APRENDIZAGEM DE MÁQUINA

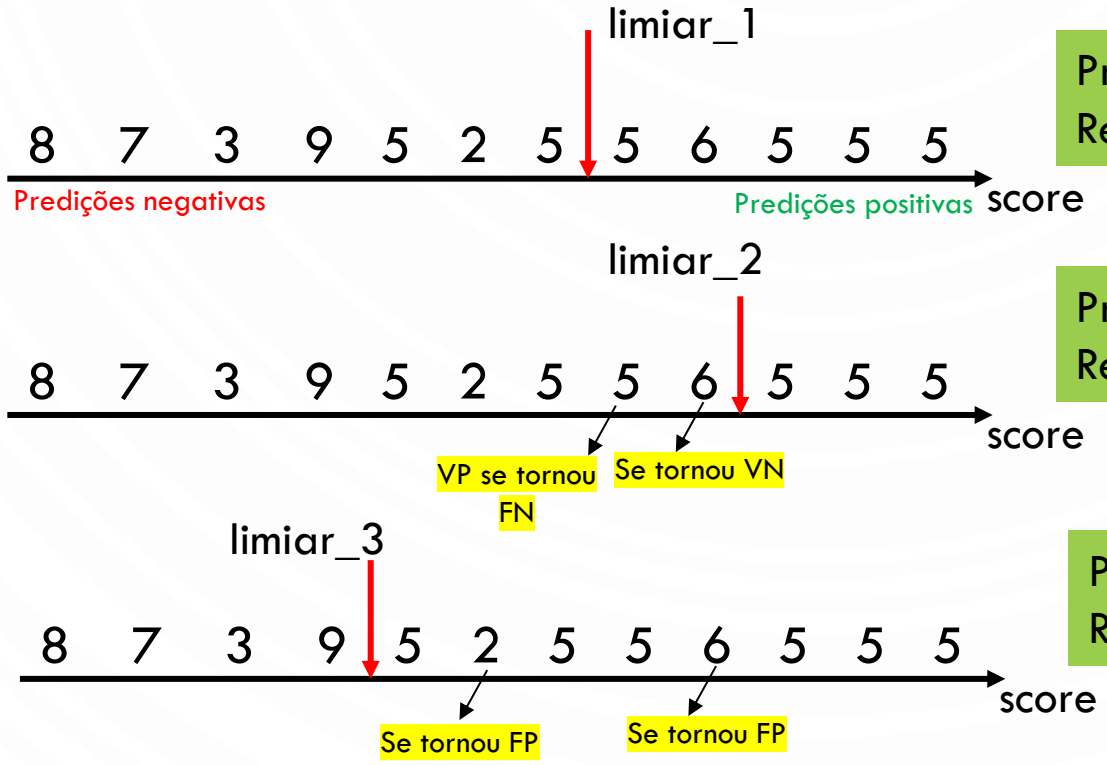
PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br
https://github.com/josenalde/machinelearning

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

A RELAÇÃO PRECISÃO/REVOCAÇÃO

[1] Como um algoritmo de classificação "decide"? Intuitivamente nos parece que é gerada alguma medida interna da relação entre as features (score) e, com base em algum limiar (threshold) a classificação é atribuída à classe X, Y, Z etc.
[2] O SGDClassifier, por exemplo, para cada instância calcula um score baseado em uma função de decisão e, se esse score for maior que um limiar, ele atribui a instância à classe positiva, ou então a atribui à classe negativa.



Precisão: 4/5 (4VP e 1FP) = 80%
Revocação: 4/6 (pois existem 6 5s reais) = 67%

Precisão: 3/3 (3VP e 0FP) = 100%
Revocação: 3/6 = 50%

Precisão: 6/8 (6VP e 2FP) = 75%
Revocação: 6/6 = 100%

A RELAÇÃO PRECISÃO/REVOCAÇÃO

Uma forma de verificar é usar o método decision_function() do modelo ao invés do predict(), a qual retorna um score e pode-se fazer a predição com base neste score, definindo-se um limiar

```
y_scores = model_sgd.decision_function([some_digit])
print(y_scores)

# definindo um threshold = 0 (padrão do SGDClassifier com predict())
thresh = 0

y_some_digit_pred = (y_scores > thresh)
print(y_some_digit_pred) # TRUE

# aumentando o threshold para 8000 dará False, diminuindo a revocação (o classificador perde esta imagem
thresh = 3000
y_some_digit_pred = (y_scores > thresh)
print(y_some_digit_pred) # FALSE
```

