). As anticipated, the most important features tend to be the ones where the distribution for benign and malignant samples differed most (see next slides).



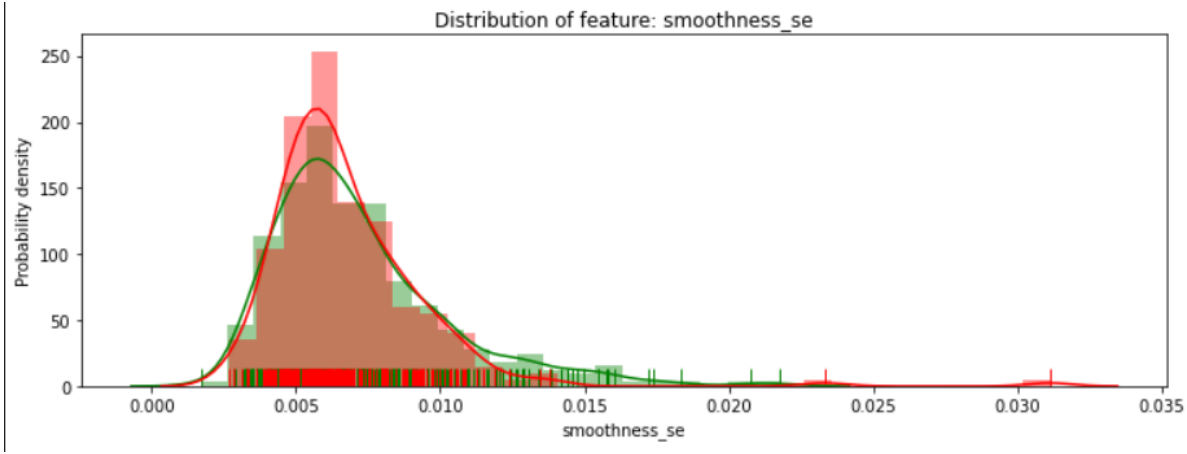
# Findings (IV)

Example for a feature with high importance and significant difference in feature distribution (green: benign samples; red: malignant samples)



# Findings (V)

Example for a feature with low importance and only little difference in feature distribution (green: benign samples; red: malignant samples)



# Limitations

The main limition that should be mentioned is that the findings are based on a very small dataset, which contains only 569 records. To draw defenitive conclusions, a much larger dataset would be required.
In addition, the data of this dataset is of very good quality, because it contains no null values and nearly no implausible outliers. Therefore, the results might lead to more optimistic conclusions than appropriate, because in real applications one would expect data of lower quality.

# Conclusions(I)

**Research question one:**

Is it possible to accurately detect breast cancer with basic machine learning algorithms like the ones introduced in the Python for Data Science course with similar data?

**Although the results show a surprisingly high accuracy of ca. 95% for these algorithms, a real application would still require higher accuracy.**

# Conclusions(II)

**Research question two:**

How do these basic algorithms perform in comparison with more advanced machine learning techniques?

**This comparison shows that the accuracy of such advanced techniques was not dramatically better (around 98% accuracy compared to around 95%).**

# Conclusions(III)

**Research question three:**

Is it possible to determine which of the features are important to the classification, and which ones have little or no contribution?

**It is possible to clearly identify features that have a high contribution to classification and others that have little to no contribution.**

# Acknowledgements

I would like to thank https://www.kaggle.com and the authors of the dataset https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Also, I would like to thank the authors of the notebooks who examined the advanced ML techniques which I used for data comparison with my own findings.

# References


- Notebook on Artificial Neural Networks (ANN): https://www.kaggle.com/thebrownviking20/intro-to-keras-with-breast-cancer-data-ann
- Notebook on Gradient Boosting Machine (GBM): https://www.kaggle.com/gpreda/breast-cancer-prediction-from-cytopathology-data
- Notebook on Support Vector Machine (SVM): https://www.kaggle.com/faressayah/support-vector-machine-pca-tutorial-for-beginner

# Appliability of basic machine learning algorithms for breast cancer diagnosis based on geometric characteristics of cell nuclei

## Description of the data set

In this notebook I am going to analyze a data set containing geometric characteristics found in samples of breast mass. The data set also contains a classification of each sample as benign (sample was taken from breast tissue that does not contain) or malignant (sample was taken from breast tissue that contains cancer).

## Research questions

The research questions I would like to answer using this data set are:

1. Using data similar to data in this data set, would it be possible to accurately detect breast cancer with basic machine learning algorithms like the ones introduced in the Python for Data Science course?
2. How do these basic algorithms perform in comparison with more advanced machine learning techniques?
3. Is it possible to determine which of the features are important to the classification, and which ones have little or no contribution?

## Research methods

To answer my first research question, I am going to pick three or four basic machine learning algorithms, train them with a portion of the data in the data set, and use the rest of the data to test their accuracy. I am also going to try different values for input parameters like the maximum node count of the decision tree and see which values result in the best accuracy.

To answer the second research question, I will examine existing notebooks on https://www.kaggle.com (https://www.kaggle.com) in which other people have described their results with the same data set using more advanced techniques like Artificial Neural Networks.

To answer the third research question, I will first examine feature distribution plots and try to predict which features should turn out to be most important to the algorithms and then compare my predictions with feature importance values calculated by the models.

First, I am going to explore the data set and make myself familiar with the data, its shape, the size of the data set and data quality. Next, I am going to

## Exploring the data set

The data set was retrieved from https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)

In [1]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:

```
df = pd.read_csv('breast_cancer_classification.csv')
df.head()
```

Out[2]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothnes |
|---|---|---|---|---|---|---|---|
| **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 | |

5 rows × 33 columns

In [3]:

```
df.describe()
```

Out[3]:

| | id | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_ |
|---|---|---|---|---|---|---|
| **count** | 5.690000e+02 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.0 |
| **mean** | 3.037183e+07 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.0 |
| **std** | 1.250206e+08 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.0 |
| **min** | 8.670000e+03 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.0 |
| **25%** | 8.692180e+05 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.0 |
| **50%** | 9.060240e+05 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.0 |
| **75%** | 8.813129e+06 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.1 |
| **max** | 9.113205e+08 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.1 |

8 rows × 32 columns

In [4]:

```
df.isnull().any()
```

Out[4]:

```
id                         False
diagnosis                  False
radius_mean                False
texture_mean               False
perimeter_mean             False
area_mean                  False
smoothness_mean            False
compactness_mean           False
concavity_mean             False
concave points_mean        False
symmetry_mean              False
fractal_dimension_mean     False
radius_se                  False
texture_se                 False
perimeter_se               False
area_se                    False
smoothness_se              False
compactness_se             False
concavity_se               False
concave points_se          False
symmetry_se                False
fractal_dimension_se       False
radius_worst               False
texture_worst              False
perimeter_worst            False
area_worst                 False
smoothness_worst           False
compactness_worst          False
concavity_worst            False
concave points_worst       False
symmetry_worst             False
fractal_dimension_worst    False
Unnamed: 32                 True
dtype: bool
```

In [5]:

```
df = df.drop(df.columns[32], axis=1)
```

In [6]:

```
df.isnull().any()
```

Out[6]:

```
id                        False
diagnosis                 False
radius_mean               False
texture_mean              False
perimeter_mean            False
area_mean                 False
smoothness_mean           False
compactness_mean          False
concavity_mean            False
concave points_mean       False
symmetry_mean             False
fractal_dimension_mean    False
radius_se                 False
texture_se                False
perimeter_se              False
area_se                   False
smoothness_se             False
compactness_se            False
concavity_se              False
concave points_se         False
symmetry_se               False
fractal_dimension_se      False
radius_worst              False
texture_worst             False
perimeter_worst           False
area_worst                False
smoothness_worst          False
compactness_worst         False
concavity_worst           False
concave points_worst      False
symmetry_worst            False
fractal_dimension_worst   False
dtype: bool
```

In [7]:

```
features = df.columns
features
```

Out[7]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       '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')
```

In [8]:

```
features = features.drop(['id', 'diagnosis'])
features
```

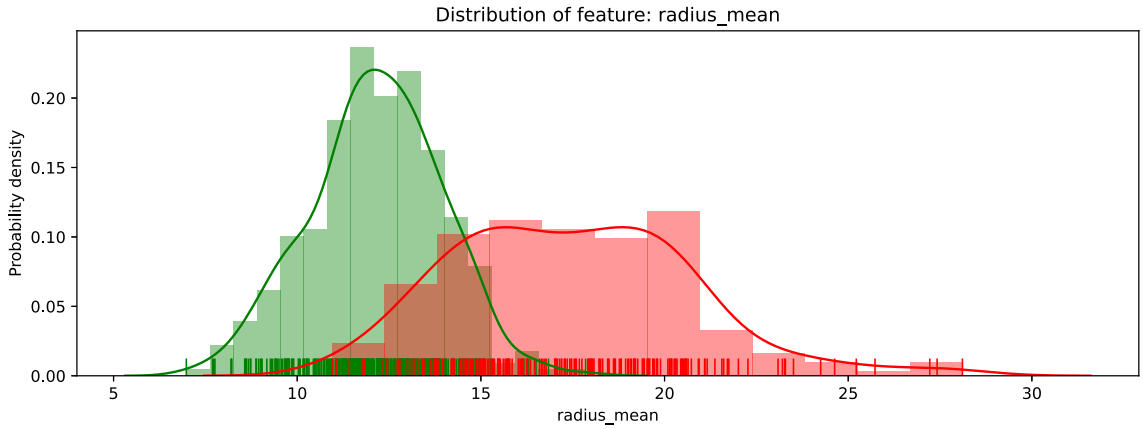
Out[8]:

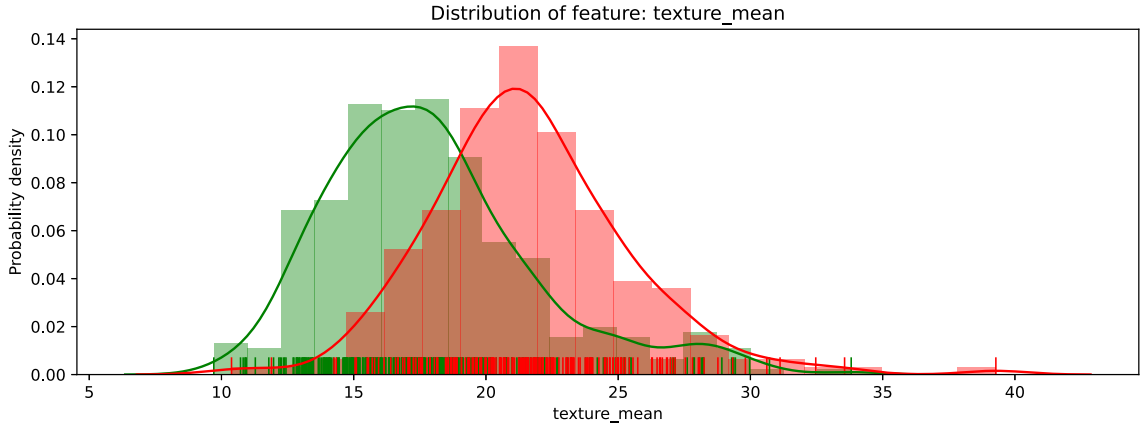
```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       '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')
```

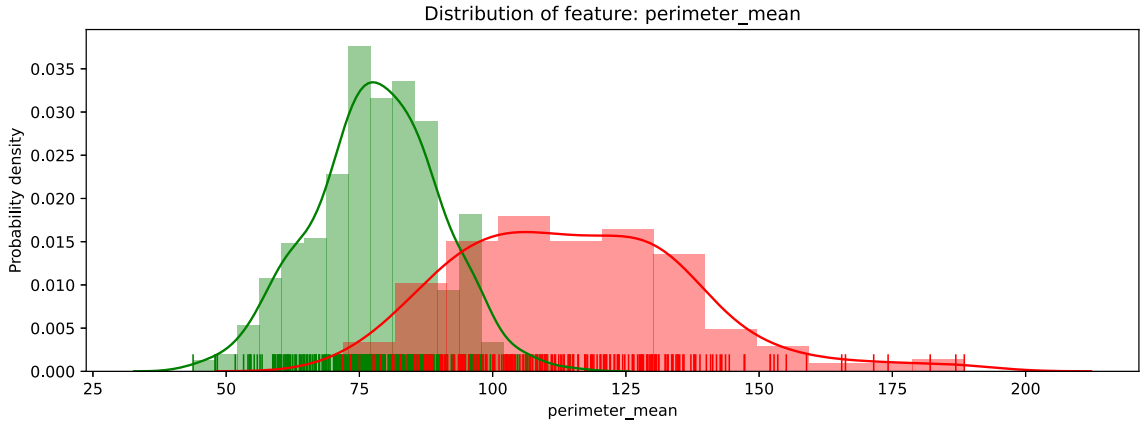
In [9]:

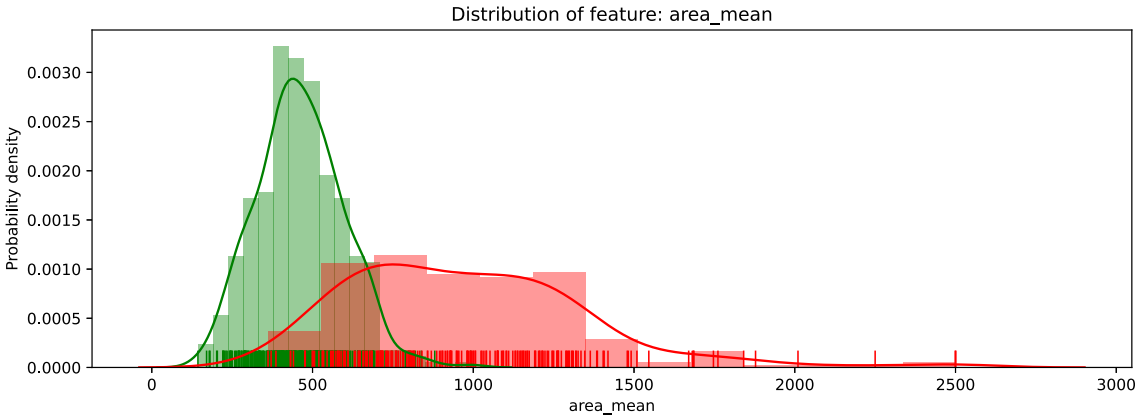
```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns

features = df.iloc[:,2:].columns
plt.suptitle('Title')
for i, f in enumerate(df[features]):
    plt.figure(figsize=(12, 4))
    ax = plt.subplot(1, 1, 1)
    sns.distplot(df[f][df.diagnosis == 'B'], rug=True, color='green')
    sns.distplot(df[f][df.diagnosis == 'M'], rug=True, color='red')
    ax.set_xlabel(str(f))
    ax.set_ylabel('Probability density')
    ax.set_title('Distribution of feature: ' + str(f))
plt.show()
```

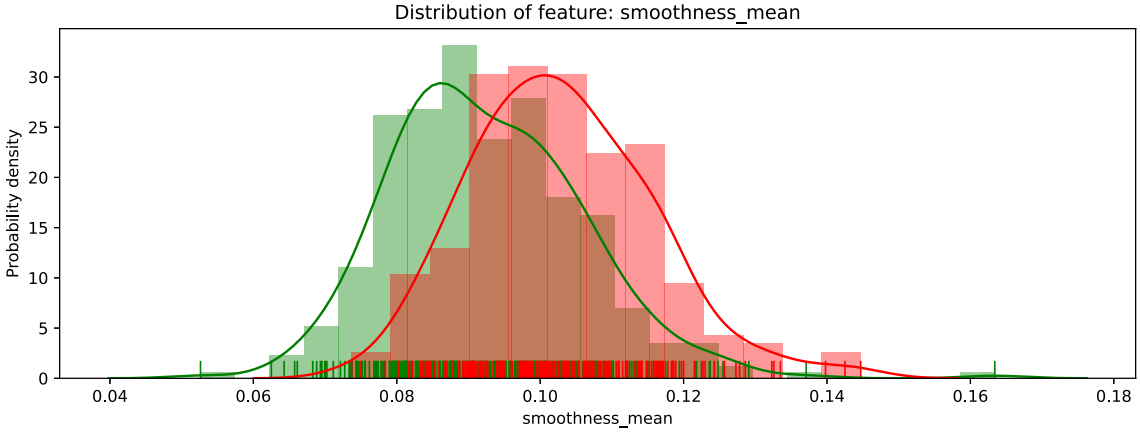
```
<Figure size 432x288 with 0 Axes>
```

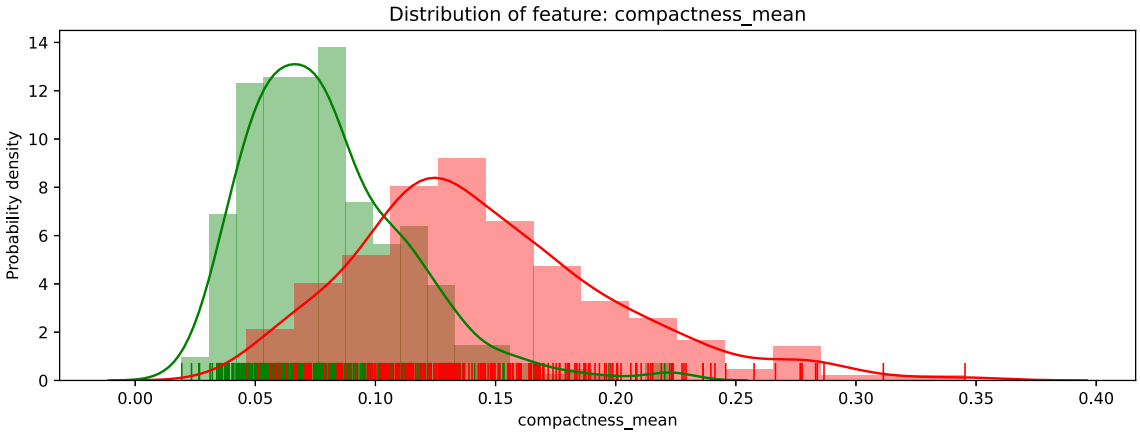


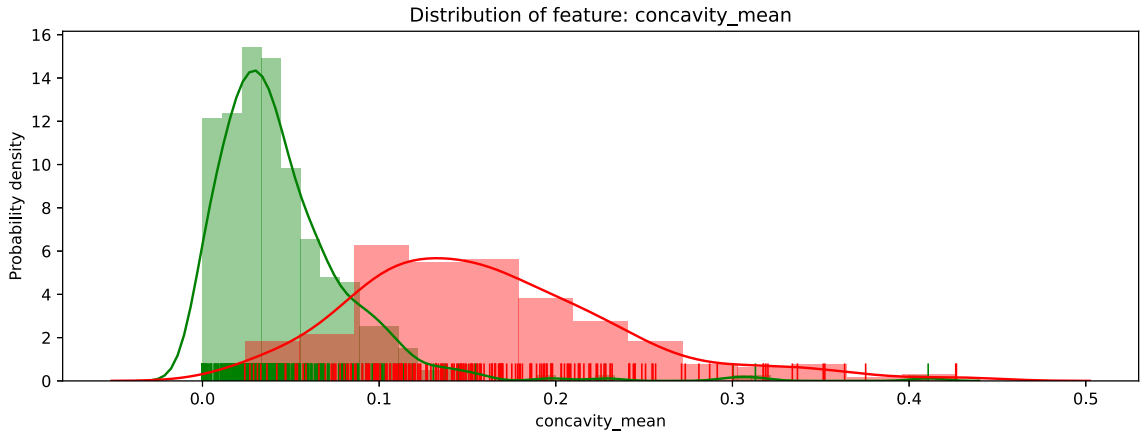



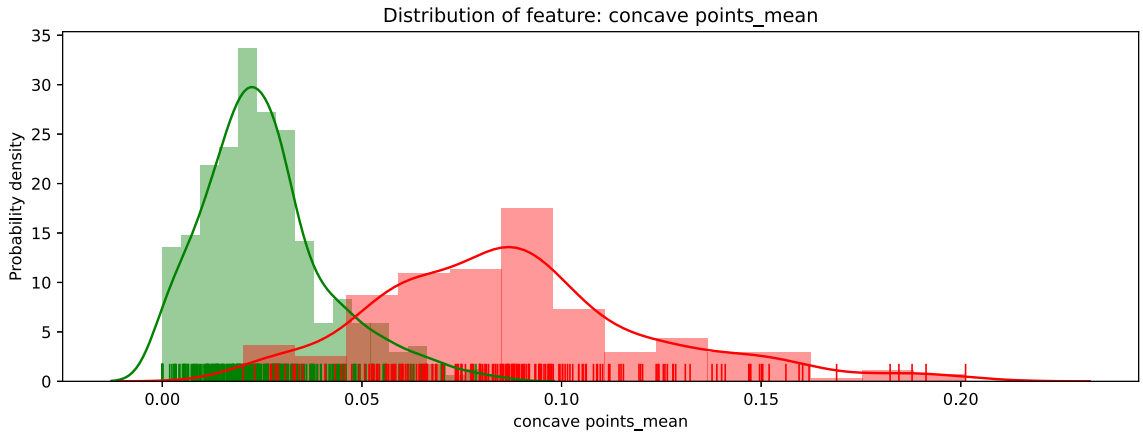



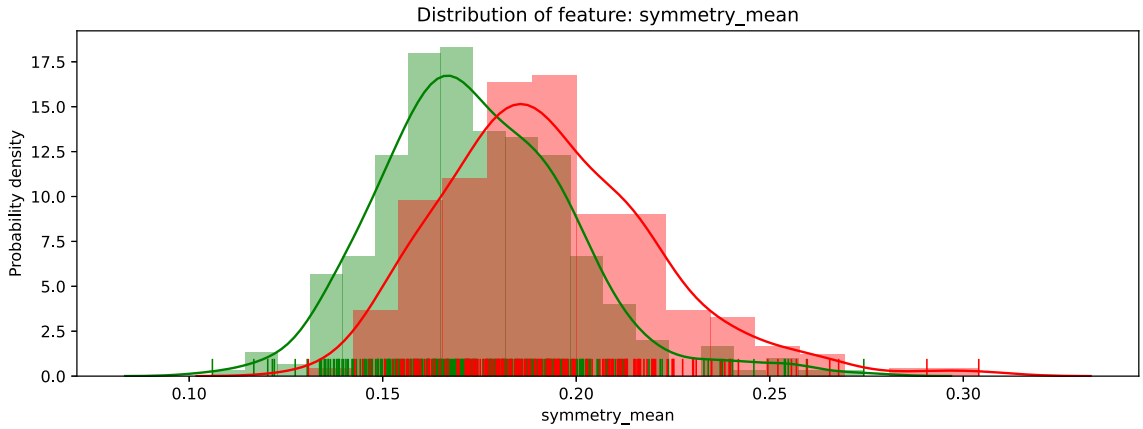

Distribution of feature: area_mean
Probability density
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
2500
3000
area_mean

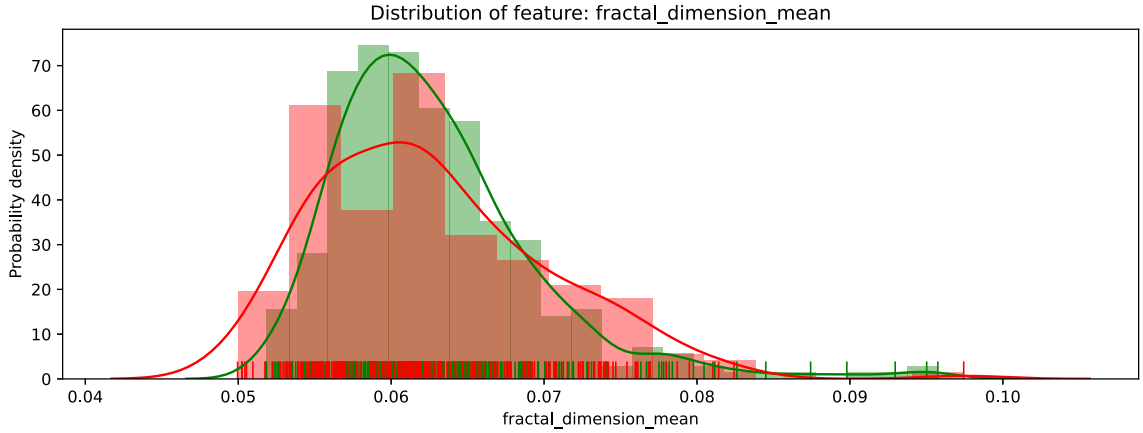

Distribution of feature: smoothness_mean
Probability density
30
25
20
15
10
5
0
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
smoothness_mean

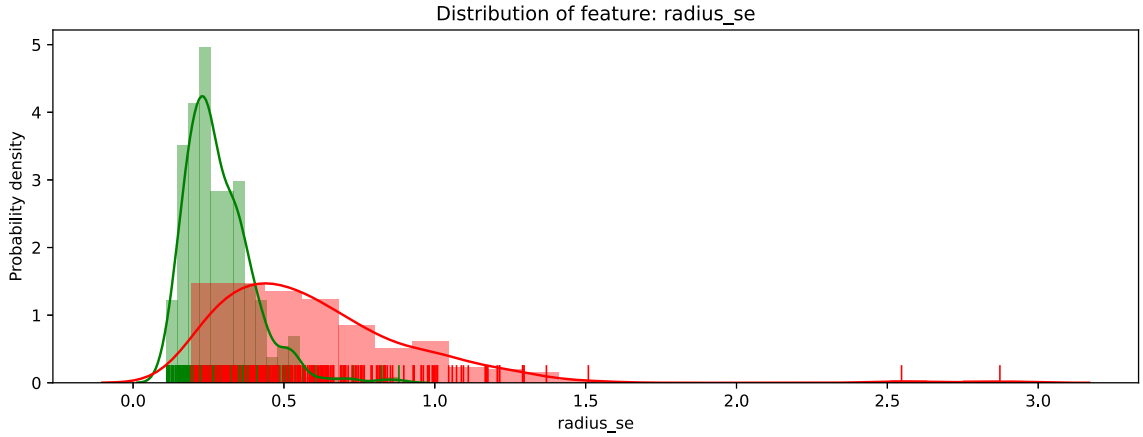

Distribution of feature: compactness_mean
Probability density
14
12
10
8
6
4
2
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
compactness_mean

