Aplicando SVM

Treine um classificador SVM com o dataset anexo. Este dataset contém as transações feitas por uma operadora de cartão de créditos. Existem 492 fraudes de um total de mais de 280 mil transações.

O atributo 'Amount' é o valor da transação. As colunas V1, V2, V3 ... representam as componentes principais do dataset original. O atributo a ser previsto está na coluna Class. O valor 1 representa uma transação fraudulenta e o valor 0, uma transação legítima.

In [1]:

```
from sklearn.metrics import confusion_matrix, f1_score, recall_score, precision_sco
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn import svm
import pandas as pd
```

In [2]:

```
data = pd.read_csv("creditcard.csv")
data
```

...

In [3]:

```
# procurando por valores nulos
n_null = data.isnull().sum().sum()
not_fraud, fraud = data.Class.value_counts()

print(f"valores nulos: {n_null}\nFraudes: {fraud}\nNão fraudes: {not_fraud}")
data.dtypes
```

```
valores nulos: 0
Fraudes: 492
Não fraudes: 284315
```

Out[3]:

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [4]:

```
target = data.Class
features = data.drop("Class", axis=1)

# Dividindo os dados em treinamento e teste
features_train, features_test, target_train, target_test = train_test_split(feature
```

In [5]:

```
for feature_to_scale in features_train:
    scaler = MinMaxScaler()
    features_train[feature_to_scale] = scaler.fit_transform(features_train[[feature

# O resultado não foi bom
# # Aplicando PCA
# pca = PCA(n_components=0.99)
# features_train_pca = pca.fit_transform(features_train)
# components = features_train_pca.shape[1]
#
# features_train_pca
```

In [6]:

```
# Realizando o under sampling para balancear os dados

ros = RandomUnderSampler()
features_train_res, target_train_res = ros.fit_resample(features_train, target_trai

target_train_res.value_counts()
```

Out[6]:

```
0    391
1    391
Name: Class, dtype: int64
```

In [7]:

```
svm_model = svm.SVC(random_state=0)
svm_model.fit(features_train_res, target_train_res);
```

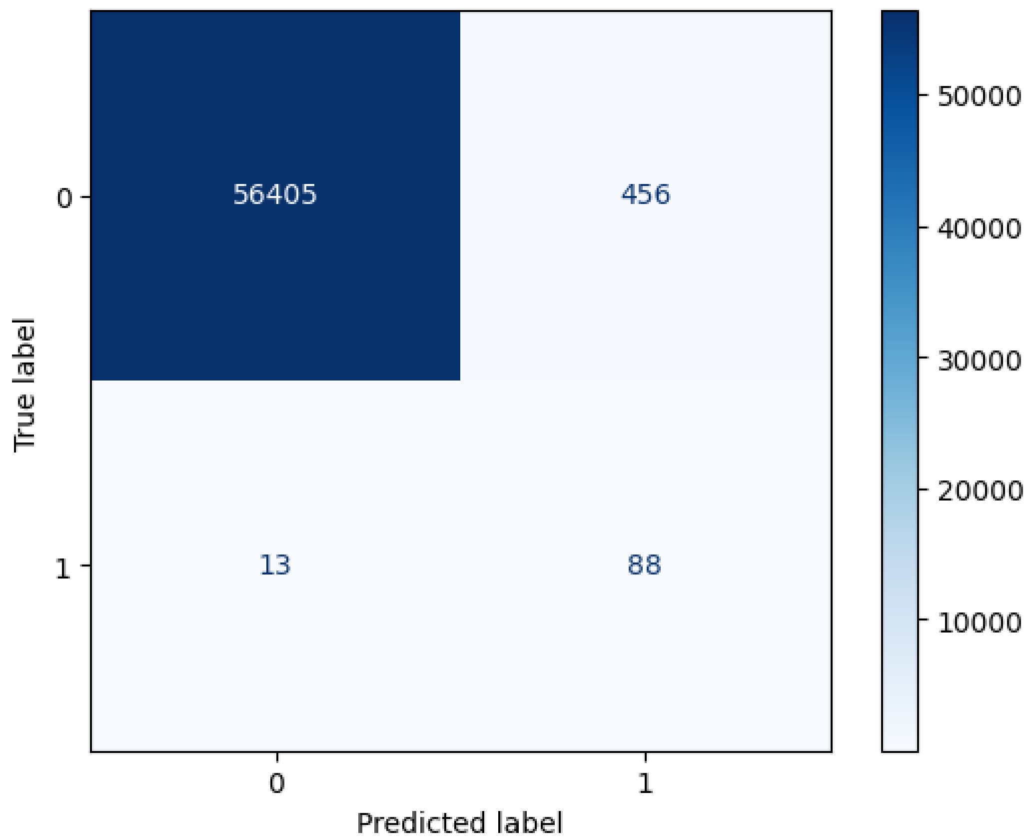
In [8]:

```
# Normalizando os dados de teste

for feature_to_scale in features_test:
    scaler = MinMaxScaler()
    features_test[feature_to_scale] = scaler.fit_transform(features_test[[feature_t

# # Aplicando PCA
# pca = PCA(n_components=components)
# features_test_pca = pca.fit_transform(features_test)
```

In [9]:

```
# Avaliando o modelo
def evaluate_classifier(model, features, target, pos_label=1):
    predictions = model.predict(features)
    accuracy = model.score(features, target)
    precision = precision_score(target, predictions, pos_label=pos_label)
    recall = recall_score(target, predictions, pos_label=pos_label)
    f1 = f1_score(target, predictions, pos_label=pos_label)
    conf_matrix = confusion_matrix(target, predictions)
    ConfusionMatrixDisplay(conf_matrix, display_labels=model.classes_).plot(cmap='B
    print(f'Accuracy: {accuracy:.4}\nPrecision: {precision:.4}\nRecall: {recall:.4}


evaluate_classifier(svm_model, features_test, target_test)
```

```
Accuracy: 0.9918
Precision: 0.1618
Recall: 0.8713
F1: 0.2729
```



In [10]:

```
svm_params = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    # 'degree': [1, 2, 3, 4, 5],
    'gamma': ['scale', 'auto'],
    'shrinking': [True, False],
    'decision_function_shape': ['ovo', 'ovr']
}

svm_cv = svm.SVC()
svm_grid = GridSearchCV(svm_cv, svm_params, cv=5, n_jobs=-1)

svm_grid.fit(features_train, target_train)

print(f'Best score: {svm_grid.best_score_:.4}')
svm_grid.best_params_
```

```
Best score: 0.9994
```

Out[10]:

```
{'decision_function_shape': 'ovo',
 'gamma': 'scale',
 'kernel': 'poly',
 'shrinking': True}
```

In [11]:

```
pd.DataFrame(svm_grid.cv_results_)
```

Out[11]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_decision_function_sl
0	663.440135	528.234927	0.723160	0.446920	
1	28.724244	2.521240	1.375512	0.242188	
2	111.481102	13.823147	0.962097	0.150886	
3	173.616485	10.993138	0.876758	0.333898	
4	23.791789	0.977101	2.311834	0.324551	
5	24.134368	1.602386	2.054058	0.142529	
6	24.525169	1.346916	3.464883	0.446183	
7	20.730744	1.569004	3.219418	0.234899	
8	632.550684	508.477236	0.634330	0.351759	
9	20.882020	1.652359	1.016051	0.153411	
10	17.314111	2.453576	1.758368	0.208948	
11	18.181318	1.432489	1.779518	0.255758	
12	16.022005	1.374712	2.195404	0.301322	
13	14.602821	1.230302	1.978407	0.252804	
14	16.547245	2.301346	2.384894	0.474116	
15	16.194019	2.180642	2.209305	0.477498	
16	599.099170	483.912632	0.504565	0.197420	
17	18.049685	3.278243	0.820245	0.129732	
18	77.414239	12.066043	0.819440	0.108210	
19	132.024068	15.929028	0.699058	0.189781	
20	16.075132	3.058810	2.019927	0.724023	
21	13.071434	2.878046	1.515757	0.144891	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_decision_function_sl
22	14.449360	0.567769	2.575886	0.109475	
23	13.550642	1.101286	2.627486	0.214791	
24	548.130829	445.286859	0.524509	0.246964	
25	15.017933	1.466612	0.820035	0.117952	
26	13.153456	1.168520	1.381385	0.126509	
27	13.409379	1.144919	1.367684	0.059106	
28	12.121113	0.642191	1.827663	0.124696	
29	11.813144	0.490719	1.863762	0.045761	
30	12.603504	0.332155	1.735116	0.189214	
31	10.793207	1.047409	1.449688	0.294332	

In [13]:

```
best_svm = svm.SVC(decision_function_shape='ovo', gamma='scale', kernel='poly', shr
best_svm.fit(features_train, target_train)
evaluate_classifier(best_svm, features_test, target_test)
```

```
Accuracy: 0.9992
Precision: 0.7477
Recall: 0.8218
F1: 0.783
```