In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf

from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import train_test_split
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

pd.set_option('display.expand_frame_repr', False)
```

Importando o banco de dados

O banco de dados importado já passou pelo pré-processamendo no algoritmo anterior e está pronto para ser treinado.

In [2]:
```
df = pd.read_csv('Outputs/filtrado.csv')
status = df.Status.unique()
print(f'Número de linhas: {df.shape [0]}')
print(f'Número de colunas: {df.shape [1]}')
```

```
Número de linhas: 1812
Número de colunas: 29
```

Gerando os dados

Foi separado 20% do bando de dados para testes e 80% para treino.

Em seguida o vetor de target foi convertido para OneHotEncoder, que será utilizado em vários dos métodos a seguir.

Por último as features são deixadas na mesma escala para melhorar o aprendizado de máquina.

In [3]:
```
# Gerando X e y

features = df.columns[:-1]
X = df[features]

y_labels = np.array(df.Status)
y = np.array(df.Status).reshape(-1, 1)
enc = OneHotEncoder(handle_unknown='ignore').fit(y)
y = enc.transform(y).toarray()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0) # Target OneHotEncoding
X2_train, X2_test, y2_train, y2_test = train_test_split(X, y_labels, test_size = 0.2, random_state = 0) # Target Label


sc = StandardScaler()
X_train= sc.fit_transform(X_train)
X_test= sc.transform(X_test)
X2_train= sc.fit_transform(X2_train)
X2_test= sc.transform(X2_test)
```

Hyper Parameter Tuning

A seguir é feito o Hyper Parameter Tuning de vários modelos de aprendizado de máquinas. Ao final do programa será rodado o melhor classificador encontrado para cada modelo e comparados lado a lado.

Árvore de Decisão

A acurária do modelo é mostrado ao final do programa junto com todos os outros.

In [4]:
```python
# Hyper Parameter Tuning Árvore de Decisão
DecisionTree = DecisionTreeClassifier(random_state = 0)

parameters = {'criterion': ('entropy', 'gini'),
              'max_depth': [10, 20, 30, 40]}

clf = GridSearchCV(DecisionTree, parameters)
clf.fit(X_train, y_train)
DecisionTree = clf.best_estimator_
print(DecisionTree)
```
```
DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)
```

Random Forest

A acurária do modelo é mostrado ao final do programa junto com todos os outros.

In [5]:
```python
# Hyper Parameter Tuning Random Forest

RandomForest = RandomForestClassifier(random_state = 0)

parameters = {'criterion': ('entropy', 'gini'),
              'max_depth': [10, 20, 30, 40],
              'bootstrap': [False, True]}

clf = GridSearchCV(RandomForest, parameters)
clf.fit(X_train, y_train)
RandomForest = clf.best_estimator_
print(RandomForest)
```
```
RandomForestClassifier(bootstrap=False, criterion='entropy', max_depth=20,
                       random_state=0)
```

Support Vector Machine

A acurária do modelo é mostrado ao final do programa junto com todos os outros.

In [6]:
```python
# Support Vector Machine

svc = SVC(random_state = 0)


parameters = {'kernel': ('linear', 'rbf', 'sigmoid', 'poly'),
              'decision_function_shape': ['ovo', 'ovr'],
              'degree': [2, 3],
              'C': [5, 7, 9, 11]}

clf = GridSearchCV(svc, parameters)
clf.fit(X2_train, y2_train)
svc = clf.best_estimator_
print(svc)
```
```
SVC(C=7, decision_function_shape='ovo', degree=2, random_state=0)
```

K Nearest Neighbours

A acurária do modelo é mostrado ao final do programa junto com todos os outros.

In [7]:
```python
# KNNeighbours

knn = KNeighborsClassifier()

parameters = {'n_neighbors': [3, 5, 7, 9, 11],
              'p': [1, 2, 3]}

clf = GridSearchCV(knn, parameters)
clf.fit(X_train, y_train)
knn = clf.best_estimator_
print(knn)
```
```
KNeighborsClassifier(n_neighbors=3, p=1)
```

Rede Neural

A acurária do modelo é mostrado ao final do programa junto com todos os outros.

In [8]:
```python
# Rede Neural
ann0 = tf.keras.models.Sequential()
ann0.add(tf.keras.layers.Dense(units=20, activation='relu'))
ann0.add(tf.keras.layers.Dense(units=10, activation='softmax'))

ann0.compile(optimizer = 'adam', loss = 'BinaryCrossentropy', metrics = ['accuracy'])
```

In [9]:
```
funcao = lambda x: 1e-4 * 10**(x/20)

n_epochs = 100

teste1 = ann0.fit(X_train, y_train, batch_size = 32, epochs = n_epochs,
                 callbacks = [tf.keras.callbacks.LearningRateScheduler( lambda epoch: funcao(epoch) )]
                 )
```

```
Epoch 1/100
46/46 [==============================] - 1s 2ms/step - loss: 0.7400 - accuracy: 0.0849 - lr: 1.0000e-04
Epoch 2/100
46/46 [==============================] - 0s 2ms/step - loss: 0.7183 - accuracy: 0.0883 - lr: 1.1220e-04
Epoch 3/100
46/46 [==============================] - 0s 2ms/step - loss: 0.6954 - accuracy: 0.0959 - lr: 1.2589e-04
Epoch 4/100
46/46 [==============================] - 0s 2ms/step - loss: 0.6713 - accuracy: 0.1035 - lr: 1.4125e-04
Epoch 5/100
46/46 [==============================] - 0s 2ms/step - loss: 0.6461 - accuracy: 0.1180 - lr: 1.5849e-04
Epoch 6/100
46/46 [==============================] - 0s 2ms/step - loss: 0.6197 - accuracy: 0.1311 - lr: 1.7783e-04
Epoch 7/100
46/46 [==============================] - 0s 2ms/step - loss: 0.5921 - accuracy: 0.1442 - lr: 1.9953e-04
Epoch 8/100
46/46 [==============================] - 0s 2ms/step - loss: 0.5631 - accuracy: 0.1594 - lr: 2.2387e-04
Epoch 9/100
46/46 [==============================] - 0s 2ms/step - loss: 0.5329 - accuracy: 0.1746 - lr: 2.5119e-04
Epoch 10/100
46/46 [==============================] - 0s 2ms/step - loss: 0.5017 - accuracy: 0.1932 - lr: 2.8184e-04
Epoch 11/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4697 - accuracy: 0.2167 - lr: 3.1623e-04
Epoch 12/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4372 - accuracy: 0.2450 - lr: 3.5481e-04
Epoch 13/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4049 - accuracy: 0.2678 - lr: 3.9811e-04
Epoch 14/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3737 - accuracy: 0.2892 - lr: 4.4668e-04
Epoch 15/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3442 - accuracy: 0.3313 - lr: 5.0119e-04
Epoch 16/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3171 - accuracy: 0.3734 - lr: 5.6234e-04
Epoch 17/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2926 - accuracy: 0.4099 - lr: 6.3096e-04
Epoch 18/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2708 - accuracy: 0.4672 - lr: 7.0795e-04
Epoch 19/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2513 - accuracy: 0.5273 - lr: 7.9433e-04
Epoch 20/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2339 - accuracy: 0.5776 - lr: 8.9125e-04
Epoch 21/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2183 - accuracy: 0.6239 - lr: 0.0010
Epoch 22/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2041 - accuracy: 0.6556 - lr: 0.0011
Epoch 23/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1909 - accuracy: 0.6805 - lr: 0.0013
Epoch 24/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1790 - accuracy: 0.7032 - lr: 0.0014
Epoch 25/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1677 - accuracy: 0.7212 - lr: 0.0016
Epoch 26/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1575 - accuracy: 0.7412 - lr: 0.0018
Epoch 27/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1479 - accuracy: 0.7619 - lr: 0.0020
Epoch 28/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1390 - accuracy: 0.7736 - lr: 0.0022
Epoch 29/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1311 - accuracy: 0.7840 - lr: 0.0025
Epoch 30/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1235 - accuracy: 0.7985 - lr: 0.0028
Epoch 31/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1167 - accuracy: 0.8102 - lr: 0.0032
Epoch 32/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1101 - accuracy: 0.8157 - lr: 0.0035
Epoch 33/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1042 - accuracy: 0.8261 - lr: 0.0040
Epoch 34/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0986 - accuracy: 0.8337 - lr: 0.0045
Epoch 35/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0937 - accuracy: 0.8364 - lr: 0.0050
Epoch 36/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0894 - accuracy: 0.8475 - lr: 0.0056
Epoch 37/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0856 - accuracy: 0.8496 - lr: 0.0063
Epoch 38/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0821 - accuracy: 0.8571 - lr: 0.0071
Epoch 39/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0783 - accuracy: 0.8661 - lr: 0.0079
Epoch 40/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0751 - accuracy: 0.8772 - lr: 0.0089
Epoch 41/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0732 - accuracy: 0.8751 - lr: 0.0100
Epoch 42/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0695 - accuracy: 0.8772 - lr: 0.0112
Epoch 43/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0679 - accuracy: 0.8930 - lr: 0.0126
Epoch 44/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0660 - accuracy: 0.8820 - lr: 0.0141
Epoch 45/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0620 - accuracy: 0.8979 - lr: 0.0158
Epoch 46/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0602 - accuracy: 0.8986 - lr: 0.0178
Epoch 47/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0585 - accuracy: 0.9020 - lr: 0.0200
Epoch 48/100
```

```
46/46 [==============================] - 0s 2ms/step - loss: 0.0585 - accuracy: 0.9048 - lr: 0.0224
Epoch 49/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0566 - accuracy: 0.9013 - lr: 0.0251
Epoch 50/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0569 - accuracy: 0.9020 - lr: 0.0282
Epoch 51/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0587 - accuracy: 0.9013 - lr: 0.0316
Epoch 52/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0621 - accuracy: 0.8965 - lr: 0.0355
Epoch 53/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0616 - accuracy: 0.9082 - lr: 0.0398
Epoch 54/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0628 - accuracy: 0.8951 - lr: 0.0447
Epoch 55/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0713 - accuracy: 0.8806 - lr: 0.0501
Epoch 56/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0696 - accuracy: 0.8882 - lr: 0.0562
Epoch 57/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0748 - accuracy: 0.8834 - lr: 0.0631
Epoch 58/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1040 - accuracy: 0.8599 - lr: 0.0708
Epoch 59/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1082 - accuracy: 0.8454 - lr: 0.0794
Epoch 60/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1305 - accuracy: 0.8406 - lr: 0.0891
Epoch 61/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1521 - accuracy: 0.8337 - lr: 0.1000
Epoch 62/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1629 - accuracy: 0.8144 - lr: 0.1122
Epoch 63/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1651 - accuracy: 0.8219 - lr: 0.1259
Epoch 64/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1847 - accuracy: 0.8033 - lr: 0.1413
Epoch 65/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2764 - accuracy: 0.7350 - lr: 0.1585
Epoch 66/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2407 - accuracy: 0.7488 - lr: 0.1778
Epoch 67/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2852 - accuracy: 0.7509 - lr: 0.1995
Epoch 68/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4850 - accuracy: 0.6984 - lr: 0.2239
Epoch 69/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3956 - accuracy: 0.6563 - lr: 0.2512
Epoch 70/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2690 - accuracy: 0.5914 - lr: 0.2818
Epoch 71/100
46/46 [==============================] - 0s 2ms/step - loss: 0.2465 - accuracy: 0.6363 - lr: 0.3162
Epoch 72/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3138 - accuracy: 0.5997 - lr: 0.3548
Epoch 73/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4668 - accuracy: 0.5549 - lr: 0.3981
Epoch 74/100
46/46 [==============================] - 0s 1ms/step - loss: 0.4619 - accuracy: 0.5204 - lr: 0.4467
Epoch 75/100
46/46 [==============================] - 0s 2ms/step - loss: 0.7530 - accuracy: 0.4872 - lr: 0.5012
Epoch 76/100
46/46 [==============================] - 0s 2ms/step - loss: 0.5425 - accuracy: 0.4451 - lr: 0.5623
Epoch 77/100
46/46 [==============================] - 0s 2ms/step - loss: 0.8494 - accuracy: 0.4720 - lr: 0.6310
Epoch 78/100
46/46 [==============================] - 0s 2ms/step - loss: 0.8855 - accuracy: 0.4465 - lr: 0.7079
Epoch 79/100
46/46 [==============================] - 0s 2ms/step - loss: 0.9721 - accuracy: 0.3982 - lr: 0.7943
Epoch 80/100
46/46 [==============================] - 0s 2ms/step - loss: 1.6971 - accuracy: 0.3941 - lr: 0.8913
Epoch 81/100
46/46 [==============================] - 0s 2ms/step - loss: 1.3306 - accuracy: 0.4251 - lr: 1.0000
Epoch 82/100
46/46 [==============================] - 0s 2ms/step - loss: 2.2398 - accuracy: 0.4217 - lr: 1.1220
Epoch 83/100
46/46 [==============================] - 0s 2ms/step - loss: 2.4595 - accuracy: 0.3754 - lr: 1.2589
Epoch 84/100
46/46 [==============================] - 0s 2ms/step - loss: 2.2519 - accuracy: 0.3458 - lr: 1.4125
Epoch 85/100
46/46 [==============================] - 0s 2ms/step - loss: 2.9557 - accuracy: 0.2802 - lr: 1.5849
Epoch 86/100
46/46 [==============================] - 0s 2ms/step - loss: 1.8292 - accuracy: 0.2802 - lr: 1.7783
Epoch 87/100
46/46 [==============================] - 0s 2ms/step - loss: 2.9472 - accuracy: 0.2664 - lr: 1.9953
Epoch 88/100
46/46 [==============================] - 0s 2ms/step - loss: 6.7484 - accuracy: 0.2471 - lr: 2.2387
Epoch 89/100
46/46 [==============================] - 0s 2ms/step - loss: 7.6770 - accuracy: 0.2471 - lr: 2.5119
Epoch 90/100
46/46 [==============================] - 0s 2ms/step - loss: 4.3252 - accuracy: 0.2257 - lr: 2.8184
Epoch 91/100
46/46 [==============================] - 0s 2ms/step - loss: 3.8720 - accuracy: 0.2105 - lr: 3.1623
Epoch 92/100
46/46 [==============================] - 0s 2ms/step - loss: 3.0176 - accuracy: 0.1836 - lr: 3.5481
Epoch 93/100
46/46 [==============================] - 0s 1ms/step - loss: 0.8032 - accuracy: 0.1573 - lr: 3.9811
Epoch 94/100
46/46 [==============================] - 0s 1ms/step - loss: 0.3786 - accuracy: 0.1663 - lr: 4.4668
Epoch 95/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3635 - accuracy: 0.1774 - lr: 5.0119
```

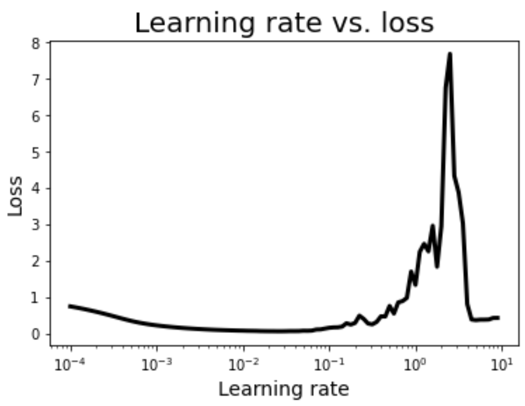
```
Epoch 96/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3747 - accuracy: 0.1760 - lr: 5.6234
Epoch 97/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3754 - accuracy: 0.1518 - lr: 6.3096
Epoch 98/100
46/46 [==============================] - 0s 2ms/step - loss: 0.3792 - accuracy: 0.1594 - lr: 7.0795
Epoch 99/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4226 - accuracy: 0.1753 - lr: 7.9433
Epoch 100/100
46/46 [==============================] - 0s 2ms/step - loss: 0.4240 - accuracy: 0.1601 - lr: 8.9125
```

In [10]:

```
learning_rates = funcao(np.arange(n_epochs))
plt.semilogx(learning_rates, teste1.history['loss'], lw=3, color='#000')
plt.title('Learning rate vs. loss', size=20)
plt.xlabel('Learning rate', size=14)
plt.ylabel('Loss', size=14);
plt.show()

teste_loss = np.array(teste1.history['loss'])
min_loss = teste_loss.min()
rate_min = learning_rates[teste_loss == min_loss][0]

print("\nLoss mínimo:", min_loss)
print("learning rate para mínimo loss:", rate_min)
```



```
Loss mínimo: 0.056622155010700226
learning rate para mínimo loss: 0.025118864315095798
```

In [11]:

```
opt = tf.keras.optimizers.Adam(learning_rate = rate_min)
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10)

ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=20, activation='relu'))
ann.add(tf.keras.layers.Dense(units=10, activation='softmax'))


ann.compile(optimizer = opt, loss = 'BinaryCrossentropy', metrics = ['accuracy'])

ann.fit(X_train, y_train, batch_size = 32, epochs = 100, callbacks= callback)
ann_y_pred = ann.predict(X_test)
ann_y_pred = (ann_y_pred > 0.5)
```

```
Epoch 1/100
46/46 [==============================] - 1s 2ms/step - loss: 0.2592 - accuracy: 0.5466
Epoch 2/100
46/46 [==============================] - 0s 2ms/step - loss: 0.1134 - accuracy: 0.7881
Epoch 3/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0922 - accuracy: 0.8192
Epoch 4/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0813 - accuracy: 0.8571
Epoch 5/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0719 - accuracy: 0.8723
Epoch 6/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0668 - accuracy: 0.8778
Epoch 7/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0617 - accuracy: 0.8875
Epoch 8/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0599 - accuracy: 0.8937
Epoch 9/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0573 - accuracy: 0.9041
Epoch 10/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0562 - accuracy: 0.9130
Epoch 11/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0526 - accuracy: 0.9172
Epoch 12/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0512 - accuracy: 0.9199
Epoch 13/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0485 - accuracy: 0.9186
Epoch 14/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0463 - accuracy: 0.9220
Epoch 15/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0441 - accuracy: 0.9289
Epoch 16/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0447 - accuracy: 0.9296
Epoch 17/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0471 - accuracy: 0.9268
Epoch 18/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0420 - accuracy: 0.9400
Epoch 19/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0380 - accuracy: 0.9386
Epoch 20/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0366 - accuracy: 0.9448
Epoch 21/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0377 - accuracy: 0.9462
Epoch 22/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0391 - accuracy: 0.9427
Epoch 23/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0353 - accuracy: 0.9482
Epoch 24/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0363 - accuracy: 0.9441
Epoch 25/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0344 - accuracy: 0.9489
Epoch 26/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0326 - accuracy: 0.9558
Epoch 27/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0327 - accuracy: 0.9503
Epoch 28/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0275 - accuracy: 0.9655
Epoch 29/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0300 - accuracy: 0.9586
Epoch 30/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0307 - accuracy: 0.9586
Epoch 31/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0294 - accuracy: 0.9565
Epoch 32/100
46/46 [==============================] - 0s 4ms/step - loss: 0.0314 - accuracy: 0.9538
Epoch 33/100
46/46 [==============================] - 0s 5ms/step - loss: 0.0288 - accuracy: 0.9614
Epoch 34/100
46/46 [==============================] - 0s 5ms/step - loss: 0.0263 - accuracy: 0.9662
Epoch 35/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0284 - accuracy: 0.9607
Epoch 36/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0316 - accuracy: 0.9565
Epoch 37/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0293 - accuracy: 0.9579
Epoch 38/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0248 - accuracy: 0.9648
Epoch 39/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0264 - accuracy: 0.9717
Epoch 40/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0325 - accuracy: 0.9572
Epoch 41/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0333 - accuracy: 0.9558
Epoch 42/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0256 - accuracy: 0.9676
Epoch 43/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0255 - accuracy: 0.9614
Epoch 44/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0273 - accuracy: 0.9627
Epoch 45/100
46/46 [==============================] - 0s 3ms/step - loss: 0.0241 - accuracy: 0.9745
Epoch 46/100
46/46 [==============================] - 0s 5ms/step - loss: 0.0240 - accuracy: 0.9689
Epoch 47/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0240 - accuracy: 0.9724
Epoch 48/100
```

```
46/46 [==============================] - 0s 2ms/step - loss: 0.0236 - accuracy: 0.9683
Epoch 49/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0216 - accuracy: 0.9724
Epoch 50/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0192 - accuracy: 0.9800
Epoch 51/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0188 - accuracy: 0.9772
Epoch 52/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0191 - accuracy: 0.9752
Epoch 53/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0207 - accuracy: 0.9807
Epoch 54/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0206 - accuracy: 0.9738
Epoch 55/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0202 - accuracy: 0.9745
Epoch 56/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0187 - accuracy: 0.9779
Epoch 57/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0226 - accuracy: 0.9745
Epoch 58/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0246 - accuracy: 0.9765
Epoch 59/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0220 - accuracy: 0.9772
Epoch 60/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0196 - accuracy: 0.9779
Epoch 61/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0199 - accuracy: 0.9821
Epoch 62/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0206 - accuracy: 0.9772
Epoch 63/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0225 - accuracy: 0.9731
Epoch 64/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0181 - accuracy: 0.9793
Epoch 65/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0159 - accuracy: 0.9848
Epoch 66/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0144 - accuracy: 0.9869
Epoch 67/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0149 - accuracy: 0.9869
Epoch 68/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0132 - accuracy: 0.9896
Epoch 69/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0121 - accuracy: 0.9876
Epoch 70/100
46/46 [==============================] - 0s 1ms/step - loss: 0.0110 - accuracy: 0.9883
Epoch 71/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0109 - accuracy: 0.9910
Epoch 72/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0100 - accuracy: 0.9945
Epoch 73/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0187 - accuracy: 0.9883
Epoch 74/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0161 - accuracy: 0.9834
Epoch 75/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0224 - accuracy: 0.9786
Epoch 76/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0236 - accuracy: 0.9786
Epoch 77/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0240 - accuracy: 0.9779
Epoch 78/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0166 - accuracy: 0.9793
Epoch 79/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0203 - accuracy: 0.9869
Epoch 80/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0168 - accuracy: 0.9848
Epoch 81/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0142 - accuracy: 0.9869
Epoch 82/100
46/46 [==============================] - 0s 2ms/step - loss: 0.0131 - accuracy: 0.9869
```

Resultados Finais

```
In [12]: DecisionTree = DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=0)
         DecisionTree.fit(X_train, y_train)
         tree_y_pred = DecisionTree.predict(X_test)
         print('Árvore de decisão:', accuracy_score(y_test, tree_y_pred))

         RandomForest = RandomForestClassifier(bootstrap=False, criterion='entropy', max_depth=20, random_state=0)
         RandomForest.fit(X_train, y_train)
         forest_y_pred = RandomForest.predict(X_test)
         print("Random Forest:", accuracy_score(y_test, forest_y_pred))

         svc = SVC(C=13, decision_function_shape='ovo', random_state=0)
         svc.fit(X2_train, y2_train)
         svc_y_pred = svc.predict(X2_test)
         print("SVC:", accuracy_score(y2_test, svc_y_pred))

         knn = KNeighborsClassifier(n_neighbors=3, p=1)
         knn.fit(X_train, y_train)
         knn_y_pred = knn.predict(X_test)
         print("KNN:", accuracy_score(y_test, knn_y_pred))

         print("Rede Neural:", accuracy_score(y_test, ann_y_pred))
```

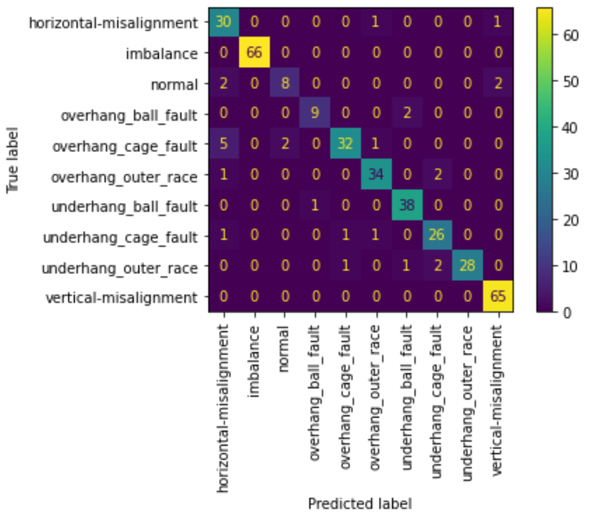
```
Árvore de decisão: 0.8457300275482094
Random Forest: 0.9008264462809917
SVC: 0.9256198347107438
KNN: 0.8677685950413223
Rede Neural: 0.9201101928374655
```

Matriz de Confusão para o SVC

Como o SVC teve o melhor resultado entre todos os modelos, iremos tomar ele como base da nossa matriz de confusão e ter mais detalhes sobre o modelo.

```
In [13]: plot_confusion_matrix(svc, X_test, y2_test, xticks_rotation = 'vertical')
         plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is
deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class method
s: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```


True label
horizontal-misalignment 30 0 0 0 0 1 0 0 0 1
imbalance 0 66 0 0 0 0 0 0 0 0
normal 2 0 8 0 0 0 0 0 0 2
overhang_ball_fault 0 0 0 9 0 0 2 0 0 0
overhang_cage_fault 5 0 2 0 32 1 0 0 0 0
overhang_outer_race 1 0 0 0 0 34 0 2 0 0
underhang_ball_fault 0 0 0 1 0 0 38 0 0 0
underhang_cage_fault 1 0 0 0 1 1 0 26 0 0
underhang_outer_race 0 0 0 0 1 0 1 2 28 0
vertical-misalignment 0 0 0 0 0 0 0 0 0 65
horizontal-misalignment
imbalance
normal
overhang_ball_fault
overhang_cage_fault
overhang_outer_race
underhang_ball_fault
underhang_cage_fault
underhang_outer_race
vertical-misalignment
Predicted label