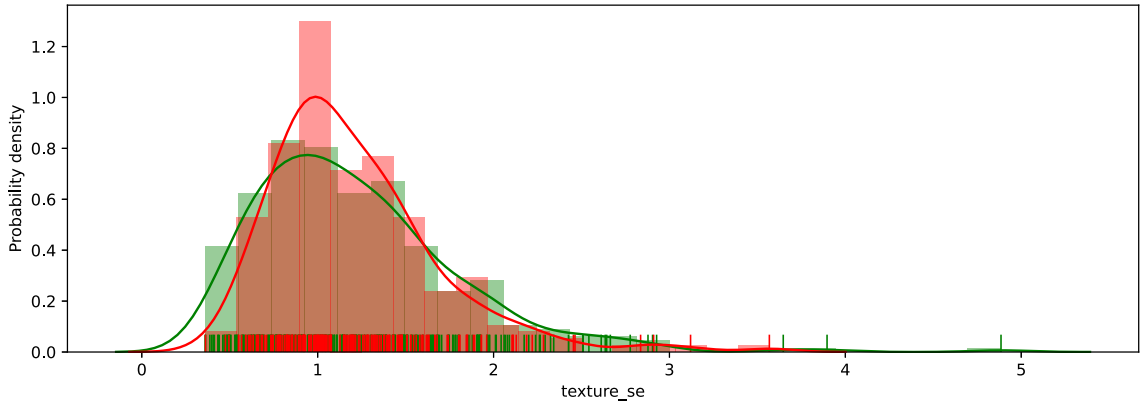

Distribution of feature: concavity_mean
Probability density
concavity_mean

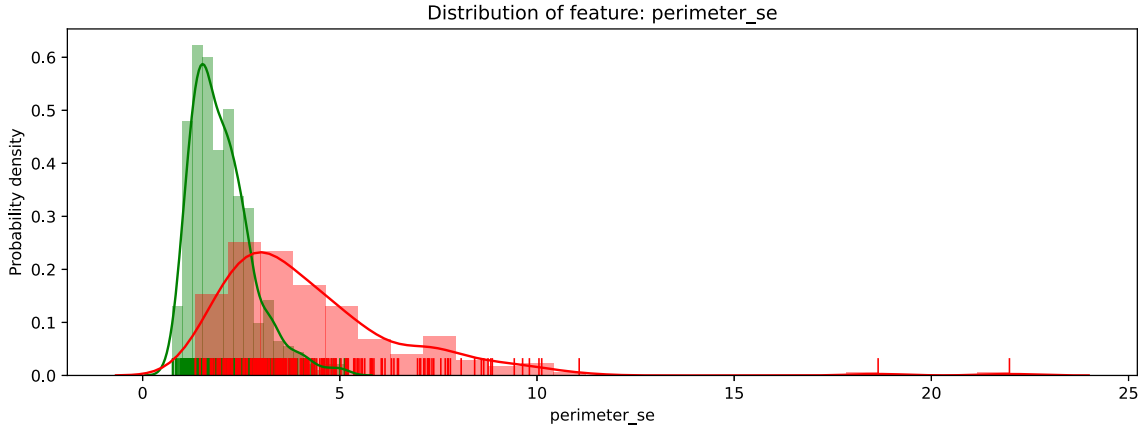

Distribution of feature: concave points_mean
Probability density
concave points_mean

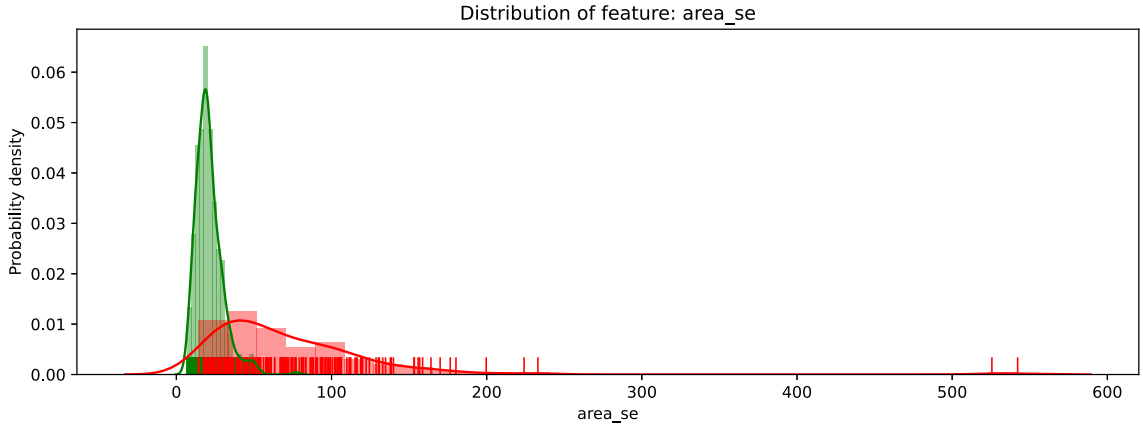

Distribution of feature: symmetry_mean
Probability density
symmetry_mean

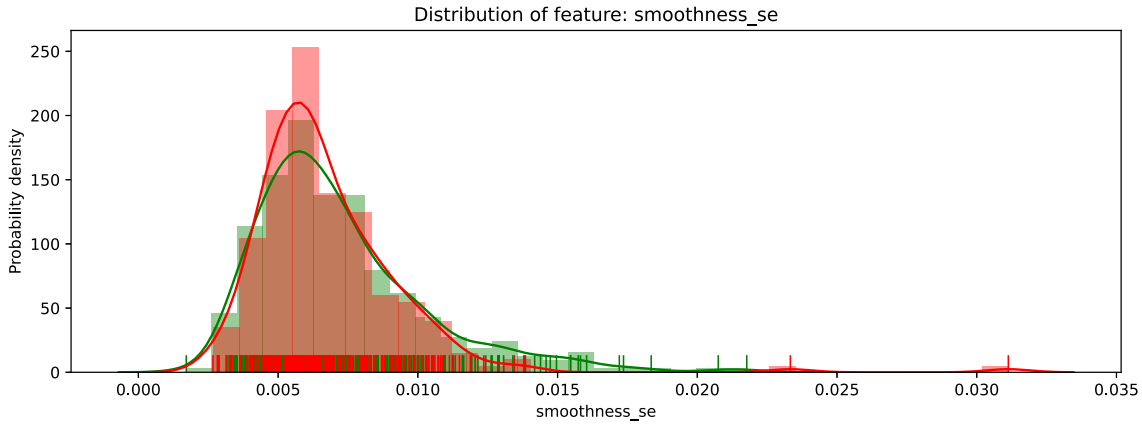

Distribution of feature: fractal_dimension_mean
Probability density
fractal_dimension_mean
0
10
20
30
40
50
60
70
0.04
0.05
0.06
0.07
0.08
0.09
0.10

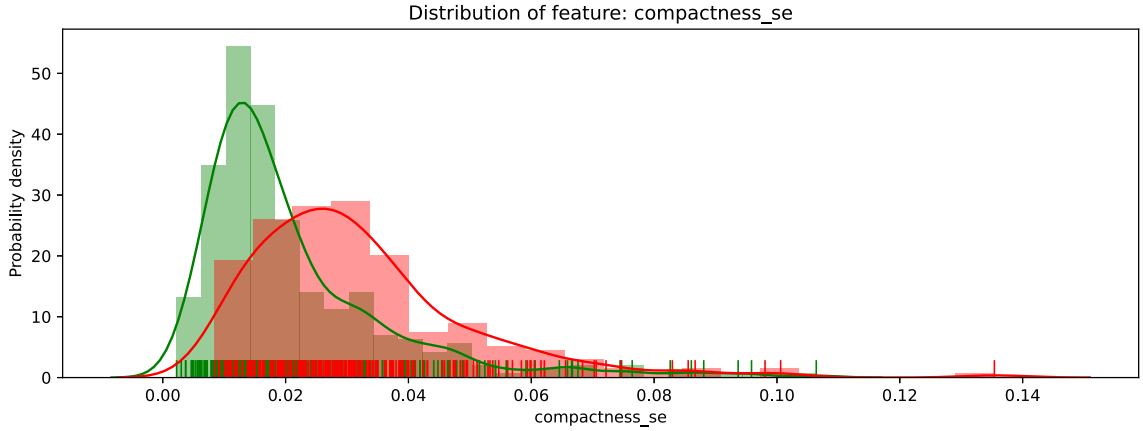

Distribution of feature: radius_se
Probability density
radius_se
0
1
2
3
4
5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

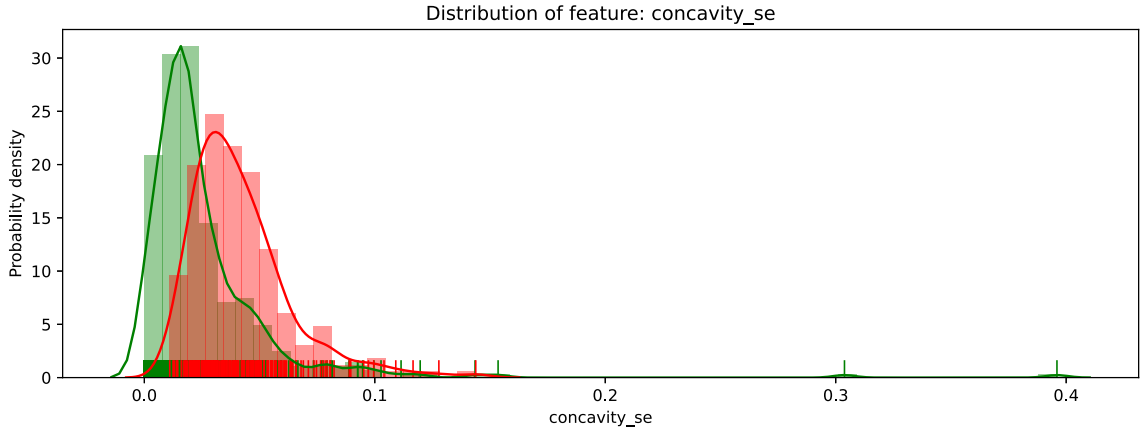

Distribution of feature: texture_se
Probability density
texture_se
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4
5

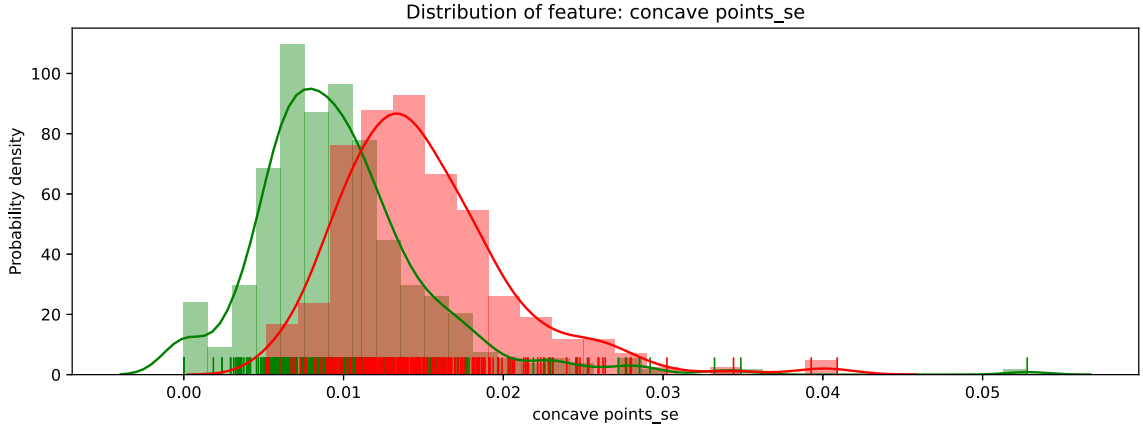

Distribution of feature: perimeter_se
Probability density
perimeter_se

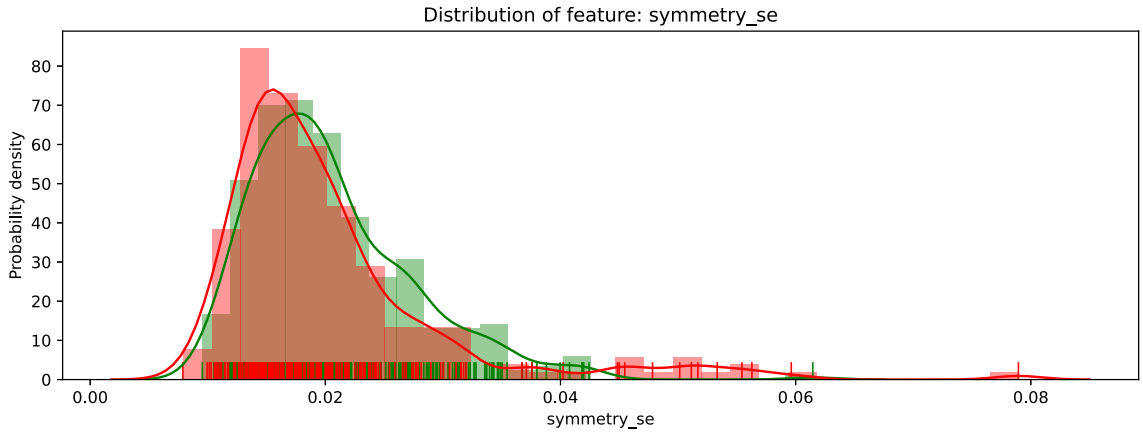

Distribution of feature: area_se
Probability density
area_se

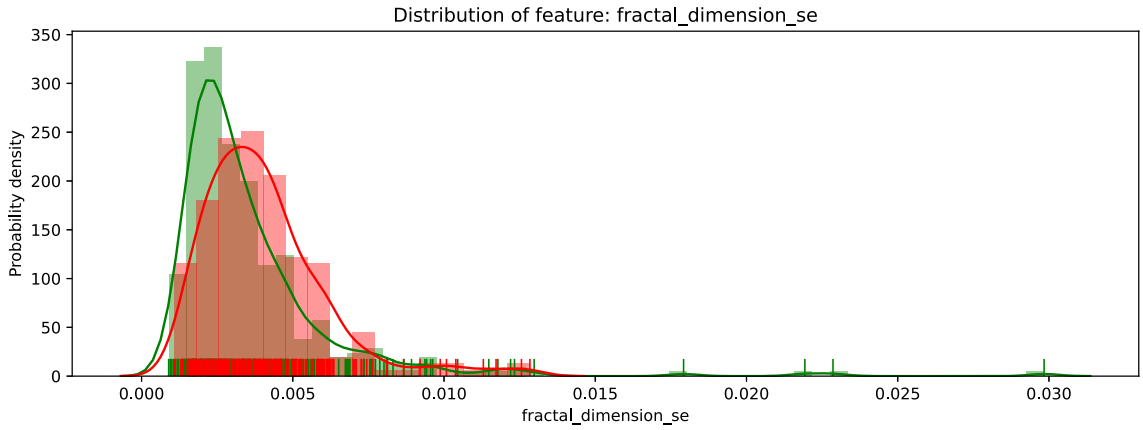

Distribution of feature: smoothness_se
Probability density