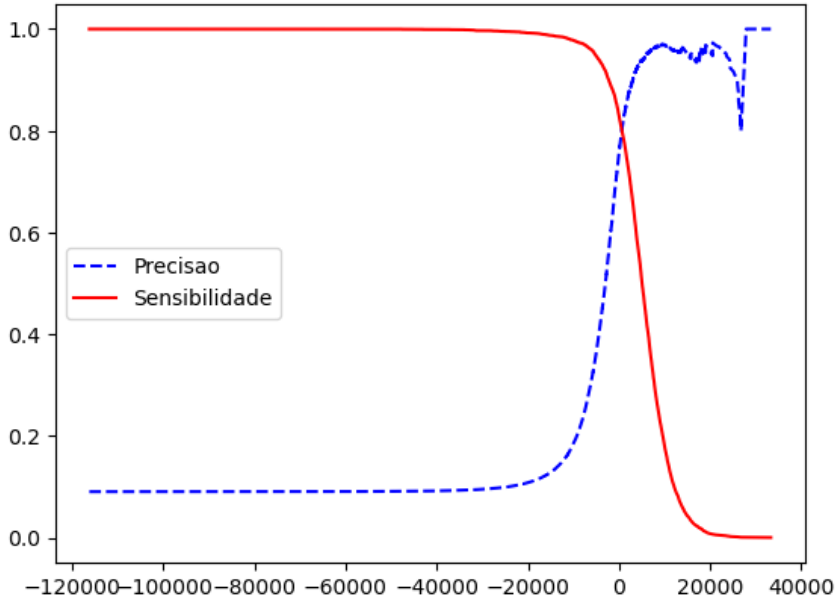
```
✓ 0.0s
2164.22030239]
 True]
False]
```

E qual limiar escolher? (para entendimento do pano de fundo...)

1. Usar cross_val_predict() para obter os scores de todas as instâncias de treinamento, com o método “decision_function”
2. Usar precision_recall_curve() para calcular precisão e revocação em todos os limiares possíveis
3. Plotar o gráfico PR/RC

A RELAÇÃO PRECISÃO/REVOCAÇÃO

```python
y_scores = cross_val_predict(model_sgd, X_train, y_train_5, cv=5, method='decision_function')

from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], 'b--', label='Precisao')
    plt.plot(thresholds, recalls[:-1], 'r-', label='Sensibilidade')
    plt.legend()

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



Para o limiar que dê uma precisão de 90%, qual a revocação associada?

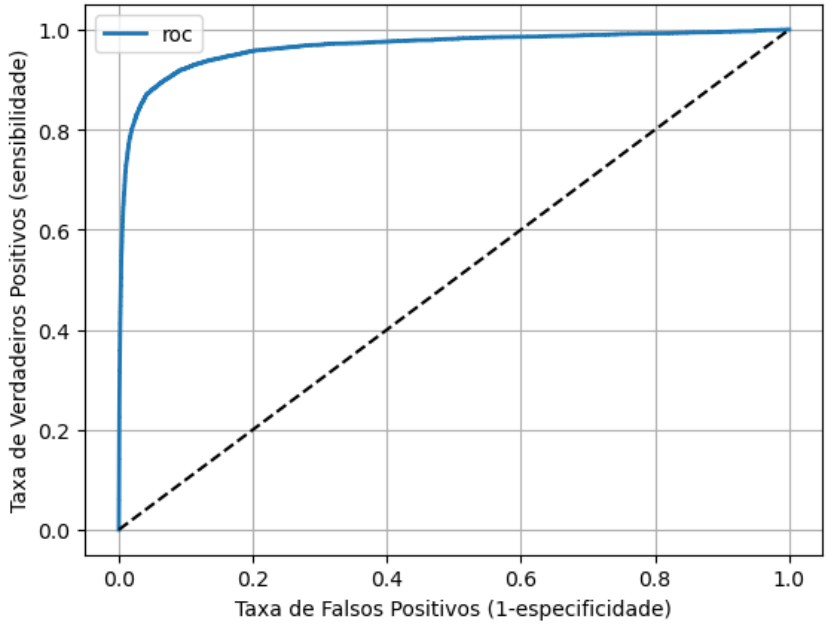
```python
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
print(threshold_90_precision) #3045

y_train_pred_90 = (y_scores >= threshold_90_precision)
from sklearn.metrics import precision_score, recall_score
print(precision_score(y_train_5, y_train_pred_90)) #90%
print(recall_score(y_train_5, y_train_pred_90)) #65%
```

✓ 0.0s

