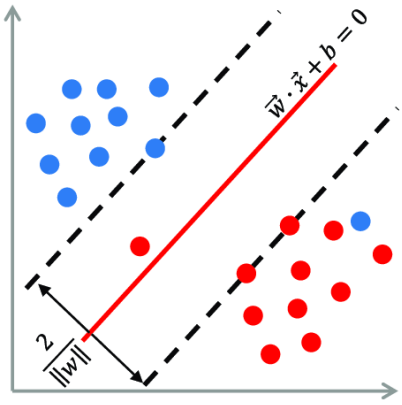
Este notebook foi criado por [Adrielson Ferreira Justino]

Para qualquer feedback, erro ou sugestão, ele pode ser contatado por e-mail (adrielferreira28@gmail.com (mailto:adrielferreira28@gmail.com)), GIT (https://github.com/Adrielson) ou LinkedIn (https://www.linkedin.com/in/adrielson-justino).

Support Vector Machines (SVM)



O que é SVM?

- **Máquinas de vetores de suporte** é um algoritmos de aprendizado supervisionado;
- Usado para **regressão, classificação e detecção de outliers**;
- SVMs são notavelmente um dos modelos poderosos no aprendizado de máquina clássico;
- São adequados para lidar com conjuntos de dados complexos e de alta dimensão.

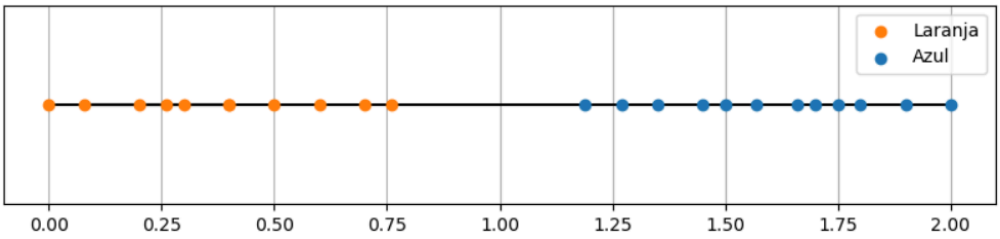
Alguns exemplos de aplicações do algoritmo SVM:

- Detecção de spam
- Analise de Sentimento (PLN)
- **Diagnóstico médico**
- Reconhecimento facial
- Detecção de fraude
- Classificação de imagens
- Regressão

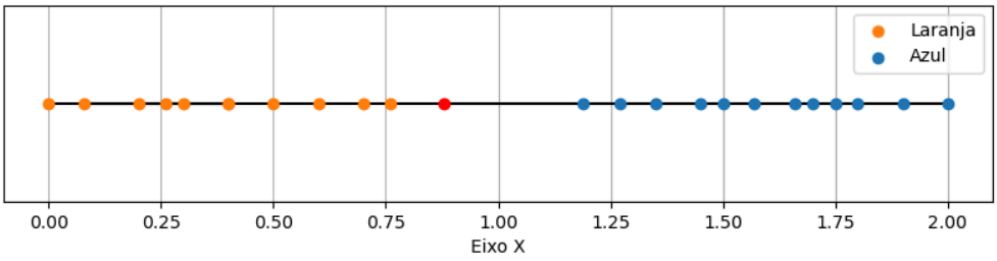
Quais tipos de dados?

- O SVM pode lidar com diferentes tipos de conjuntos de dados;
- Tanto **lineares quanto não lineares**;
- Isso é possível pois o SVM **suporta diferentes kernels**:
- *Linear,*
- *polinomial,*
- *Radial Basis Function (rbf),*
- *sigmóide.*

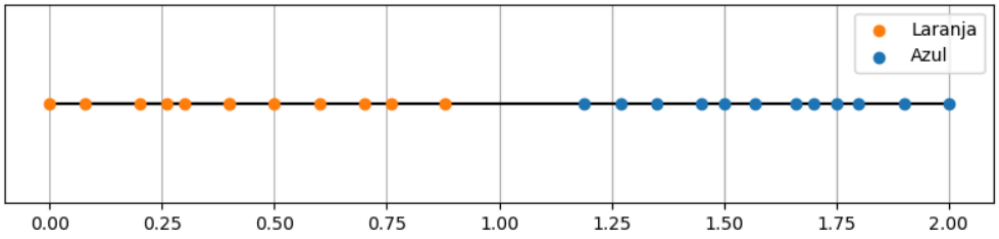
Primeiro vamos supor que temos um conjunto de pontos como do gráfico abaixo.



No gráfico abaixo entre os pontos laranjas e azuis temos um ponto vermelho.



Qual seria a classificação deste ponto? Laranja ou azul?



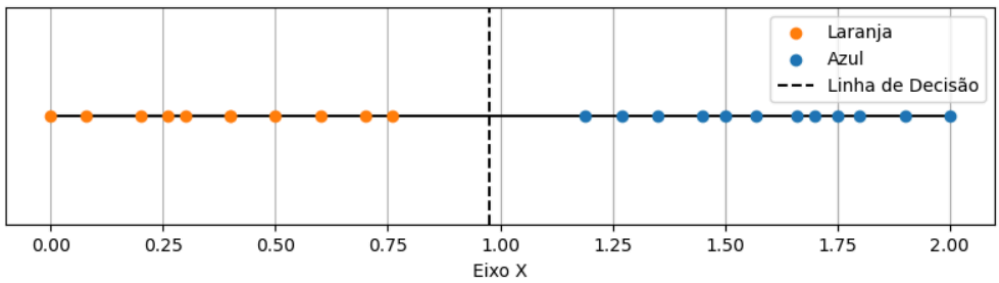
Intuitivamente, pela proximidade do ponto vermelho com os pontos laranjas é muito provável a classificação desse ponto é **laranja**.

Essa lógica é muito parecida com a utilizada no SVM.

- Podemos basicamente pegar os pontos que estão mais no extremo dos dados;
- **Os pontos de duas classes diferentes que estão mais próximos entre si**;
- Usar esses pontos para determinar um **hiperplano** capaz de separar essas classes.

A pergunta que queremos responder é:

- Qual seria o melhor lugar entre os pontos para colocar o nosso **limite de separação**?

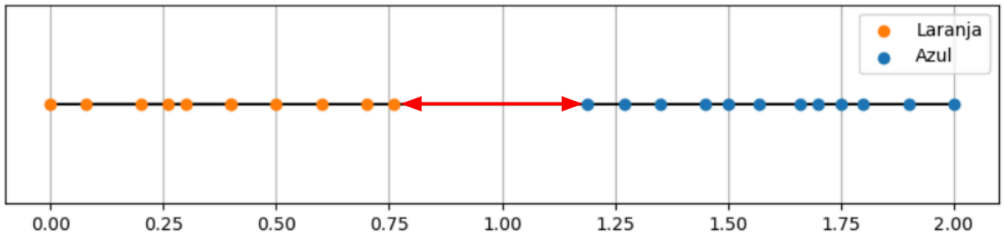


- Este lugar irá indicar:

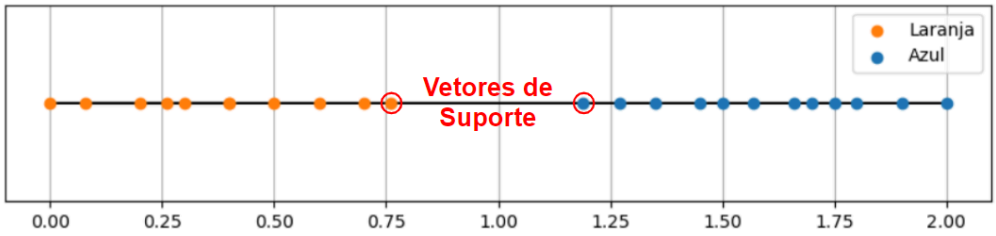
- Se o ponto for menor do que o valor do limite de decisão então -**classificado como laranja;**
- Se for maior - **classificado como azul.**
- Nesta escala nós conseguimos perceber visualmente que talvez este local ou ponto seja o 0,975.

Conceitualmente o que tem por trás disso que estamos fazendo?

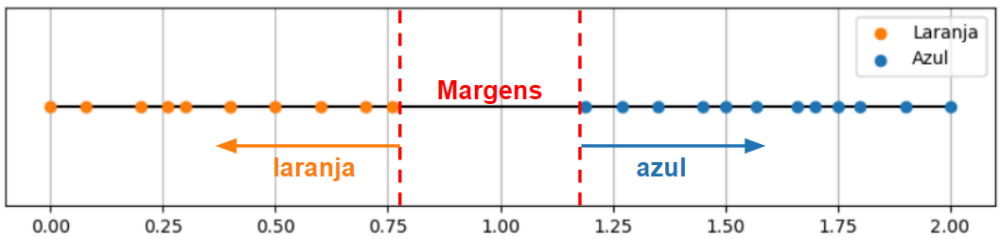
- O que o algoritmo vai fazer é pegar os pontos que estão mais extremos;
- Os pontos que pertencem a classes diferentes e que tenham a **menor distância** entre eles:



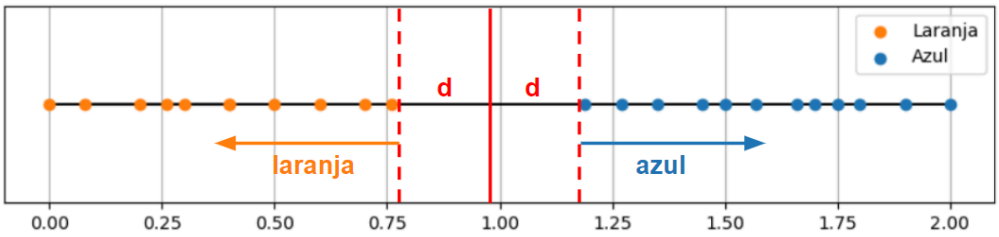
A estes pontos damos o nome de **vetores de suporte**



- Os vetores de suporte servem para traçarmos as **margens**

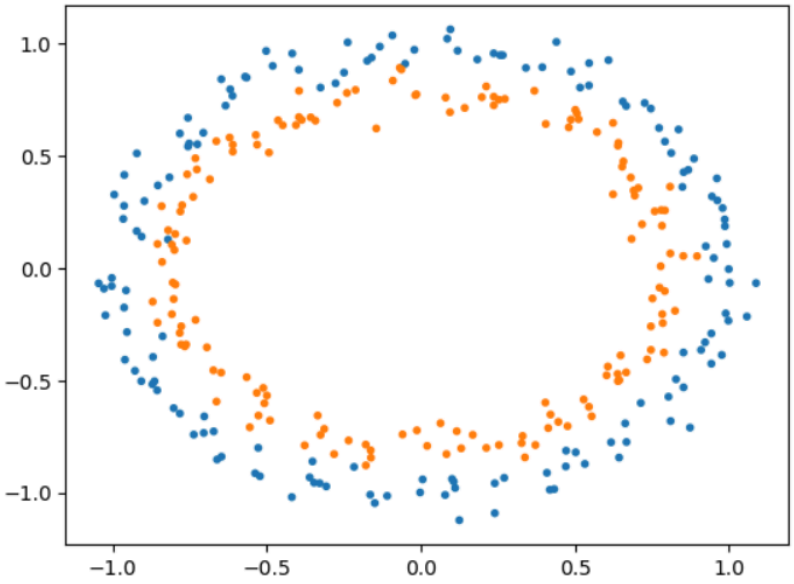


- As margens vão definir qual será o nosso **hiperplano de separação.**
- Quanto maior a margem, melhor!
- **OBJETIVO: maximizar a distância entre as margens.**



Dados não lineares

- Um fato importante é que ele pode ser utilizado tanto em dados **linearmente separáveis** como em dados que **não são linearmente separáveis**



- Quando nós falamos linearmente separados não estamos falando somente de uma **reta**.
- Podemos estar falando de uma **reta** quando temos **duas dimensões**.
- Falando de um **plano** quando temos **três dimensões**.
- **Hiperplanos** quando temos **mais de três dimensões** em que não temos uma capacidade de visualização muito clara.

Como isso é feito?

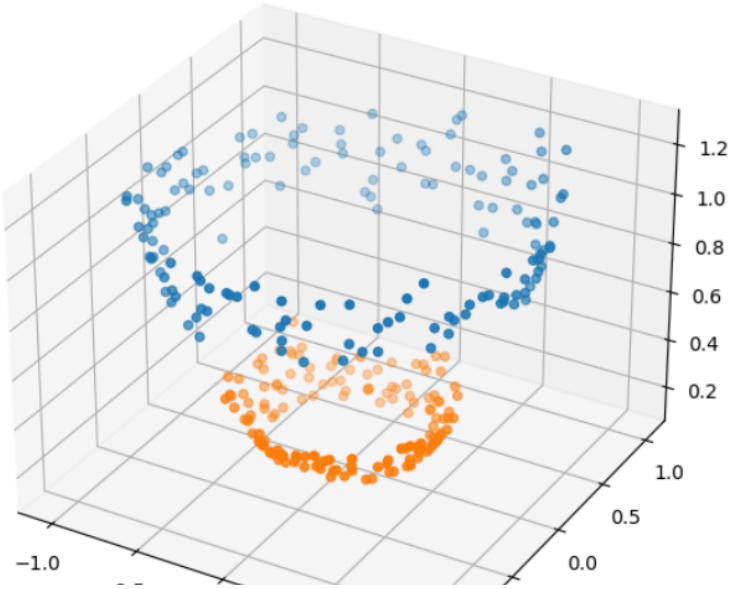
- O SVM vai usar as funções de kernel

Funções de Kernel

- **Na prática, os Kernels mais usados são:**

Tipo	$K(\vec{x}_i, \vec{x}_j)$	Parâmetros
Linear	$\delta(\vec{x}_i \cdot \vec{x}_j) + \kappa$	δ e κ
Polinomial	$(\delta(\vec{x}_i \cdot \vec{x}_j) + \kappa)^d$	δ, κ e d
Gaussiano	$e^{-\sigma\|\vec{x}_i - \vec{x}_j\|^2}$	σ
Sigmoidal	$\tanh(\delta(\vec{x}_i \cdot \vec{x}_j) + \kappa)$	δ e κ

- Isso é feito no algoritmo elevando as dimensões dos dados.



Vantagens e Desvantagens do SVM

- Ele é eficaz em espaços de alta dimensionalidade;
- É capaz de lidar com dados não linearmente separáveis e possui uma boa capacidade de generalização;
- SVM pode ser computacionalmente caro, especialmente para grandes conjuntos de dados;
- Versátil: diferentes funções do Kernel podem ser especificadas para a função de decisão.
- Kernels comuns são fornecidos, mas também é possível especificar kernels personalizados.

Estudo de Caso

Contents

1 - Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import accuracy_score, precision_score, recall_sc
from sklearn.metrics import classification_report, confusion_matrix
%matplotlib inline
```

2 - Carregando os dados

- Vamos usar o conjunto de dados de *Breast Cancer* disponível no **Scikit-Learn**.
- Este conjunto de dados é amplamente utilizado para problemas de **classificação binária.**
- Contém informações de características extraídas de imagens digitalizadas de massas mamárias.
- O objetivo é classificar as massas como malignas ou benignas com base nas características fornecidas.

In [2]:
```
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y=True, as_frame=True)
```

3 - Análise Exploratória

In [3]:
```
X.head()
```

Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	me symme
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1

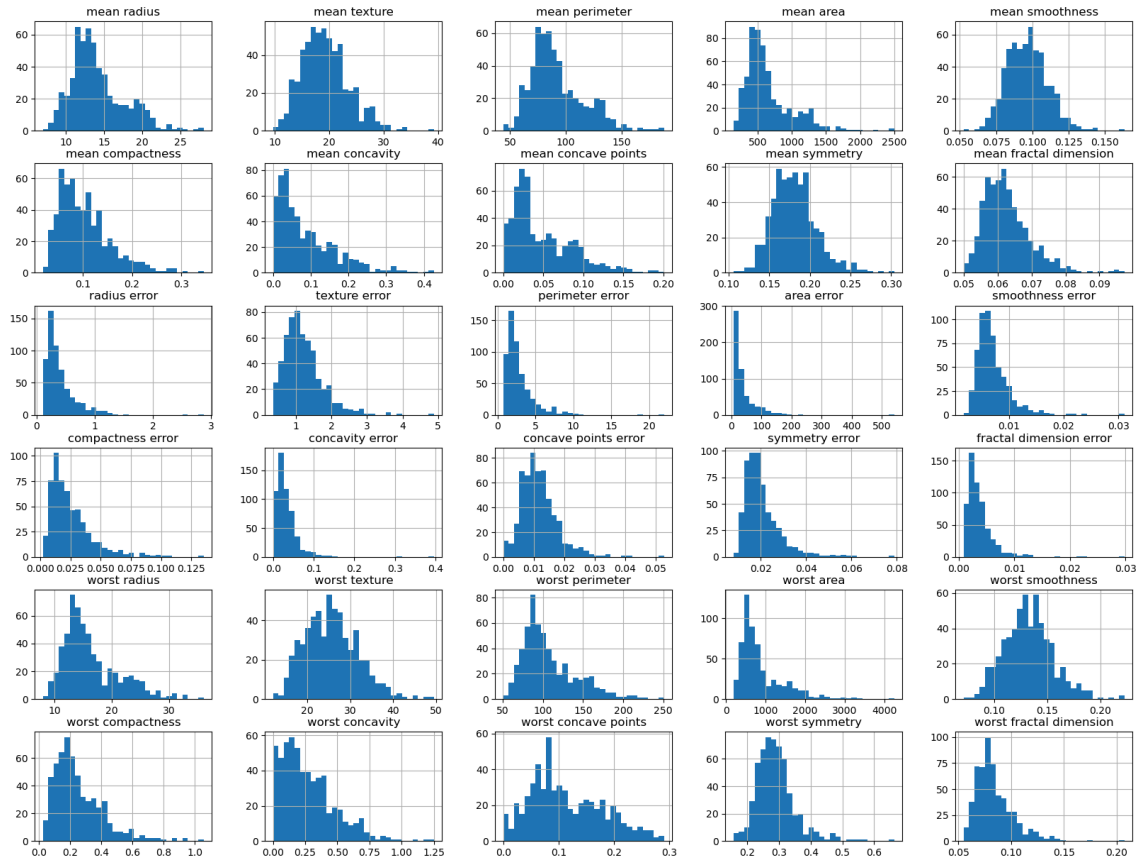
5 rows × 30 columns

In [4]:
```
1 X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

Exibindo histograma das características

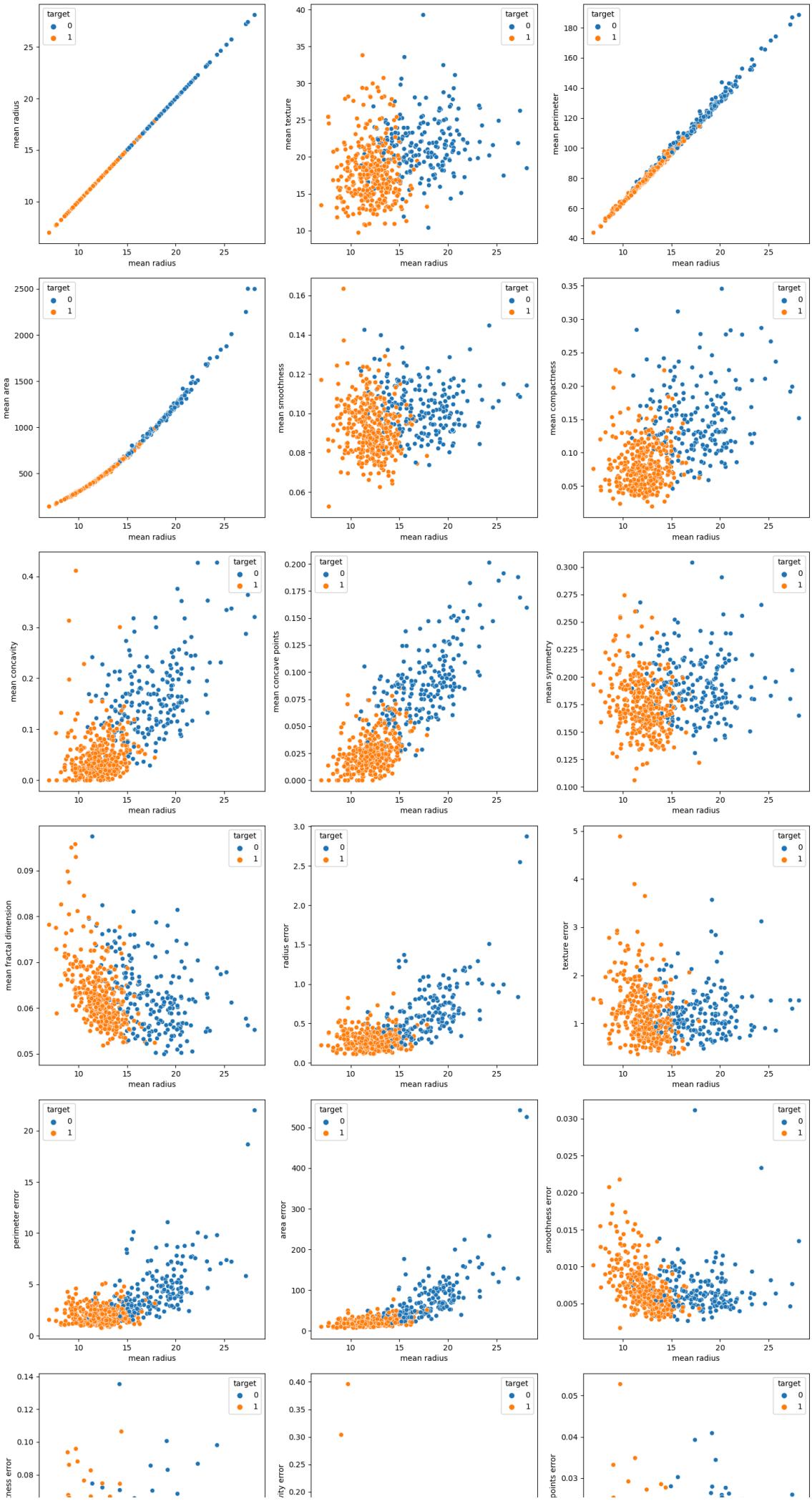
In [5]:
```
1 X.hist(bins=30, figsize=(20,15))
2 plt.show()
```

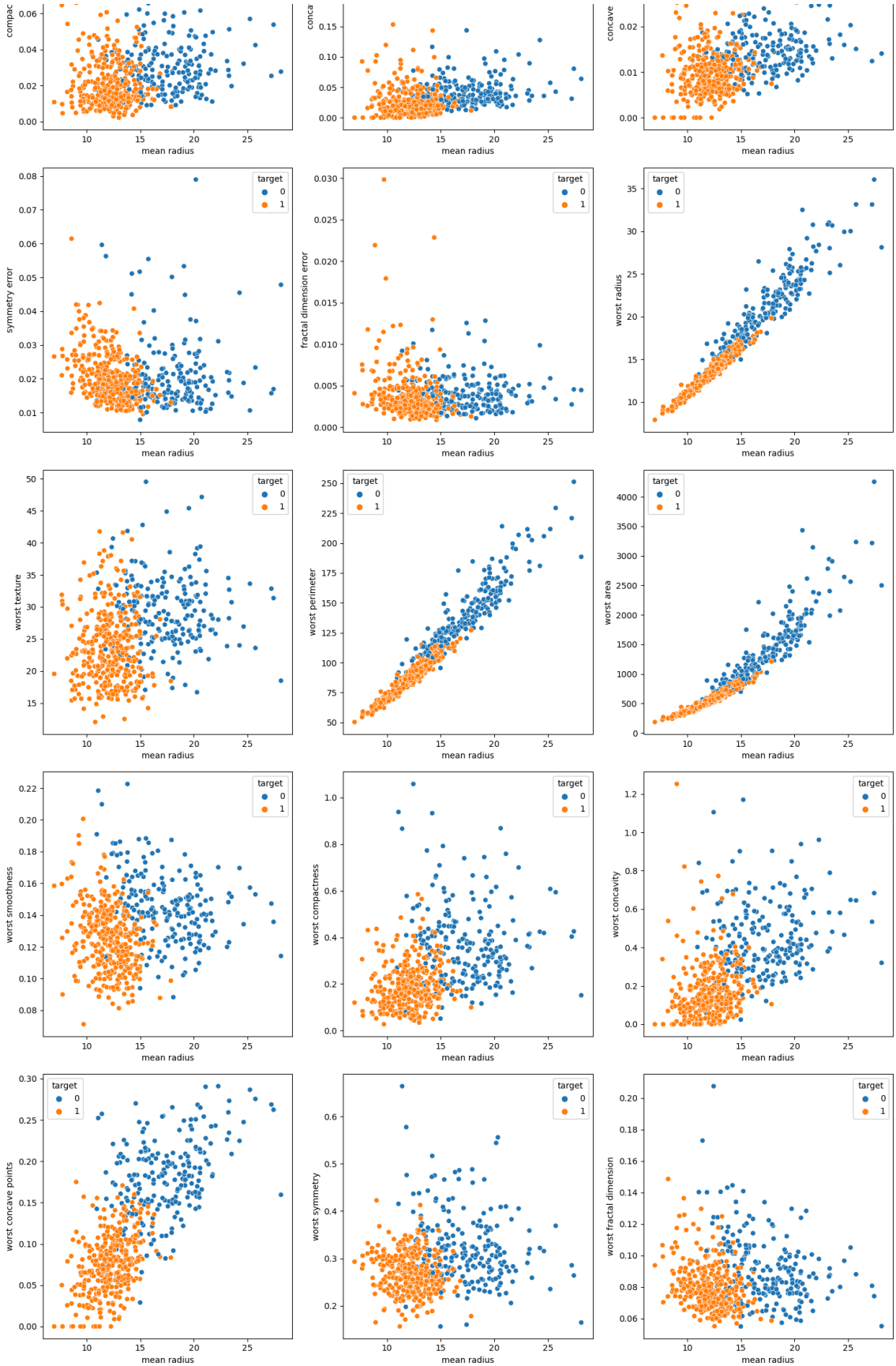


Traçando os gráficos de dispersão das caracteríticas

In [6]:

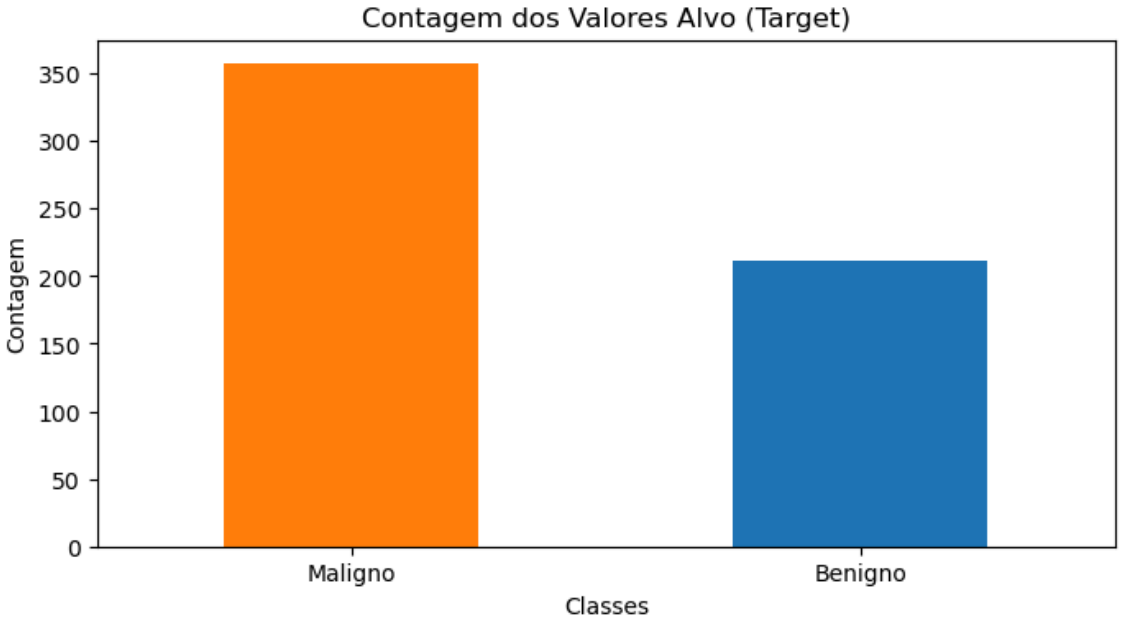
```
X_copy = X.copy()
X_copy['target'] = y

palette = sns.color_palette()[:2]
num_features = X.shape[1]  # Número de características
num_cols = 3  # Número de colunas para os subplots
num_rows = (num_features + num_cols - 1) // num_cols  # Calcular o núme

plt.figure(figsize=(15, 50))

# Laço para criar os gráficos de dispersão
for i, feature in enumerate(X.columns):  # Iterar por todas as colunas
    plt.subplot(num_rows, num_cols, i + 1)
    sns.scatterplot(x=X_copy['mean radius'], y=X_copy[feature], hue=X_c
    plt.xlabel('mean radius')
    plt.ylabel(feature)

plt.tight_layout()
plt.show()
```


target
0
1
mean radius
mean texture
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


compac
conca
concave
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
mean radius
target
0
1

Exibindo a contagem dos valores da coluna alvo (Target)

In [7]:
```
palette = sns.color_palette()
value_counts = y.value_counts()
plt.figure(figsize=(8, 4))

colors = [palette[1],palette[0]]  # Cores diferentes para as classes
value_counts.plot(kind='bar', color=colors)
plt.title('Contagem dos Valores Alvo (Target)')
plt.xlabel('Classes')
plt.ylabel('Contagem')
plt.xticks(ticks=[0, 1], labels=['Maligno', 'Benigno'], rotation=0)
plt.show()
```



In [8]:
```
y.value_counts()
```

Out[8]:
```
1    357
0    212
Name: target, dtype: int64
```

Separar conjunto de dados de treino e teste

Antes de explorar alguns insights sobre os dados, vamos dividi-los em conjunto de teste e conjunto de treinamento.

In [9]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,

print('O tamanho dos dados de treinamento é: {} \nO tamanho dos dados d
```

```
O tamanho dos dados de treinamento é: 398
O tamanho dos dados de teste é: 171
```

In [10]:
```
1 y_test.value_counts()
```

Out[10]:
```
1    107
0     64
Name: target, dtype: int64
```

4 - Pré-processamento

- O SVM funciona bem com valores escalonados.
- Aqui é feita a configuração para escalar esses valores para ficarem entre 0 e 1.

In [11]:
```
1 X_train.max()
```

Out[11]:
```
mean radius                 28.11000
mean texture                39.28000
mean perimeter             188.50000
mean area                 2499.00000
mean smoothness              0.14470
mean compactness             0.34540
mean concavity               0.42680
mean concave points          0.20120
mean symmetry                0.30400
mean fractal dimension       0.09296
radius error                 2.87300
texture error                4.88500
perimeter error             21.98000
area error                 525.60000
smoothness error             0.03113
compactness error            0.13540
concavity error              0.39600
concave points error         0.05279
symmetry error               0.07895
fractal dimension error      0.02984
worst radius                33.12000
worst texture               49.54000
worst perimeter            220.80000
worst area                3432.00000
worst smoothness             0.22260
worst compactness            0.93790
worst concavity              1.25200
worst concave points         0.29100
worst symmetry               0.57740
worst fractal dimension      0.14860
dtype: float64
```

In [12]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

scale_pipe = Pipeline([
    ('scaler', MinMaxScaler())
])

X_train_scaled = scale_pipe.fit_transform(X_train)
X_test_scaled = scale_pipe.transform(X_test)
```

In [13]:
```
X_train_scaled
```

Out[13]:
```
array([[0.33125089, 0.31972318, 0.32706793, ..., 0.62783505, 0.38346401,
        0.53933305],
       [0.48364807, 0.48927336, 0.48655933, ..., 0.65257732, 0.41530055,
        0.84288157],
       [0.17128118, 0.29653979, 0.17614539, ..., 0.27237113, 0.32668092,
        0.22263788],
       ...,
       [0.20393772, 0.09204152, 0.19653099, ..., 0.30068729, 0.27750059,
        0.26913211],
       [0.18453311, 0.18131488, 0.18395412, ..., 0.2737457 , 0.2413875 ,
        0.56605387],
       [0.26404468, 0.30069204, 0.2634925 , ..., 0.31838488, 0.13209789,
        0.17999145]])
```

In [14]:
```
X_test_scaled
```

Out[14]:
```
array([[0.17648729, 0.34636678, 0.17766568, ..., 0.29553265, 0.24708957,
        0.34170586],
       [0.40839604, 0.31141869, 0.38843204, ..., 0.29443299, 0.26562129,
        0.14162035],
       [0.55132756, 0.50968858, 0.55980927, ..., 0.63505155, 0.44642433,
        0.45724669],
       ...,
       [0.35964788, 0.38581315, 0.37053417, ..., 0.92817869, 0.64124495,
        0.77447627],
       [0.37810592, 0.22733564, 0.36231083, ..., 0.29158076, 0.19933476,
        0.09876015],
       [0.45525108, 0.61245675, 0.44578813, ..., 0.48728522, 0.15514374,
        0.24754168]])
```

5 - Classificador SVM

Parâmetros do SVM

Principais: C, Gamma, Degree, Coef0.

C (Parâmetro de Regularização):

- Controla a penalidade dos erros de classificação.
- Alto: Modelo se ajusta bem aos dados de treino, menor margem de erro, mas risco de overfitting.
- Baixo: Maior margem de erro, pode generalizar melhor, mas risco de underfitting.

Gamma:

- Coeficiente do kernel para RBF, polinomial e sigmoidal.
- Alto: Pequeno alcance, alta complexidade, risco de overfitting.
- Baixo: Grande alcance, menor complexidade, risco de underfitting.

Degree:

- Grau do polinômio no kernel polinomial.
- Alto: Maior complexidade, captura padrões complexos, risco de overfitting.
- Baixo: Menor complexidade, pode não capturar padrões complexos, risco de underfitting.

Coef0 (κ):

- Termo independente nos kernels polinomial e sigmoidal.
- Alto: Maior deslocamento, pode melhorar a separação em alguns casos, risco de overfitting.
- Baixo: Menor deslocamento, menos influência, pode não capturar bem a separação.

Scikit-learn

- **O Scikit-learn fornece várias opções para configurar o SVM.**
- **Alguns dos principais parâmetros são:**
- `kernel` : Especifica o tipo de kernel a ser usado na função de decisão. SVM suporta kernels `linear` , `polinomial` , `rbf` , `sigmoid` , `precomputed` ou um `callable` .
- `C` : Parâmetro de regularização. um valor de `C` alto pode fazer o modelo se ajustar bem aos dados, e ter menor a margem de erro, mas corre o risco de memorizar os dados (overfitting).
- `gamma` : Coeficiente do kernel para 'rbf', 'poly' e 'sigmoid'. Se `gamma` for 'scale', `gamma` será 1 / (n_features * X.var()), se for 'auto', `gamma` será 1 / n_features.
- `degree` : Grau do polinômio se o kernel for 'poly'. Ignorado por outros kernels.
- `coef0` : Término independente no kernel 'poly' e 'sigmoid'.
- `shrinking` : Se deve usar heurísticas de encolhimento.
- `probability` : Se deve habilitar estimativas de probabilidade. Isso é mais demorado e deve ser ativado antes de ajustar o modelo.
- `tol` : Critério de tolerância para parada.
- `cache_size` : Tamanho do cache em MB.
- `class_weight` : Peso associado às classes.
- `verbose` : Habilitar saída detalhada.
- `max_iter` : Número máximo de iterações.

Treinamento do Classificador SVM

- Aqui é realizado o treinamento de um classificador: Linear SVC (Baseline)
- Posteriormente serão treinados modelos SVC em podemos usar kernels diferentes.

In [15]:
```
#Linear SVC
lin_svc = LinearSVC(dual=False)
lin_svc.fit(X_train_scaled, y_train)

print('Modelo treinado com sucesso!')
```

```
Modelo treinado com sucesso!
```

6 - Avaliando o classificador SVM - Baseline

Previsões do modelo:

In [16]:
```
lin_pred = lin_svc.predict(X_test_scaled)
```

- Primeiro vamos exibir a matriz de confusão e o relatório de classificação no LinearSVC.
- O relatório de classificação vai além da exatidão, incluindo recall, precisão e pontuação f1.

Matriz de confusão LinearSVC

In [17]:
```
confusion_matrix(y_test, lin_pred)
```

Out[17]:
```
array([[ 63,   1],
       [  0, 107]], dtype=int64)
```

Relatório de classificação LinearSVC

In [18]:
```
print(classification_report(y_test, lin_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        64
           1       0.99      1.00      1.00       107

    accuracy                           0.99       171
   macro avg       1.00      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171
```

- Os resultados são bastante impressionantes;
- Não foi necessário que ajustar nenhum hiperparâmetro.

7 - Funções do kernel

Linear

- São comumente recomendados para classificação de texto;
- A maioria desses tipos de problemas de **classificação são linearmente separáveis.**
- O kernel linear funciona muito bem quando há muitas características;
- Os problemas de classificação de texto têm muitas características;
- As funções lineares do kernel são mais rápidas que a maioria das outras e você tem menos parâmetros para otimizar.

Polinomial

- **Usa uma curva polinomial (tipo uma parábola);**
- O kernel polinomial não é usado na prática com muita frequência;
- Porque não é tão eficiente computacionalmente quanto outros kernels;
- Suas previsões não são tão precisas.

Função de base radial gaussiana (RBF)

- **Usa uma função que cria bolhas ao redor dos dados;**
- Esse é um dos kernels mais poderosos e comumente usados em SVMs;
- Ele geralmente é a escolha para dados **não lineares.**

Sigmoide

- **Usa uma função que se parece com uma curva sigmoidal (parecida com uma**

Treinando e avaliando o SVM com diferentes kernels

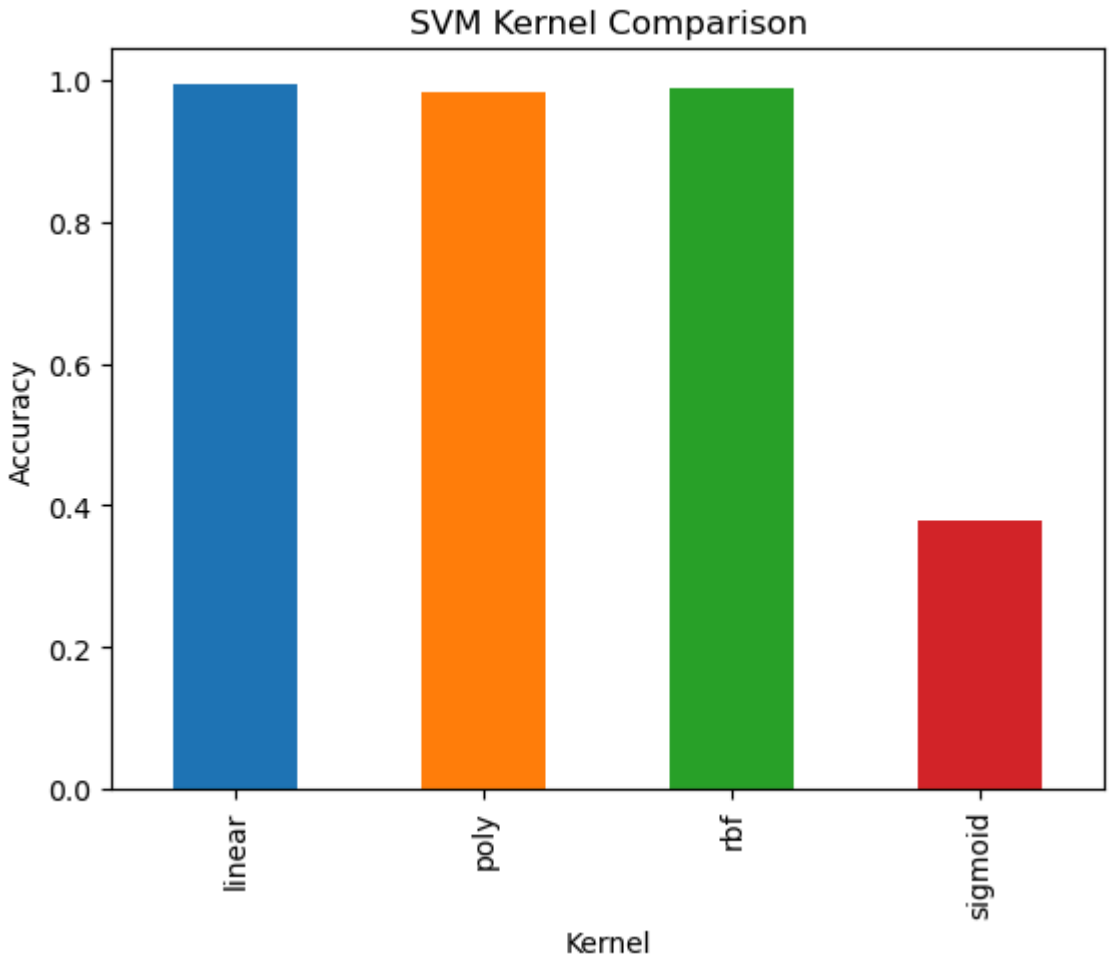
In [19]:
```
def train_and_evaluate_svm(kernel):
    svm = SVC(kernel=kernel)
    svm.fit(X_train_scaled, y_train)
    y_pred = svm.predict(X_test_scaled)

    print(f"Kernel: {kernel}")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print("\n")
```

In [20]:
```
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for kernel in kernels:
    train_and_evaluate_svm(kernel)
```

```
Kernel: linear
[[ 63   1]
 [  0 107]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        64
           1       0.99      1.00      1.00       107

    accuracy                           0.99       171
   macro avg       1.00      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171



Kernel: poly
[[ 62   2]
 [  1 106]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.98        64
           1       0.98      0.99      0.99       107

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171



Kernel: rbf
[[ 63   1]
 [  1 106]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        64
           1       0.99      0.99      0.99       107

    accuracy                           0.99       171
   macro avg       0.99      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171



Kernel: sigmoid
[[ 1 63]
 [43 64]]
              precision    recall  f1-score   support

           0       0.02      0.02      0.02        64
           1       0.50      0.60      0.55       107

    accuracy                           0.38       171
   macro avg       0.26      0.31      0.28       171
weighted avg       0.32      0.38      0.35       171
```

In [21]:

```
results = []
for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(X_train_scaled, y_train)
    score = svm.score(X_test_scaled, y_test)
    results.append((kernel, score))

results_df = pd.DataFrame(results, columns=['Kernel', 'Accuracy'])
colors = sns.color_palette()[:4]
results_df.plot(kind='bar', x='Kernel', y='Accuracy', legend=False, tit
plt.ylabel('Accuracy')
plt.show()
```



- Os resultados continuam bons, mesmo não tendo sido realizado ajustes de nenhum hiperparâmetro.
- Embora isso seja suficiente para nosso conjunto de dados, é improvável que seu modelo funcione bem inicialmente na vida real.
- Talvez seja necessário ajustar hiperparâmetros.
- Existem duas técnicas comuns para pesquisa de hiperparâmetros.
- **Estes são Pesquisa Aleatória e GridSearch.**

8 - Melhorando o classificador de vetores de suporte

Pesquisa de hiperparâmetros - Gridsearch

- Neste exemplo iremos tentar encontrar melhores hiperparâmetros para o kernel sigmoid.

In [22]:
```
from sklearn.model_selection import GridSearchCV

params_grid = {'C':[0.001,10,100,1000],
               'gamma':[1,0.1,0.01,0.001],
               'coef0':[0,1,2,4]

               }

grid_search = GridSearchCV(SVC(kernel='sigmoid'), params_grid, verbose=

grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
[CV] END .........................C=0.001, coef0=0, gamma=1; total tim
e=   0.0s
[CV] END .........................C=0.001, coef0=0, gamma=1; total tim
e=   0.0s
[CV] END .........................C=0.001, coef0=0, gamma=1; total tim
e=   0.0s
[CV] END .........................C=0.001, coef0=0, gamma=1; total tim
e=   0.0s
[CV] END .........................C=0.001, coef0=0, gamma=1; total tim
e=   0.0s
[CV] END .......................C=0.001, coef0=0, gamma=0.1; total tim
e=   0.0s
[CV] END .......................C=0.001, coef0=0, gamma=0.1; total tim
e=   0.0s
[CV] END .......................C=0.001, coef0=0, gamma=0.1; total tim
e=   0.0s
[CV] END .......................C=0.001, coef0=0, gamma=0.1; total tim
e=   0.0s
```

Melhores Parâmetros

In [23]:
```
grid_search.best_params_
```

Out[23]:
```
{'C': 100, 'coef0': 1, 'gamma': 0.1}
```

Treinando o melhor modelo encontrado pelo grid search com os dados de treinamento escalados

In [24]:
```
1 sig_best = grid_search.best_estimator_.fit(X_train_scaled, y_train)
```

Previsões do Modelo

In [25]:
```
1 grid_pred = sig_best.predict(X_test_scaled)
```

In [26]:
```
1 confusion_matrix(y_test, grid_pred)
```

Out[26]:
```
array([[ 60,   4],
       [  0, 107]], dtype=int64)
```

In [27]:
```
1 print(classification_report(y_test, grid_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        64
           1       0.96      1.00      0.98       107

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

Conclusões

- Esta apresntação tratou do uso de máquinas de vetores de suporte para **tarefas de classificação.**
- Como você pode ver, SVM é um algoritmo robusto, dada a forma como suporta diferentes kernels.
- Esses kernels são o que o tornam adequado para problemas lineares e não lineares.
- No mundo real, muitos conjuntos de dados não são lineares.
- Portanto, quando você não conseguir bons resultados com modelos lineares, SVM é uma boa alternativa.

Referências:

IZBICKI, Rafael; DOS SANTOS, Tiago Mendonça. Aprendizado de máquina: uma abordagem estatística. Rafael Izbicki, 2020.

TAULLI, Tom. Introdução à Inteligência Artificial: Uma abordagem não técnica. Novatec Editora, 2020.

ELKORANY, Ahmed S. et al. Breast cancer diagnosis using support vector machines optimized by whale optimization and dragonfly algorithms. IEEE Access, v. 10, p. 69688-69699, 2022.