smoothness_se

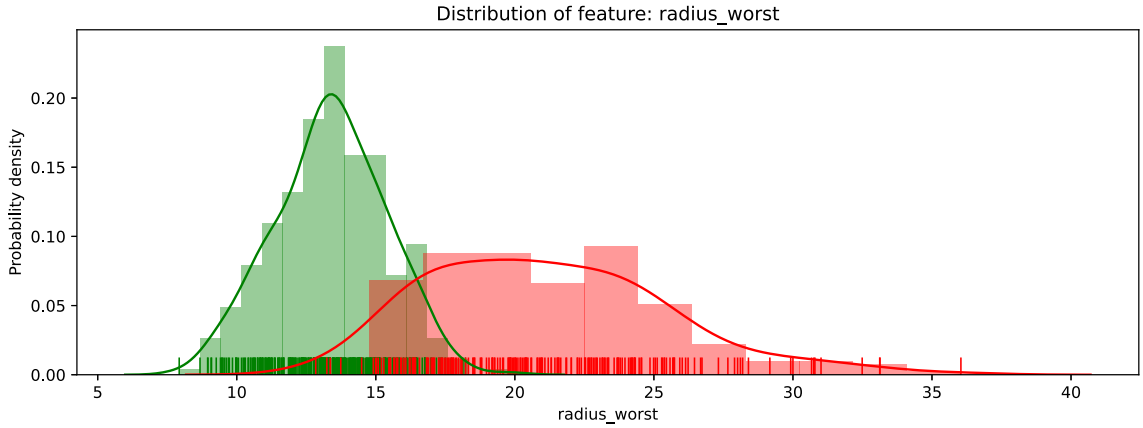

Distribution of feature: compactness_se
Probability density
compactness_se
0
10
20
30
40
50
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

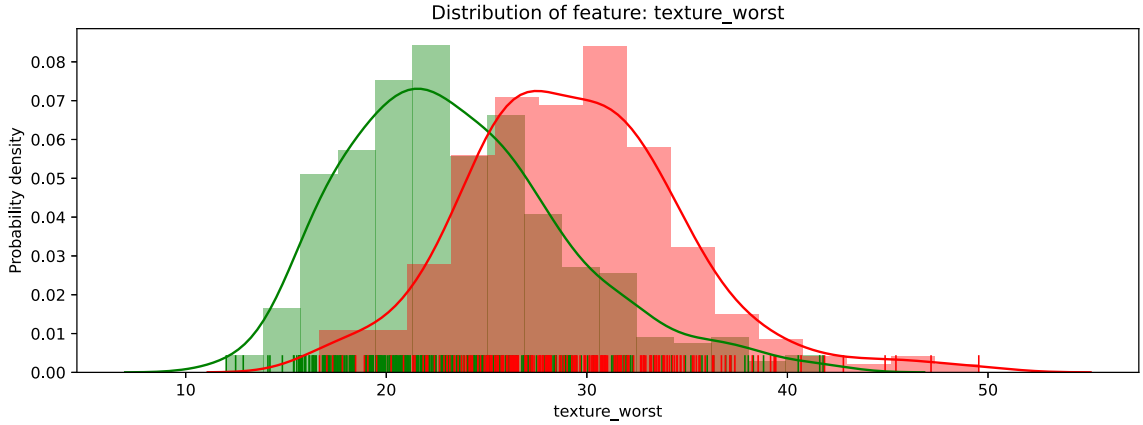

Distribution of feature: concavity_se
Probability density
concavity_se
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4

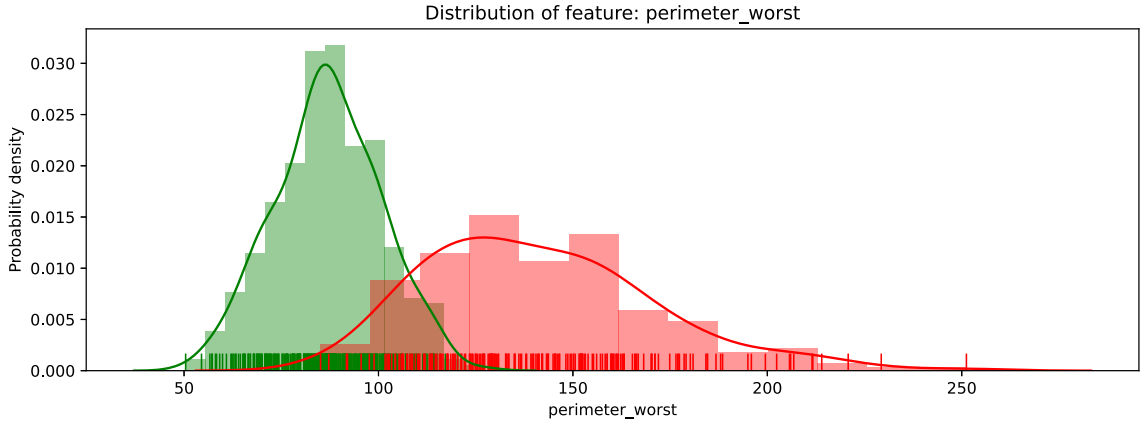

Distribution of feature: concave points_se
Probability density
concave points_se
0
20
40
60
80
100
0.00
0.01
0.02
0.03
0.04
0.05

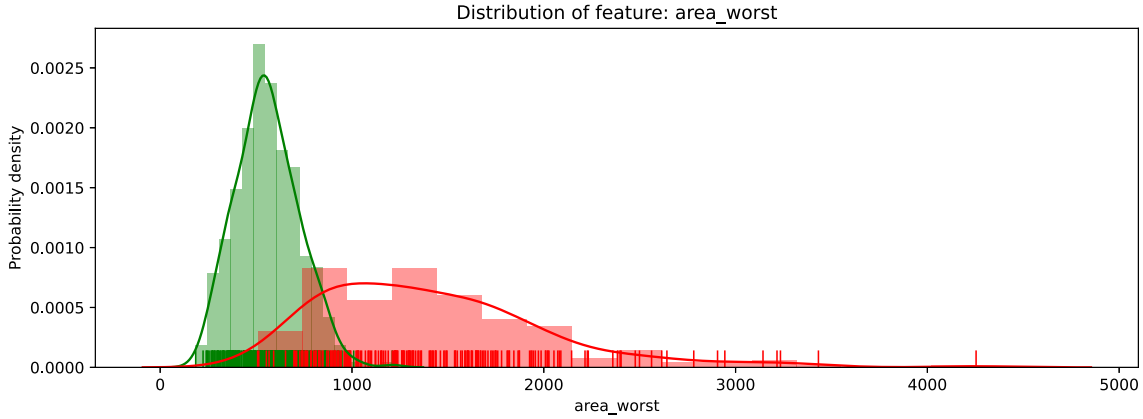

Distribution of feature: symmetry_se
Probability density
0
10
20
30
40
50
60
70
80
0.00
0.02
0.04
0.06
0.08
symmetry_se

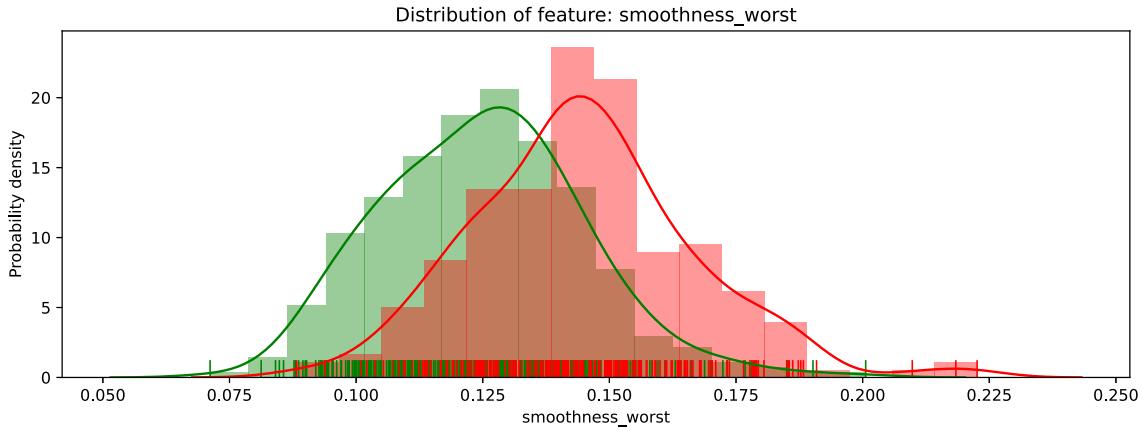

Distribution of feature: fractal_dimension_se
Probability density
0
50
100
150
200
250
300
350
0.000
0.005
0.010
0.015
0.020
0.025
0.030
fractal_dimension_se

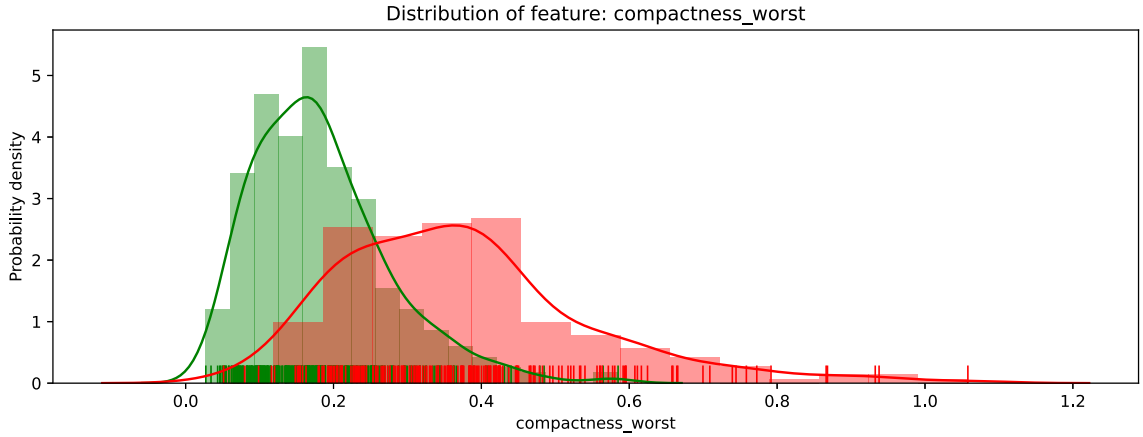

Distribution of feature: radius_worst
Probability density
0.00
0.05
0.10
0.15
0.20
5
10
15
20
25
30
35
40
radius_worst

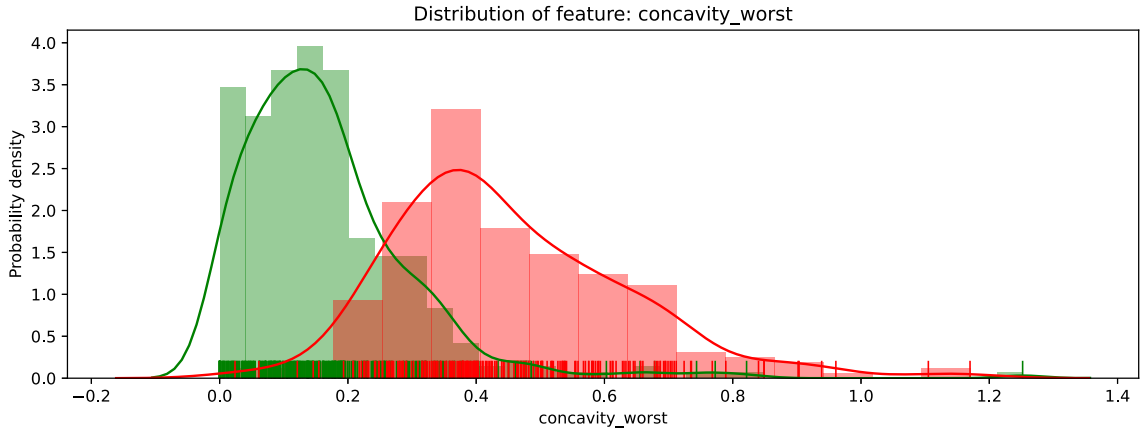

Distribution of feature: texture_worst
Probability density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