```
3045.9258227053647
0.9002016129032258
0.6589190186312488
```

A CURVA ROC (CARACTERÍSTICA DE OPERAÇÃO)

- Muito usada para avaliar classificadores binários
- É a curva entre a Taxa de Verdadeiros Positivos (TVP = Revocação = Sensibilidade) X Taxa de Falsos Positivos (TFP, taxa de instâncias negativas classificadas como positivas)
- TFP = 1 – TVN ou 1 - especificidade




```
from sklearn.metrics import roc_curve
tfp, tvp, thresholds = roc_curve(y_train_5, y_scores)

def plot_roc_curve(tfp, tvp, label='roc'):
    plt.plot(tfp,tvp,linewidth=2,label=label)
    plt.plot([0,1],[0,1],'k--')
    plt.legend()
    plt.grid()
    plt.xlabel('Taxa de Falsos Positivos (1-especificidade)')
    plt.ylabel('Taxa de Verdadeiros Positivos (sensibilidade)')

plot_roc_curve(tfp, tvp)
plt.show()
```

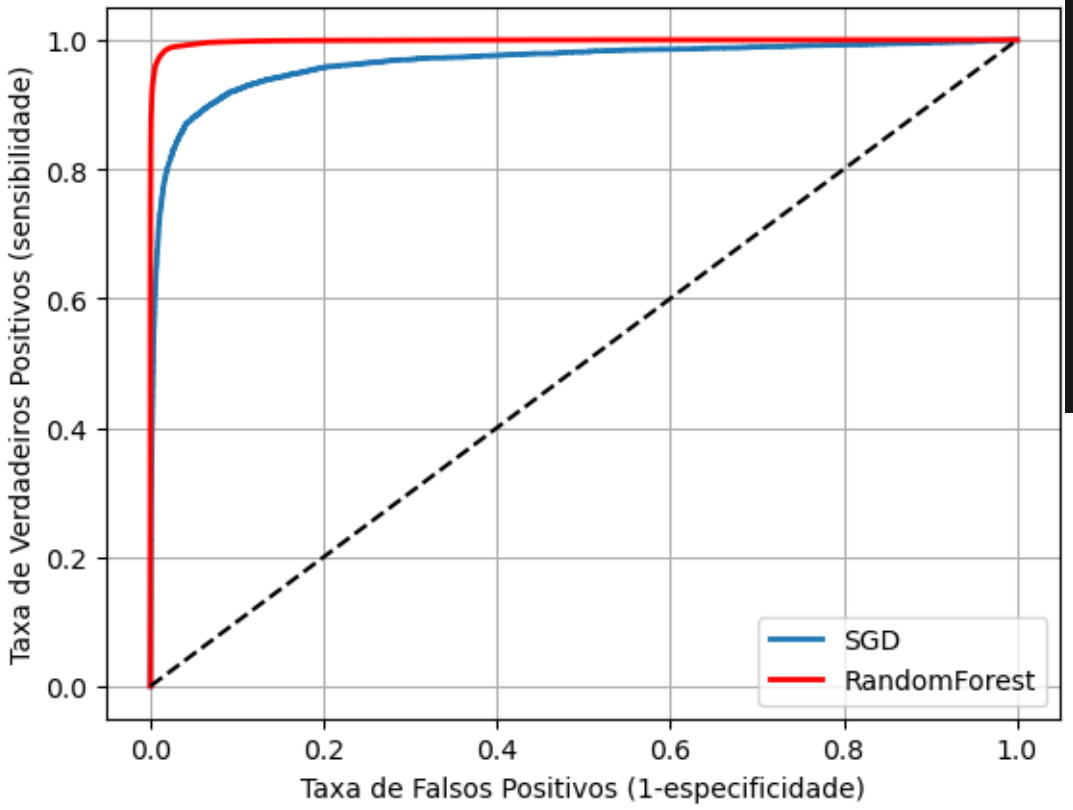
A área sob a curva (AUC) é uma forma de comparar classificadores: (quanto mais próximo de 1, melhor)

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores)
✓ 0.0s
0.9648211175804801
```

A CURVA ROC (CARACTERÍSTICA DE OPERAÇÃO)

- Comparando SGD X RandomForest para o classificador binário de 5s no MNIST

```python
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(random_state=42)

y_probas_rf = cross_val_predict(model_rf, X_train, y_train_5, cv=5, method='predict_proba')

y_scores_rf = y_probas_rf[:,1] # ou seja as probabilidades da classe positiva (5)
tfp_rf, tvp_rf, thresholds_rf = roc_curve(y_train_5, y_scores_rf)

def plot_roc_curve(tfp_1, tvp_1, tfp_2, tvp_2, label_1, label_2):
    plt.plot(tfp_1,tvp_1, linewidth=2, label=label_1)
    plt.plot(tfp_2,tvp_2, 'r', linewidth=2, label=label_2)
    plt.plot([0,1],[0,1],'k--')
    plt.legend()
    plt.grid()
    plt.xlabel('Taxa de Falsos Positivos (1-especificidade)')
    plt.ylabel('Taxa de Verdadeiros Positivos (sensibilidade)')

plot_roc_curve(tfp, tvp, tfp_rf, tvp_rf, 'SGD', 'RandomForest')
plt.show()
```



ROC AUC_RF = 0.9984

APERFEIÇOANDO O MODELO

- A escolha manual de hiperparâmetros é tediosa (encontrar uma combinação)
- No Scikit-learn pode-se usar o GridSearchCV ou o RandomizedSearchCV
- No GridSearch ele avalia TODAS as combinações por meio de validação cruzada
- Exemplo para o RandomForest

```
from sklearn.model_selection import GridSearchCV
hyperparam_grid = [{'n_estimators':[3,10,30],
                    'max_features':[2,4,6,8]},
                   {'bootstrap':[False], 'n_estimators':[3,10], 'max_features':[2,3,4]},]
model_rf = RandomForestRegressor()
model_rf_gs = GridSearchCV(model_rf, hyperparam_grid, cv=5,scoring='f1',return_train_score=True)
model_rf_gs.fit(X,y)
```

Testa no primeiro dict 3x4=12 combinações com o parâmetro bootstrap:True (default)
Depois testa no segundo dict 2x3=6 combinações com bootstrap:False
Testa portanto 12+6 combinações e cada rodada de 5 treinos com validação cruzada, ou seja, 18 x 5 = 90 rodadas de treinamento. Ao fim, os melhores parâmetros estão aqui:

```
model_rf_gs.best_params_ # se os melhores resultados forem com os limites superiores, pense em aumenta-los
model_rf_gs.best_estimator_ (melhor estimador)
cvres = model_rf_gs.cv_results_
for mean_f1, params in zip(cvres['f1'], cvres['params']):
    print(mean_f1, params)
```

APERFEIÇOANDO O MODELO

- Outro exemplo com KNN

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1,40,2), 'weights': ['uniform', 'distance'], 'p': [1, 2, 3]}
grid = GridSearchCV(KNeighborsClassifier(),param_grid, verbose = 3)#verbose indica a quantidade de detalhame
grid.fit(X_train,y_train)
```

Então você pode executar previsões neste objeto da grade com o conjunto de teste

```
grid_predictions = grid.predict(X_test)
```

```
dfGridSearch = pd.DataFrame(grid.cv_results_)
```

```
dfGridSearch.loc[dfGridSearch['rank_test_score'] == 1, :]
```

APERFEIÇOANDO O MODELO

- Se o espaço de pesquisa for grande é melhor usar busca randomizada
- Avalia determinado número de combinações aleatórias, selecionando um valor aleatório para cada hiperparâmetro por iteração (1000 iterações no Randomized explorar mais do que no GridSearch) e pode-se definir o número de iterações desejado para controlar melhor o custo computacional

```
from sklearn.model_selection import RandomizedSearchCV
hyperparam_grid = [{'n_estimators':[3,10,30],
                    'max_features':[2,4,6,8]},
                   {'bootstrap':[False], 'n_estimators':[3,10], 'max_features':[2,3,4]},]
model_rf = RandomForestRegressor()
model_rf_gs = RandomizedSearchCV(model_rf, hyperparam_grid, n_iter=10, scoring='f1', cv=5, return_train_score=True)
model_rf_gs.fit(X,y)
```