10
20
30
40
50
texture_worst

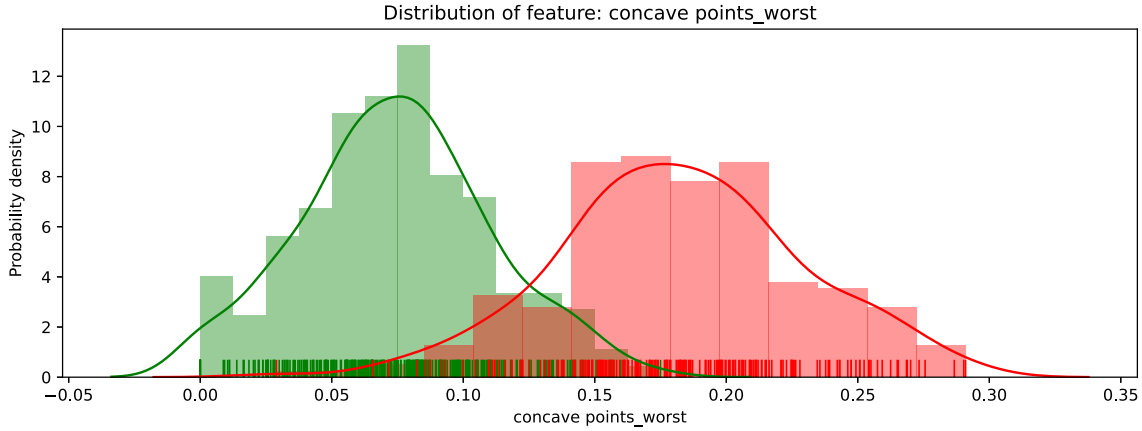

Distribution of feature: perimeter_worst
Probability density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
50
100
150
200
250
perimeter_worst

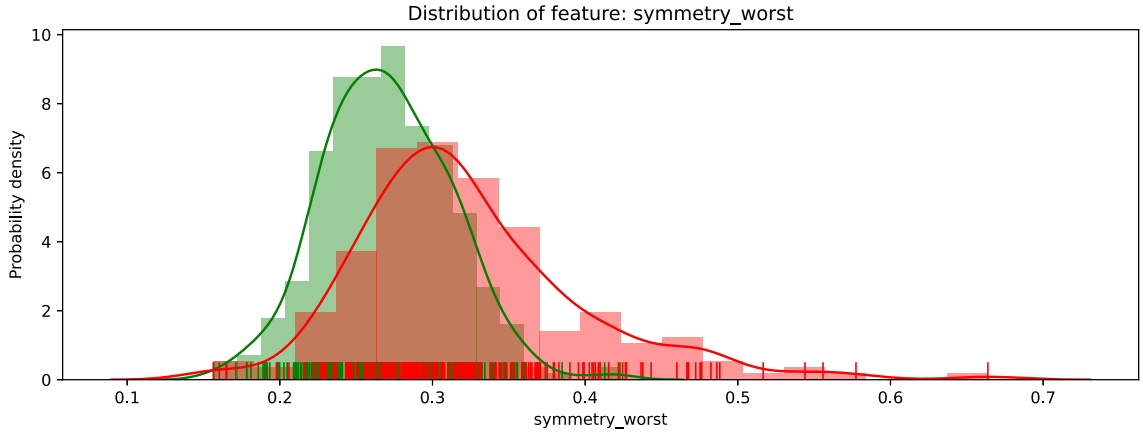

Distribution of feature: area_worst
Probability density
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
1000
2000
3000
4000
5000
area_worst

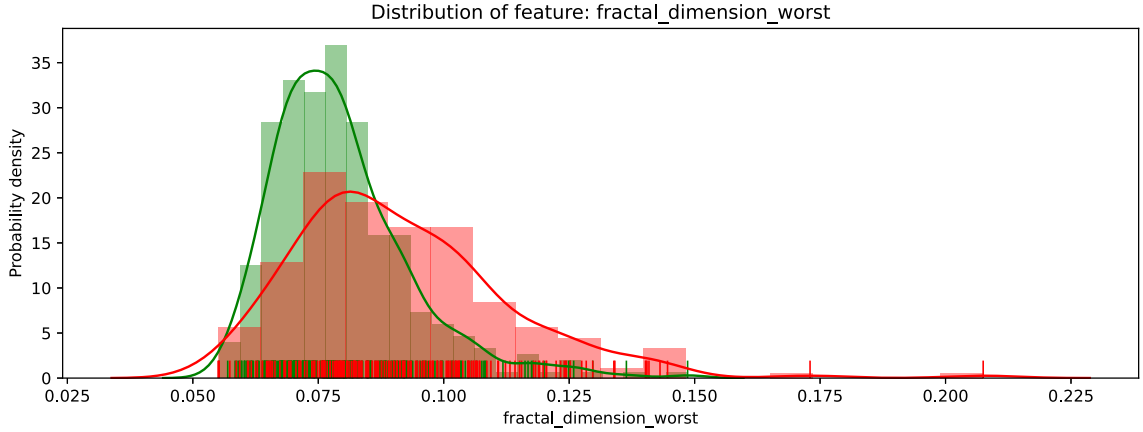

Distribution of feature: smoothness_worst
Probability density
0
5
10
15
20
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
0.250
smoothness_worst


Distribution of feature: compactness_worst
Probability density
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0
1.2
compactness_worst


Distribution of feature: concavity_worst
Probability density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
concavity_worst


Distribution of feature: concave points_worst
Probability density
concave points_worst


Distribution of feature: symmetry_worst
Probability density
symmetry_worst

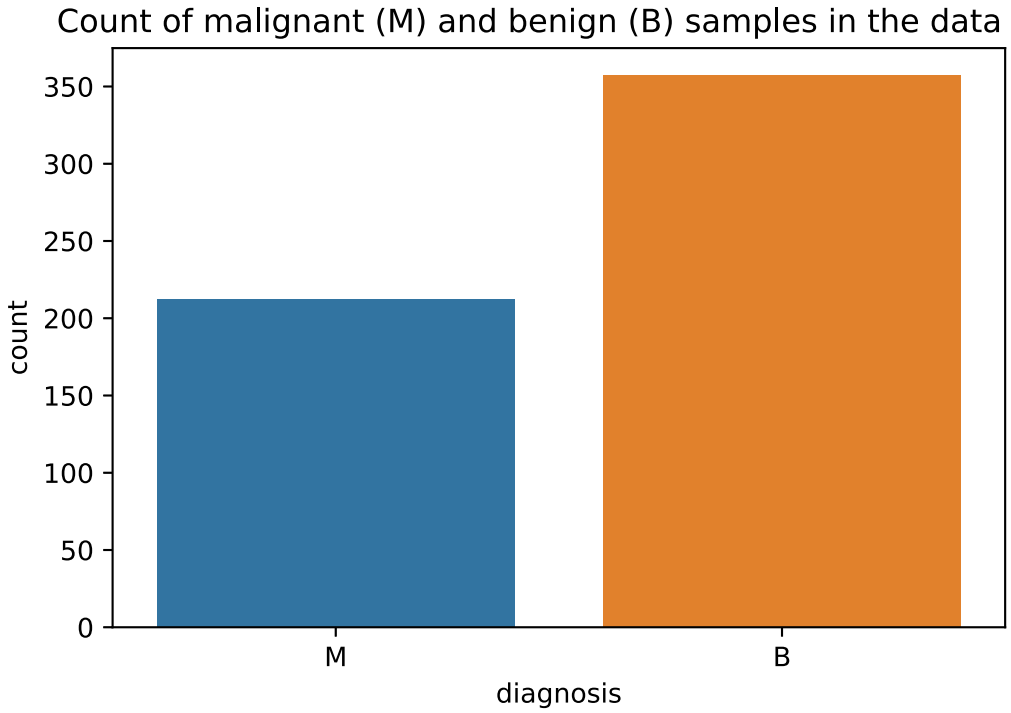

Distribution of feature: fractal_dimension_worst
Probability density
fractal_dimension_worst

In [10]:

```
ax = sns.countplot(df['diagnosis'], label = "Occurrences")
plt.title("Count of malignant (M) and benign (B) samples in the data set")
```

Out[10]:

```
Text(0.5, 1.0, 'Count of malignant (M) and benign (B) samples in the data
set')
```



In [11]:

```
X = df[features]
X
```

Out[11]:

| | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactne |
|---|---|---|---|---|---|---|
| **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 | |

569 rows × 30 columns

In [12]:

```
Y = df['diagnosis']
```

In [13]:

```
X = StandardScaler().fit_transform(X)
X
```

Out[13]:

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=
7383)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
381
188
381
188
```

# Application of basic machine learning algorithms

In [15]:

```
scores = []
for i in range(2, 20):
    classifier = DecisionTreeClassifier(max_leaf_nodes=i, random_state=123)
    predictions = classifier.fit(X_train, y_train).predict(X_test)
    score = accuracy_score(y_true = y_test, y_pred = predictions)
    scores.append(dict(score=score, classifier=classifier))
    print('With ' + str(i) + 'nodes is score: ' + str(score))

max_score = max(s['score'] for s in scores)
best_classifier = next(s for s in scores if s['score'] == max_score)['classifier']
print('Feature importances:')
imp_dict = zip (features, best_classifier.feature_importances_)
imp_dict = sorted(imp_dict, key = lambda t: t[1])
for f, i in imp_dict:
    print(f + ': ' + str(i))
print('Max score: ' + str(max_score))
```

```
With 2nodes is score: 0.9095744680851063
With 3nodes is score: 0.925531914893617
With 4nodes is score: 0.9414893617021277
With 5nodes is score: 0.9361702127659575
With 6nodes is score: 0.9361702127659575
With 7nodes is score: 0.9361702127659575
With 8nodes is score: 0.9361702127659575
With 9nodes is score: 0.9308510638297872
With 10nodes is score: 0.925531914893617
With 11nodes is score: 0.925531914893617
With 12nodes is score: 0.925531914893617
With 13nodes is score: 0.925531914893617
With 14nodes is score: 0.925531914893617
With 15nodes is score: 0.925531914893617
With 16nodes is score: 0.9202127659574468
With 17nodes is score: 0.9042553191489362
With 18nodes is score: 0.9042553191489362
With 19nodes is score: 0.9042553191489362
Feature importances:
radius_mean: 0.0
texture_mean: 0.0
perimeter_mean: 0.0
area_mean: 0.0
smoothness_mean: 0.0
compactness_mean: 0.0
concavity_mean: 0.0
concave points_mean: 0.0
symmetry_mean: 0.0
fractal_dimension_mean: 0.0
radius_se: 0.0
texture_se: 0.0
perimeter_se: 0.0
area_se: 0.0
smoothness_se: 0.0
compactness_se: 0.0
concavity_se: 0.0
concave points_se: 0.0
symmetry_se: 0.0
fractal_dimension_se: 0.0
radius_worst: 0.0
perimeter_worst: 0.0
smoothness_worst: 0.0
compactness_worst: 0.0
concavity_worst: 0.0
symmetry_worst: 0.0
fractal_dimension_worst: 0.0
texture_worst: 0.06198140435134387
concave points_worst: 0.12598114750037204
area_worst: 0.8120374481482842
Max score: 0.9414893617021277
```

In [16]:

```
scores = []
for i in range(2, 20):
    classifier = RandomForestClassifier(max_leaf_nodes=i)
    predictions = classifier.fit(X_train, y_train).predict(X_test)
    score = accuracy_score(y_true = y_test, y_pred = predictions)
    scores.append(dict(score=score, classifier=classifier))
    print('With ' + str(i) + 'nodes is score: ' + str(score))

max_score = max(s['score'] for s in scores)
best_classifier = next(s for s in scores if s['score'] == max_score)['classifier']
print('Feature importances:')
imp_dict = zip (features, best_classifier.feature_importances_)
imp_dict = sorted(imp_dict, key = lambda t: t[1])
for f, i in imp_dict:
    print(f + ': ' + str(i))
print('Max score: ' + str(max_score))
rfc = best_classifier
```

```
With 2nodes is score: 0.9202127659574468
With 3nodes is score: 0.9414893617021277
With 4nodes is score: 0.9521276595744681
With 5nodes is score: 0.9414893617021277
With 6nodes is score: 0.9627659574468085
With 7nodes is score: 0.9574468085106383
With 8nodes is score: 0.9521276595744681
With 9nodes is score: 0.9574468085106383
With 10nodes is score: 0.9627659574468085
With 11nodes is score: 0.9627659574468085
With 12nodes is score: 0.9627659574468085
With 13nodes is score: 0.9680851063829787
With 14nodes is score: 0.9627659574468085
With 15nodes is score: 0.9680851063829787
With 16nodes is score: 0.9627659574468085
With 17nodes is score: 0.9680851063829787
With 18nodes is score: 0.9627659574468085
With 19nodes is score: 0.9627659574468085
Feature importances:
symmetry_se: 0.0029504800389258887
smoothness_mean: 0.0030968062684194617
compactness_se: 0.00328462254770653
fractal_dimension_worst: 0.003439161756059001
fractal_dimension_mean: 0.0038299257456068402
concavity_se: 0.0038422706962367623
texture_se: 0.0040181991191703004
concave points_se: 0.004621914551159907
smoothness_se: 0.004916791050718517
fractal_dimension_se: 0.004994991931322467
symmetry_mean: 0.005059165720012492
compactness_mean: 0.007553693867938604
smoothness_worst: 0.007623544755808637
symmetry_worst: 0.007941398442551586
compactness_worst: 0.0117430899405953
radius_se: 0.012089140339998286
texture_mean: 0.013432121420938196
texture_worst: 0.015263839515334438
perimeter_se: 0.015596444622860495
concavity_worst: 0.025311817377956597
area_mean: 0.0306070729403948
concave points_mean: 0.044910959648023364
perimeter_mean: 0.05515154488563485
radius_mean: 0.058659120273257113
area_se: 0.05955916991080493
concavity_mean: 0.06595506328161753
concave points_worst: 0.12213140759542765
perimeter_worst: 0.12836355178700246
area_worst: 0.13539900343410682
radius_worst: 0.13865368653441035
Max score: 0.9680851063829787
```

In [17]:

```
classifier = GaussianNB()
predictions = classifier.fit(X_train, y_train).predict(X_test)
score = accuracy_score(y_true = y_test, y_pred = predictions)
print(str(score))
from sklearn.inspection import permutation_importance
imps = permutation_importance(classifier, X_test, y_test)
print(imps.importances_mean)
```

```
0.9468085106382979
[ 6.38297872e-03  1.06382979e-03  7.44680851e-03  1.06382979e-02
  0.00000000e+00 -2.22044605e-17 -3.19148936e-03  9.57446809e-03
  0.00000000e+00  0.00000000e+00  1.27659574e-02  0.00000000e+00
  1.17021277e-02  5.31914894e-02 -7.44680851e-03 -4.25531915e-03
 -1.06382979e-03 -1.06382979e-03  0.00000000e+00 -2.22044605e-17
  8.51063830e-03  1.06382979e-03  6.38297872e-03  2.76595745e-02
  0.00000000e+00 -4.25531915e-03 -3.19148936e-03  6.38297872e-03
  0.00000000e+00 -4.25531915e-03]
```

In [18]:

```
classifier = KNeighborsClassifier()
predictions = classifier.fit(X_train, y_train).predict(X_test)
score = accuracy_score(y_true = y_test, y_pred = predictions)
print(str(score))
```

```
0.9680851063829787
```

# Comparison of the accuracy the used basic algorithms with more sophisticated machine learning algorithms

## Basic algorithms examined in this project

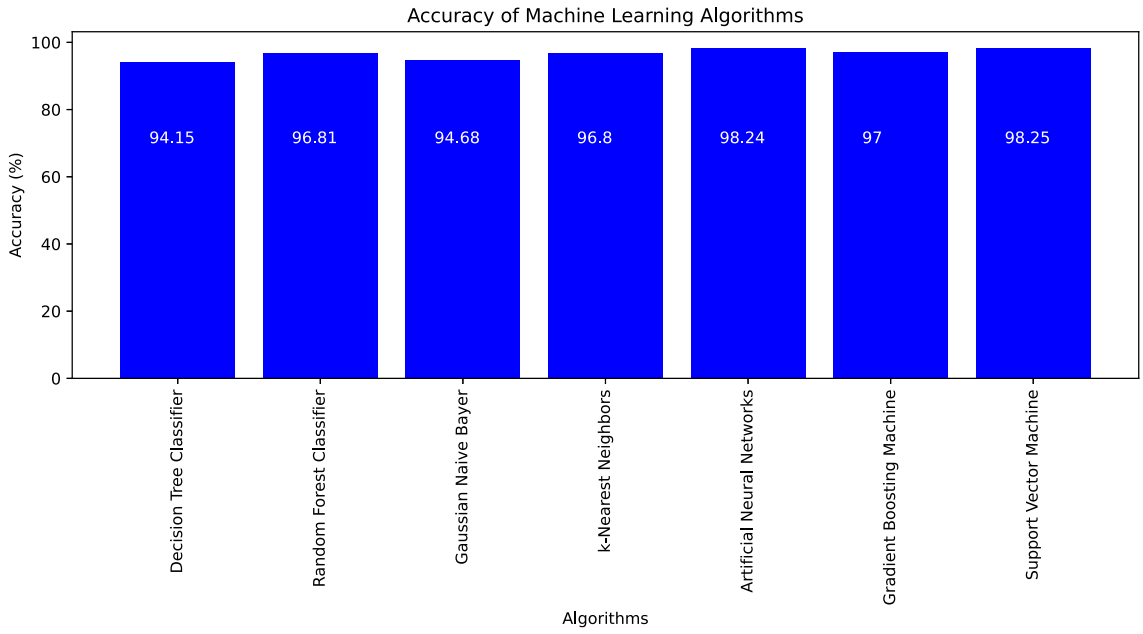
- Decision Tree Classifier (DTC): 94.15%
- Random Forest Classifier (RFC): 96.81%
- Gaussian Naive Bayes (GNB): 94.68%
- k-Nearest Neighbors (KNN): 96,80% ### Advanced algorithms examined by others
- Artificial Neural Networks (ANN): 98.24% https://www.kaggle.com/thebrownviking20/intro-to-keras-with-breast-cancer-data-ann (https://www.kaggle.com/thebrownviking20/intro-to-keras-with-breast-cancer-data-ann)
- Gradient Boosting Machine (GBM): 97% https://www.kaggle.com/gpreda/breast-cancer-prediction-from-cytopathology-data (https://www.kaggle.com/gpreda/breast-cancer-prediction-from-cytopathology-data)
- Support Vector Machine (SVM): 98.25% https://www.kaggle.com/faressayah/support-vector-machine-pca-tutorial-for-beginner (https://www.kaggle.com/faressayah/support-vector-machine-pca-tutorial-for-beginner)

In [19]:

```
algorithm_accuracy = {
    'Decision Tree Classifier': 94.15,
    'Random Forest Classifier': 96.81,
    'Gaussian Naive Bayer': 94.68,
    'k-Nearest Neighbors': 96.80,
    'Artificial Neural Networks': 98.24,
    'Gradient Boosting Machine': 97,
    'Support Vector Machine': 98.25
}

plt.figure(figsize=(12, 4))

for index, value in enumerate(algorithm_accuracy.values()):
    plt.text(index - 0.2, 70, str(value), color='white')

plt.title('Accuracy of Machine Learning Algorithms')
plt.xticks(rotation=90)
plt.xlabel('Algorithms')
plt.ylabel('Accuracy (%)')
plt.bar(algorithm_accuracy.keys(), algorithm_accuracy.values(), color='b')
```

Out[19]:

```
<BarContainer object of 7 artists>
```



# Most important features

Looking at the distribution graphs, the following features appeared as the most important ones to me for the classification problem because they have the smallest overlapping areas for benign and malignant samples:

- cancavity_mean
- concave points_mean
- radius_worst
- perimeter_worst
- area_worst
- cancavity_worst
- concave points_worst

This is how the classifiers weighed the feature importance: The Decision Tree Classifier found only three features being important for classification and ignored the other features:

- texture_worst: 0.06198140435134387
- concave points_worst: 0.12598114750037204
- area_worst: 0.8120374481482842

The Random Forest Classifier weighed these features as most important:

- radius_mean: 0.05872515489464252
- area_mean: 0.06469903094642988
- perimeter_mean: 0.0665823176981204
- concave points_worst: 0.09296126521373517
- radius_worst: 0.11300785482788624
- perimeter_worst: 0.12171853874149956
- area_worst: 0.15396899594687236

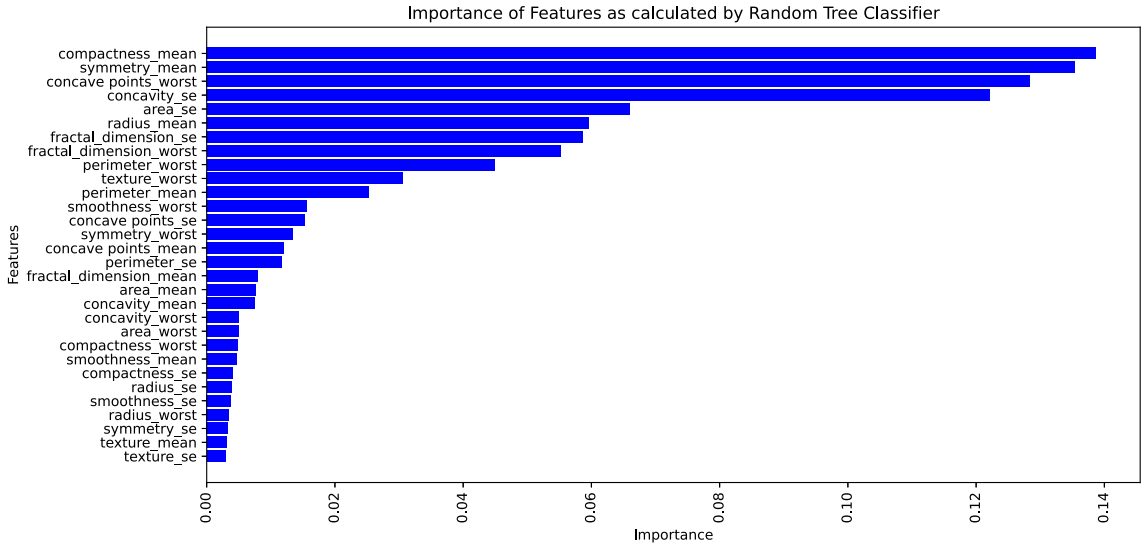
The other two classification algorithms have no concept of feature importance

In [27]:

```
imp_dict = zip (features, rfc.feature_importances_)
imp_list = sorted(imp_dict, key = lambda t:(t[1], t[0]))
features = []
importances = []
for i in imp_list:
   features += [i[0]]
   importances += [i[1]]
plt.figure(figsize=(12, 6))

# for index, value in enumerate(imp_dict_sorted.values()):
#     plt.text(index - 0.2, 70, str(value), color='white')

plt.title('Importance of Features as calculated by Random Tree Classifier')
plt.xticks(rotation=90)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.barh(features, importances, color='b')
```

Out[27]:

```
<BarContainer object of 30 artists>
```




In [ ]: