Aprendizado de Máquina

Classificação

Prof. Regis Pires Magalhães

regismagalhaes@ufc.br - http://bit.ly/ufcregis

GÉRON, Aurélien; **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow:** Conceitos, Ferramentas e Técnicas para a Construção de Sistemas Inteligentes. 2ª Ed. Alta Books, 2021.

PARTE I - Os conceitos básicos do aprendizado de máquina

1. O Cenário do Aprendizado de Máquina
2. Projeto de Aprendizado de Máquina Ponta a Ponta
3. **Classificação**
4. Treinando Modelos
5. Máquinas de Vetores de Suporte
6. Árvores de Decisão
7. Aprendizado Ensemble e Florestas Aleatórias (Bagging, Random Forests, Boosting, Stacking)
8. Redução de Dimensionalidade (PCA, Kernel PCA, LLE)
9. Técnicas de Aprendizado Não Supervisionado (Clusterização, Misturas de gaussianas)

PARTE II - Redes Neurais e Aprendizado Profundo

10. Introdução às Redes Neurais Artificiais com a Biblioteca Keras
11. Treinando Redes Neurais Profundas
12. Modelos Customizados e Treinamento com a Biblioteca TensorFlow
13. Carregando e Pré-processando Dados com a TensorFlow
14. Visão Computacional Detalhada das Redes Neurais Convolucionais
15. Processamento de Sequências Usando RNNs e CNNs
16. Processamento de Linguagem Natural com RNNs e Mecanismos de Atenção
17. Aprendizado de Representação e Aprendizado Gerativo com Autoencoders e GANs
18. Aprendizado por Reforço
19. Treinamento e Implementação de Modelos TensorFlow em Larga Escala

MNIST

- Conjunto de dados com 70 mil imagens pequenas de algarismos escritos à mão.
- Cada imagem é rotulada com o algarismo que a representa.
- Chamado de “hello world” do aprendizado de máquina.
- O Scikit-Learn fornece funções auxiliares para fazer o download de conjuntos de dados populares.

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details',
           'categories', 'url'])
```

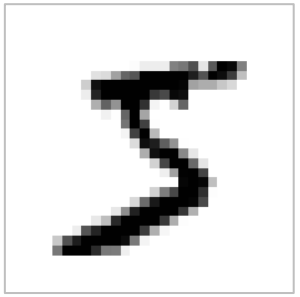
MNIST

```
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

- 70 mil imagens, cada uma com 784 características.
- Cada imagem tem 28 × 28 pixels.
- Cada característica representa a intensidade de um pixel, de 0 (branco) a 255 (preto).

MNIST

```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



```
>>> y[0]
'5'
```

A maioria dos algoritmos de ML espera números, logo, converteremos para inteiro:

```
>>> y = y.astype(np.uint8)
```

MNIST – Divisão treino / teste

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Treinando um Classificador Binário

- Simplificando tudo:
  - Tentar classificar somente um algarismo: 5

```
y_train_5 = (y_train == 5)  # True for all 5s, False for all other digits
y_test_5 = (y_test == 5)
```

- Usaremos:
  - O gradiente descendente estocástico (SGD)
    - Lida eficientemente com conjuntos de dados muito grandes.
      - Lida com instâncias de treinamento de forma independente, uma de cada vez, tornando-o adequado para o aprendizado online.

Treinando um Classificador Binário

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

```
>>> sgd_clf.predict([some_digit])
array([ True])
```

Medidas de Desempenho

- Calculando a acurácia com a validação cruzada

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

Medidas de Desempenho

- Implementando `cross_val_score`...

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565, and 0.96495
```

Medidas de Desempenho

- Criando um classificador “burro” (“dumb”), que diz que tudo é “não 5”:

```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        return self
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

- Qual a acurácia dele?

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.91125, 0.90855, 0.90915])
```

Medidas de Desempenho

- Isso acontece porque apenas cerca de 10% das imagens são “5”.
- Assim, caso sempre estime que uma imagem não é um 5, acertará cerca de 90% das vezes.
- Por isso a acurácia geralmente não é a medida de desempenho preferida para os classificadores.
  - Especialmente quando se manipula conjuntos de dados assimétricos (quando algumas classes são muito mais frequentes que outras).

Medidas de Desempenho

- Matriz de confusão
  - Preenchimento a partir da comparação entre predições e valores reais.

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

Matriz de Confusão

```
y = ['cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat',
     'dog', 'dog', 'dog', 'dog', 'dog', 'dog',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit']

y_pred = ['cat', 'cat', 'cat', 'cat', 'cat', 'dog', 'dog', 'dog',
     'cat', 'cat', 'dog', 'dog', 'dog', 'rabbit',
     'dog', 'dog', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit']
```

		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	5	2	0
	Dog	3	3	2
	Rabbit	0	1	11

Matriz de Confusão

		Actual class	
		Cat	**Non-cat**
Predicted class	**Cat**	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

Matriz de Confusão

```
from sklearn import metrics

y = ['cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat',
     'dog', 'dog', 'dog', 'dog', 'dog', 'dog',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit']

y_pred = ['cat', 'cat', 'cat', 'cat', 'cat', 'dog', 'dog', 'dog',
     'cat', 'cat', 'dog', 'dog', 'dog', 'rabbit',
     'dog', 'dog', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit',
     'rabbit', 'rabbit', 'rabbit', 'rabbit', 'rabbit']

cm = metrics.confusion_matrix(y, y_pred, labels=['cat', 'dog', 'rabbit'])
print(cm)
```

Predicted class

Actual class

```
     c   d   r
c [[ 5   3   0]
d  [ 2   3   1]
r  [ 0   2  11]]
```

		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	5	2	0
	Dog	3	3	2
	Rabbit	0	1	11

Acurácia

Predicted class

Actual class

```
     c   d   r
c [[ 5   3   0]
d  [ 2   3   1]
r  [ 0   2  11]]
```

Accuracy = (TP + TN) / Total

True Positive (TP) + True Negative (TN) = 19

Total = 27

Accuracy = 19 / 27 = 0.7037037037

```
metrics.accuracy_score(y, y_pred)
```

```
0.70370370370372
```

```
accuracy = np.sum(np.diagonal(cm)) / np.sum(cm)
print(accuracy)
```

```
0.703703703704
```

Medidas de Desempenho

- Precisão e revocação
  - Precisão = acurácia das predições positivas (TP).

$$\text{precision} = \frac{TP}{TP + FP}$$

  - Revocação ou Sensibilidade ou Taxa de Verdadeiros Positivos (TPR)
    - proporção de instâncias positivas que são detectadas corretamente pelo classificador.

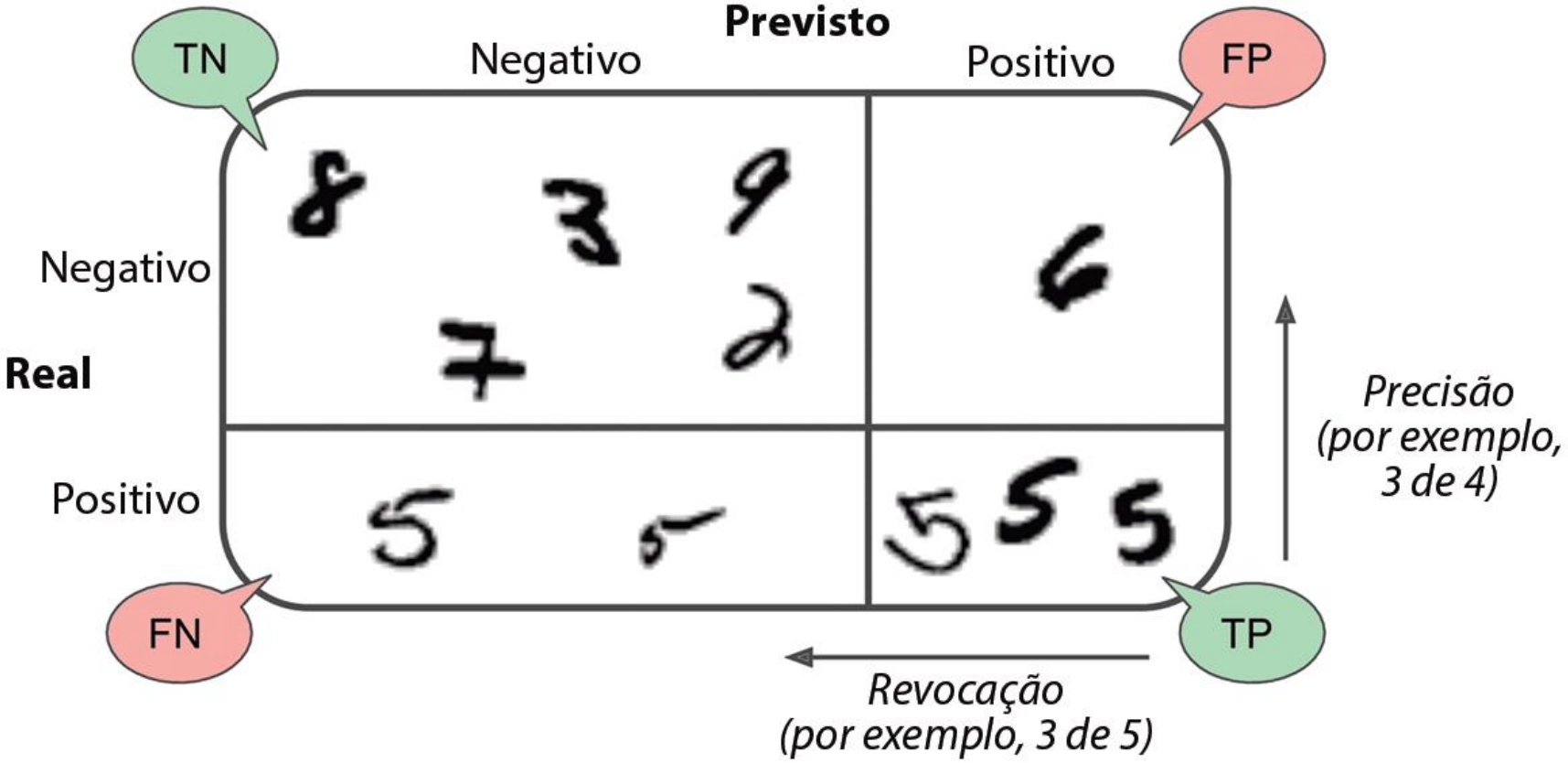
$$\text{recall} = \frac{TP}{TP + FN}$$

Medidas de Desempenho

- Precisão e revocação

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$



Medidas de Desempenho

- Precisão e revocação

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],        → FP
       [ 1325,  4096]])       → TP
```

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

Medidas de Desempenho

- Precisão e revocação

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

FN ← [1325, 4096]] → TP

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

Precisão ou Valor Preditivo Positivo

Predicted class

Actual class	c	d	r
c	5	3	0
d	2	3	1
r	0	2	11

Precision = TP / (TP + FP)

$TP_{cat} = 5$

$TP_{cat} + FP_{cat} = 5 + 2 = 7$

$Precision_{cat} = 5 / 7 = 0.7142857143$

```
metrics.precision_score(y, y_pred, average=None)
```

```
[ 0.71428571,  0.375     ,  0.91666667]
```

```
precision = cm[0,0] / np.sum(cm[:,0])
print(precision)
```

```
0.714285714286
```

Revocação / True Positive Rate / Sensitividade

Predicted class

Actual class

```
      c   d   r
c  [[ 5   3   0]
d   [ 2   3   1]
r   [ 0   2  11]]
```

Revocação

Recall = TP / (TP + FN)

$TP_{cat} = 5$

$TP_{cat} + FN_{cat} = 5 + 3 = 8$

$Recall_{cat} = 5 / 8 = 0.625$

```
metrics.recall_score(y, y_pred, average=None)
```

```
[ 0.625      ,  0.5        ,  0.84615385]
```

```
recall = cm[0,0] / np.sum(cm[0,:])
print(recall)
```

```
0.625
```

Medidas de Desempenho

- F1-Score
  - Combina a precisão e a revocação em uma única métrica.
  - Maneira simples de comparar dois classificadores.
  - É a média harmônica da precisão e revocação.
  - A média regular trata igualmente todos os valores.
  - Já a média harmônica dá mais importância aos valores mais baixos.
  - Assim, o classificador só obterá um score F1 alto se a revocação e a precisão forem altas.

Medidas de Desempenho

- F1-Score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

Medidas de Desempenho

- F1-Score
  - Favorece os classificadores que têm precisão e revocação semelhantes.
  - Mas em alguns contextos, você se preocupa basicamente com a precisão, e em outros, com a revocação.
  - Ex 1: detectar vídeos seguros para crianças
    - Prefere-se um classificador que rejeite muitos vídeos bons (baixa revocação) e mantenha apenas os seguros (**alta precisão** – foco na redução dos FPs).
    - Pode-se adicionar um humano para checar vídeos dados como seguros para crianças do classificador.

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Medidas de Desempenho

- Precisão x Revocação
  - Ex 2: detectar ladrões de lojas em imagens de vigilância
    - É bom que o classificador tenha somente 30% de precisão (baixa precisão), desde que tenha 99% de revocação (**alta revocação** – foco na redução dos FNs).
    - Os guardas de segurança receberão alertas falsos (FPs), mas quase todos os ladrões de lojas serão pegos.
  - Ex 3: empréstimo de dinheiro
  - Ex 4: exame médico sobre doença grave

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Relatório de Classificação

Predicted class

Actual class	c	d	r	s
c	5	3	0	8
d	2	3	1	6
r	0	2	11	13

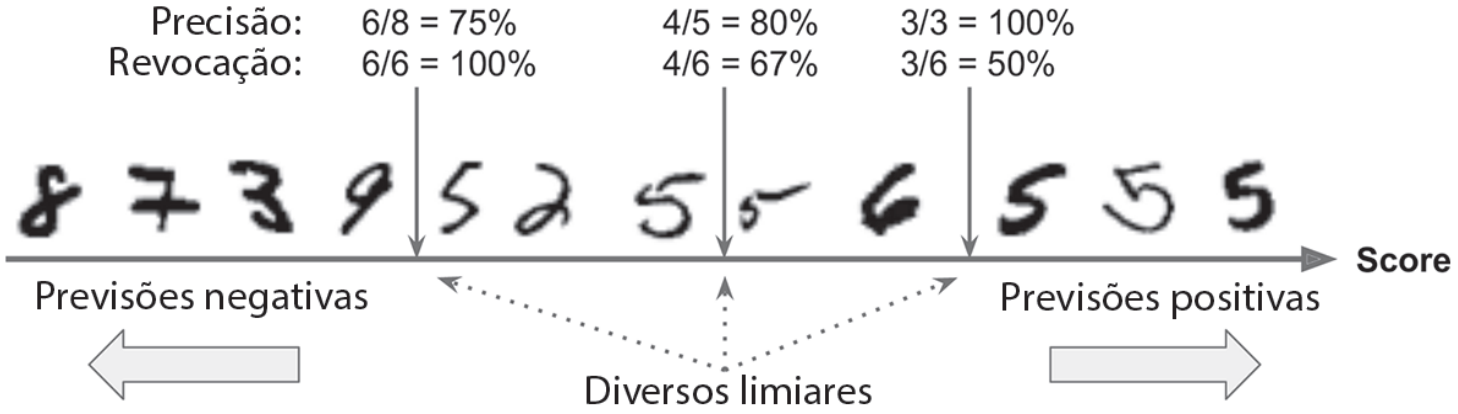
$$\text{Precision}_{avg} = (0.71 * 8 + 0.38 * 6 + 0.92 * 13) / 27 = 0.74$$

```
metrics.classification_report(y, y_pred)
```

	precision	recall	f1-score	support
cat	0.71	0.62	0.67	8
dog	0.38	0.50	0.43	6
rabbit	0.92	0.85	0.88	13
avg / total	0.74	0.70	0.72	27

Medidas de Desempenho

- Trade-off precisão/ revocação
  - Aumentar a precisão reduz a revocação e vice-versa.
  - Funcionamento do SGDClassifier
    - Para cada instância, ele calcula um score baseado em uma função de decisão.
    - Se o score for maior que um limiar (*threshold*), ele atribui a classe positiva, caso contrário, atribui a classe negativa.
    - Quanto maior o limiar, menor a revocação e maior (em geral) a precisão.



Medidas de Desempenho

- Trade-off precisão/ revocação
  - O Scikit não permite definir o limiar diretamente.
    - Mas deixa acessar os scores de decisão que ela emprega para fazer predições.

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2412.53175101])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])
```

  - O SGDClassifier utiliza um limiar igual a 0.
  - Aumentando o limiar:

```
>>> threshold = 8000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

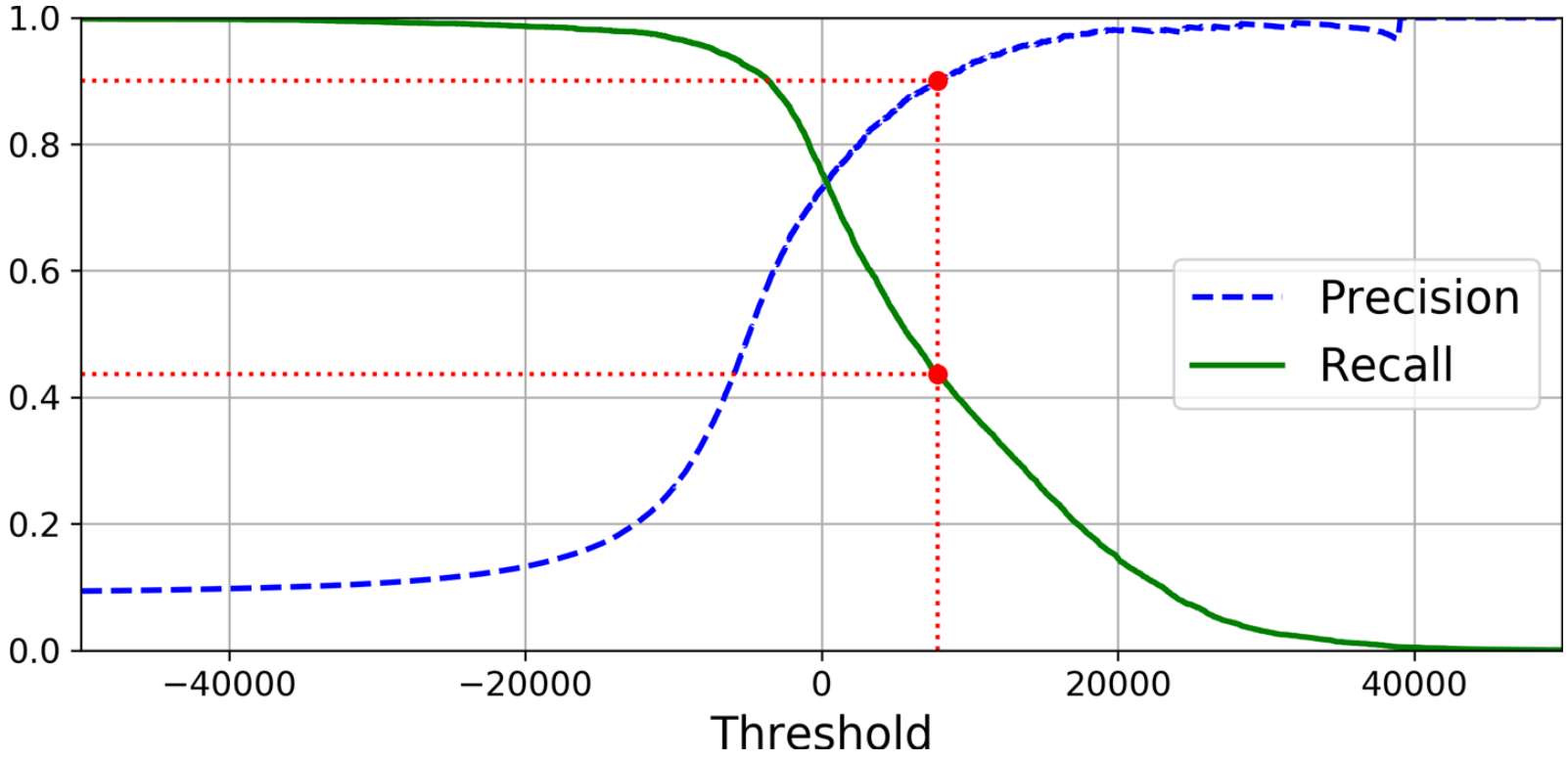
Medidas de Desempenho

- Trade-off precisão/ revocação
  - Decidindo qual limiar usar: curva de precisão/revocação

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             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="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid

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

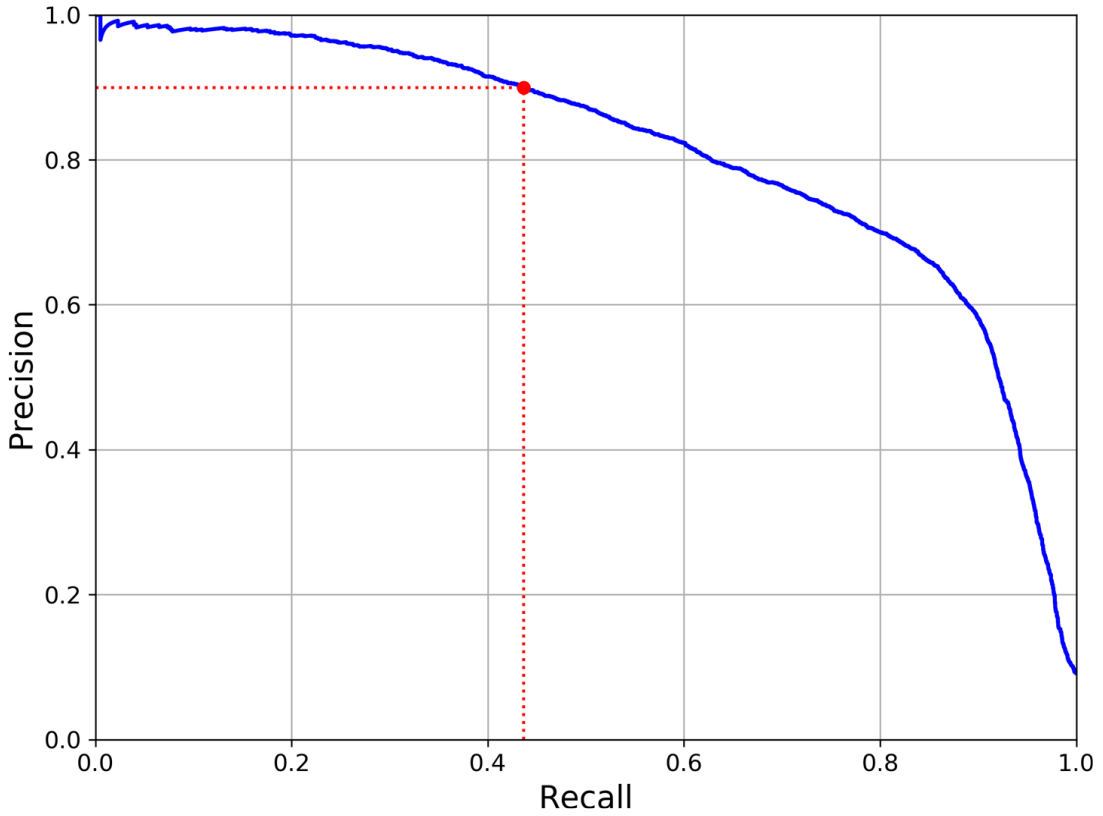
Medidas de Desempenho

- Trade-off precisão/ revocação
  - Decidindo qual limiar usar: curva de precisão/revocação



Medidas de Desempenho

- Trade-off precisão/ revocação
  - Decidindo qual limiar usar: precisão vs. revocação



Neste exemplo, a precisão começa a diminuir acentuadamente em torno de 80% da revocação.
Suponha que você quer 90% de precisão.

- Procure o limiar mais baixo que fornece pelo menos 90% de precisão.

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
```

Medidas de Desempenho

- Trade-off precisão/ revocação
  - Decidindo qual limiar usar: precisão vs. revocação

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816

y_train_pred_90 = (y_scores >= threshold_90_precision)

>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

Temos agora um classificador de precisão de 90%!
Mas um classificador de alta precisão não é lá muito útil, se sua revocação for muito baixa!

Medidas de Desempenho

- A curva ROC - Receiver Operating Characteristic - Curva de característica de operação.
- Semelhante à curva de precisão/ revocação, mas, em vez de plotar precisão vs. revocação, ela mostra a taxa de verdadeiros positivos (Revocação) em relação à taxa de positivos falsos (FPR).
  - FPR é a proporção de instâncias negativas classificadas incorretamente como positivas.
  - FPR = 1 — a taxa de verdadeiros negativos (TNR ou especificidade)

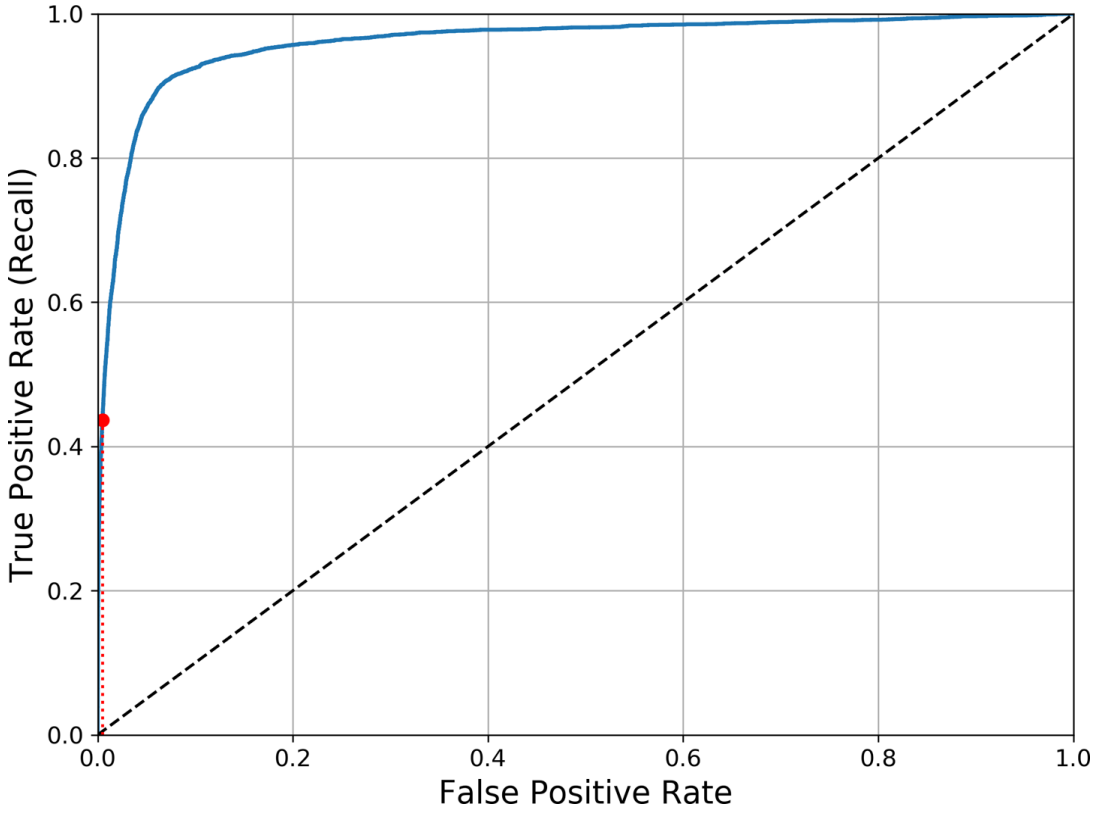
Medidas de Desempenho

- Curva ROC

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
    [...] # Add axis labels and grid

plot_roc_curve(fpr, tpr)
plt.show()
```

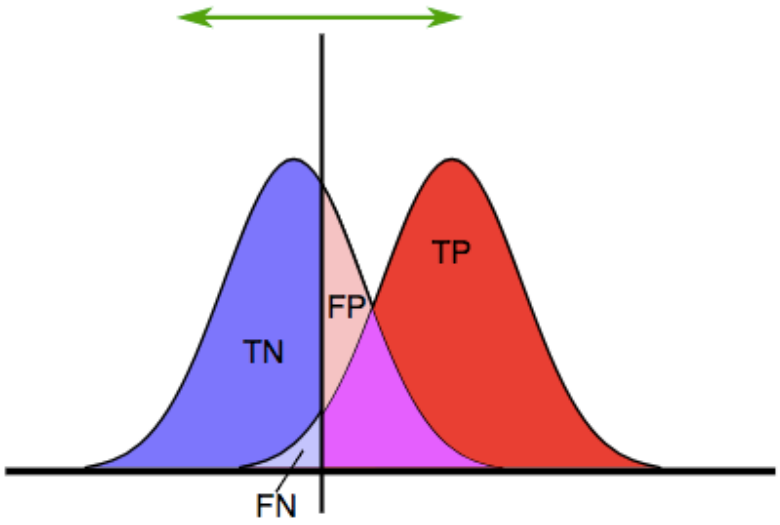
Medidas de Desempenho

- Curva ROC

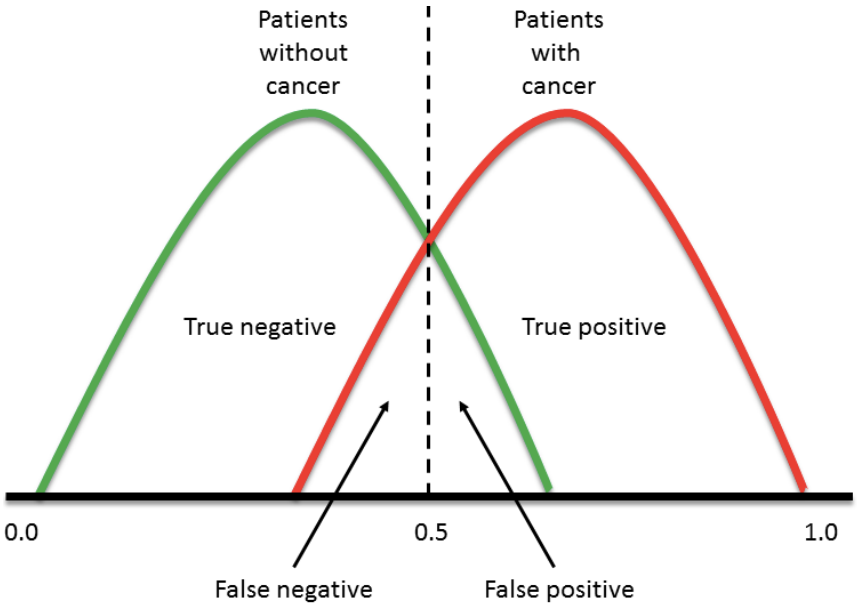


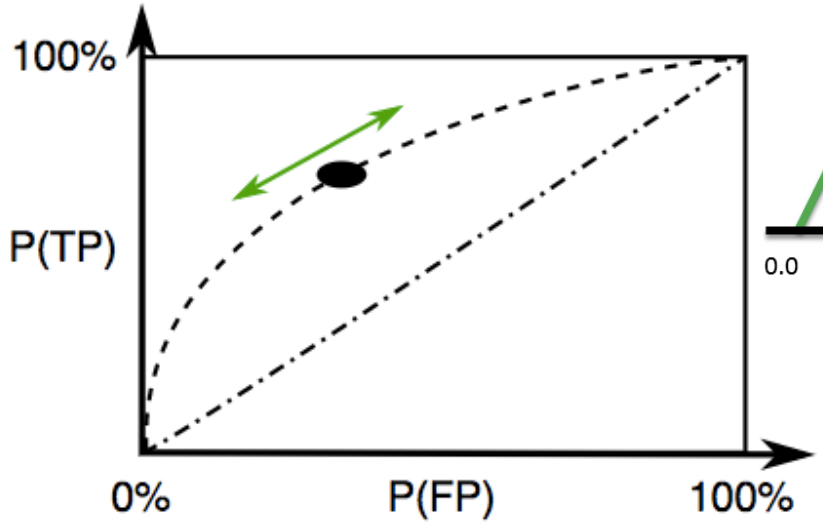
- Trade-off: quanto maior a revocação (TPR), mais falsos positivos (FPR) o classificador produz.
- A linha pontilhada representa a curva ROC de um classificador exclusivamente aleatório.
- Um bom classificador fica o mais distante possível dessa linha.

Curva ROC



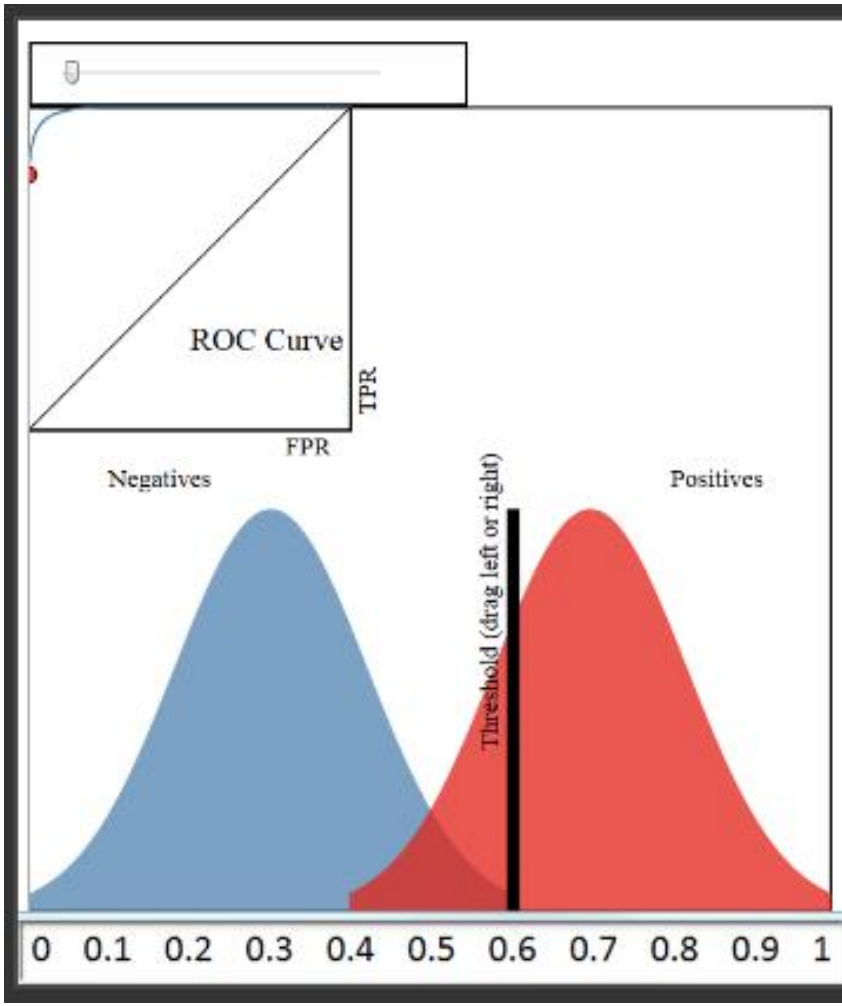
TP	FP
FN	TN





Understanding ROC curves

http://www.navan.name/roc/

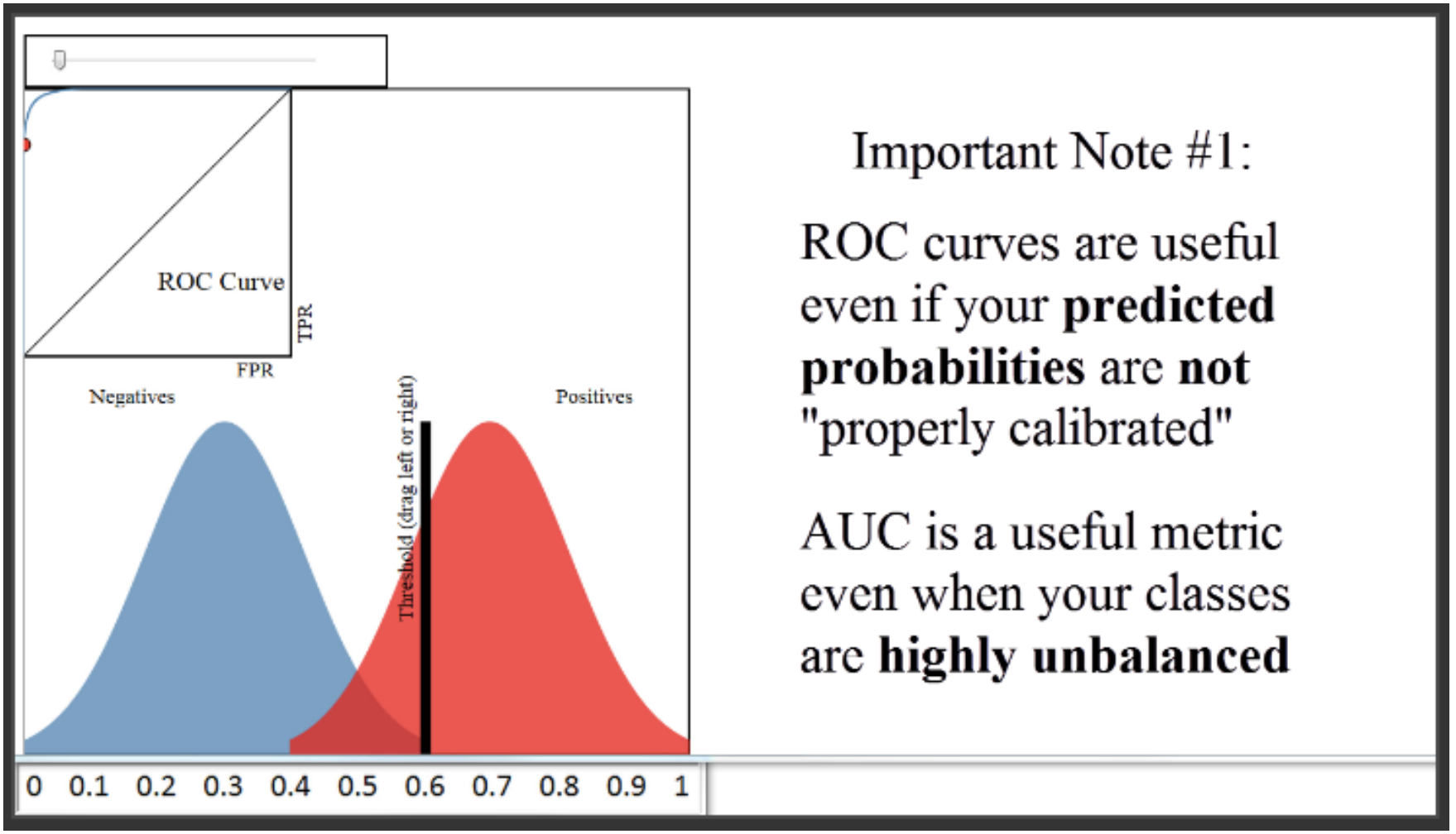


This diagram is **not** representative of most **real-world** problems:

1. Most problems **don't** have balanced classes
2. Predicted probabilites are unlikely to have a "smooth" distribution

Understanding ROC curves

http://www.navan.name/roc/



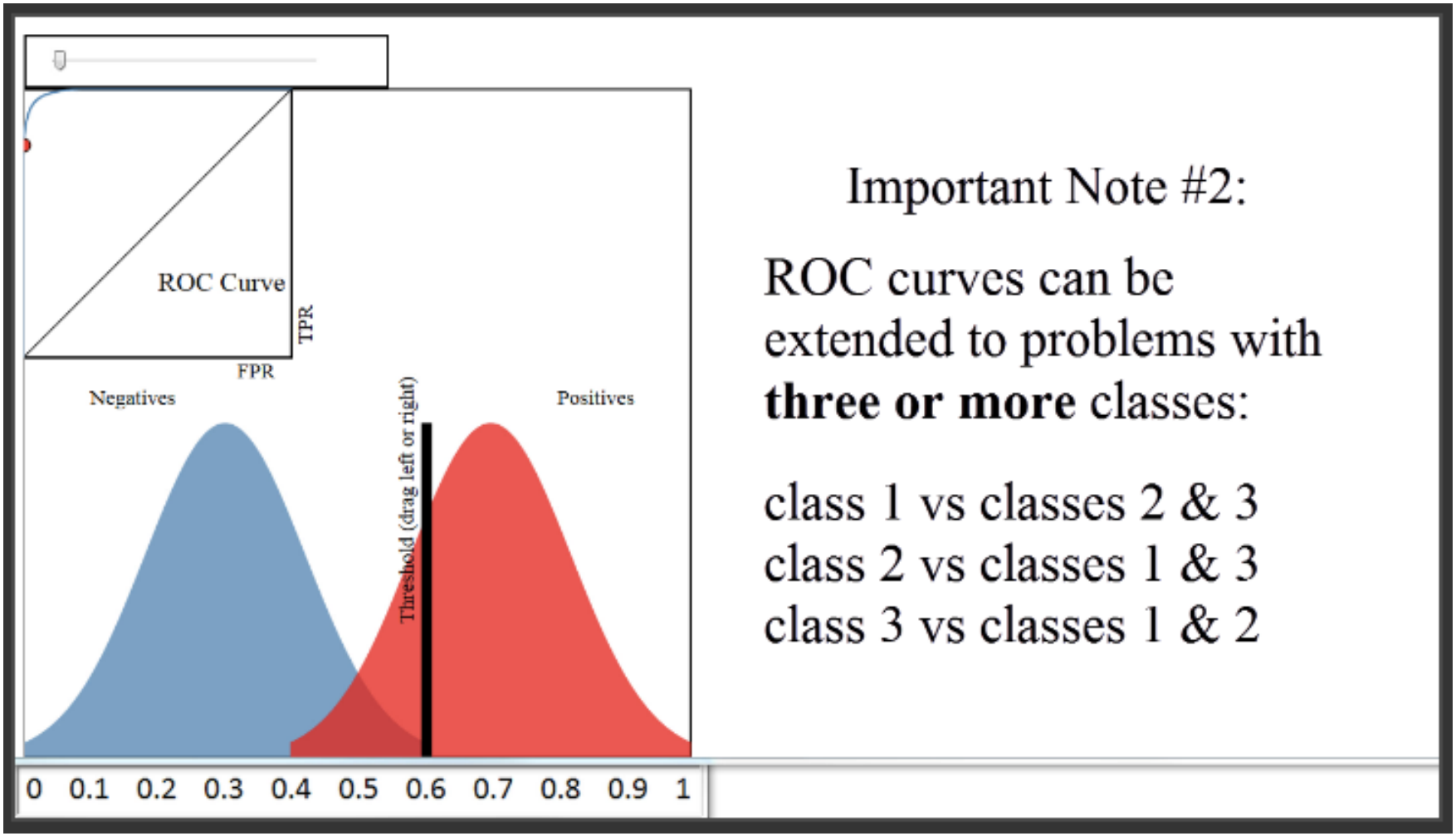
Important Note #1:

ROC curves are useful even if your **predicted probabilities** are **not** "properly calibrated"

AUC is a useful metric even when your classes are **highly unbalanced**

Understanding ROC curves

http://www.navan.name/roc/

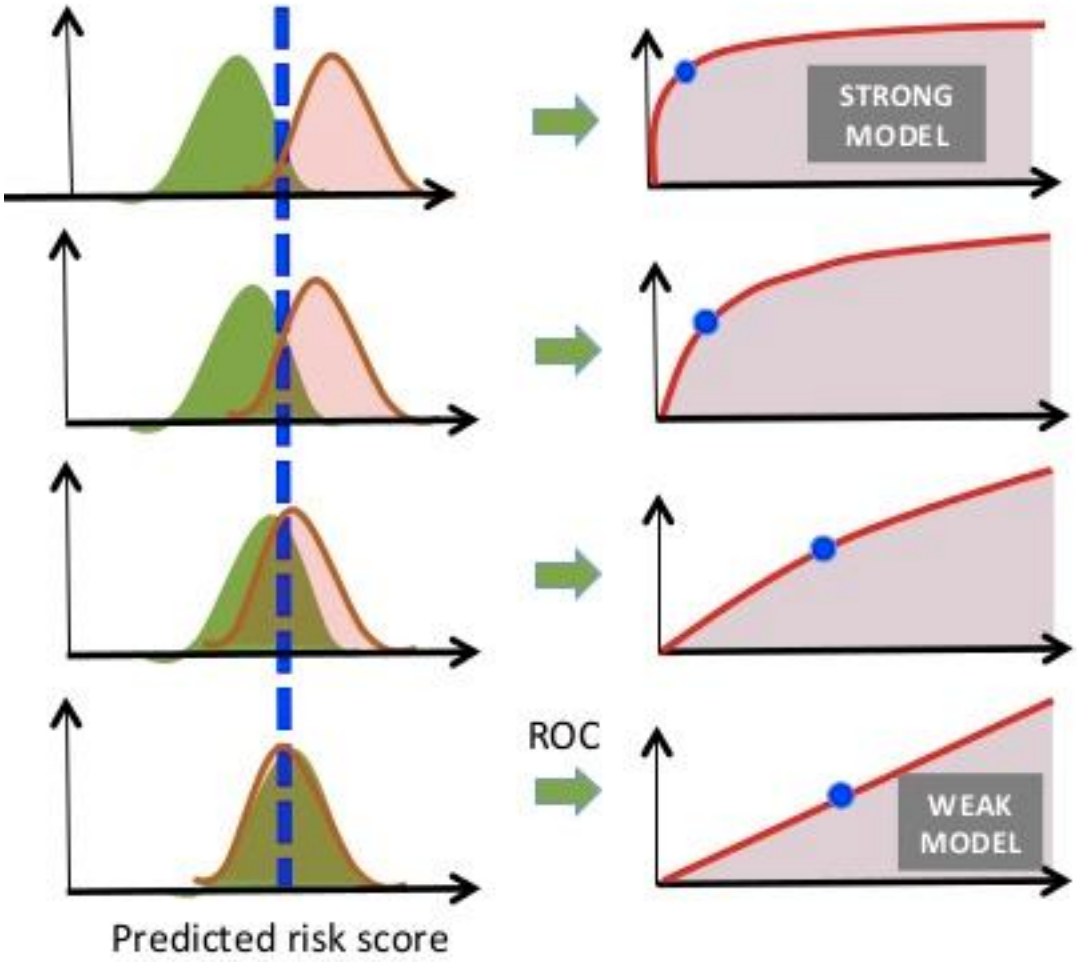


Important Note #2:

ROC curves can be extended to problems with **three or more** classes:

class 1 vs classes 2 & 3
class 2 vs classes 1 & 3
class 3 vs classes 1 & 2

Model Performance



Overlap is a measure of the model's ability to separate between success and failure.

With a strong model you can be confident of assigning a particular score to an outcome category.

With a weaker model, there is a large amount of overlap, so a particular score could mean that an outcome can be either good or bad with equal probability.

Medidas de Desempenho

- ROC AUC – Area Under Curve
  - Uma forma de comparar os classificadores é calcular a ROC AUC.
  - ROC AUC perfeita = 1
  - ROC AUC de classificador aleatório = 0.5

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9611778893101814
```

Medidas de Desempenho

- ROC AUC – Area Under Curve
  - Preferir a curva PR sempre que a classe positiva for rara ou quando os falsos positivos forem mais importantes do que com os falsos negativos.
  - Caso contrário, use a curva ROC. E
  - Exemplo: ao analisar a curva ROC anterior, você pode pensar que o classificador é realmente bom.
    - Isso ocorre basicamente porque existem poucos positivos (5s) em comparação aos negativos (não-5s).
    - Por outro lado, a curva PR deixa claro que o classificador tem espaço para melhorias (a curva pode estar mais próxima do canto superior esquerdo).

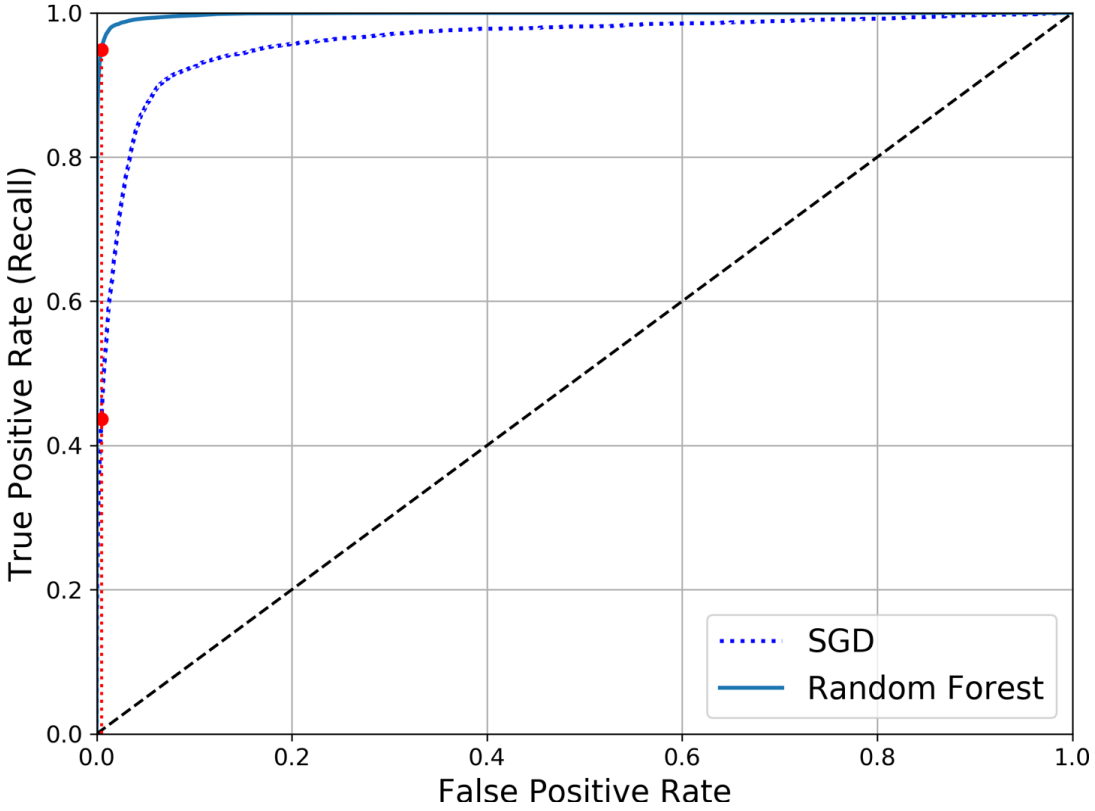
Medidas de Desempenho

- ROC AUC – Area Under Curve
  - O método predict_proba() retorna a probabilidade de que a instância especificada pertença à classe especificada.
    - Exemplo: 70% de probabilidade de que a imagem represente um 5.

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")

y_scores_forest = y_probas_forest[:, 1]   # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)

plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()
```

Medidas de Desempenho

- ROC AUC – Area Under Curve



Random Forest é superior ao classificador SGD, porque sua curva ROC está mais próxima do canto superior esquerdo e tem uma AUC maior.

```
>>> roc_auc_score(y_train_5, y_scores_forest)
0.998436731328145
```

Classificação Multiclasse

- Classificadores multiclasses ou multinomiais podem distinguir entre mais de duas classes.
- Alguns algoritmos (como classificadores SGD, classificadores de florestas aleatórias e classificadores naive Bayes) conseguem lidar com diversas classes de forma nativa.
- Outros (como classificadores de regressão logística ou máquina de vetores de suporte) são estritamente classificadores binários.
- Existem estratégias para fazer a classificação em diversas classes usando vários classificadores binários.

Classificação Multiclasse

- One-versus-the-Rest (OvR) (um contra todos [one-versus-all])
  - Treinar 10 classificadores binários, um para cada algarismo.
  - Quando quiser classificar uma imagem, você obtém o score de decisão de cada classificador para essa imagem e seleciona a classe cujo classificador gere a saída do maior score.
- One-versus-One [um contra um] (OvO)
  - Treinar um classificador binário para cada par de algarismos: um para distinguir 0s e 1s, outro para distinguir 0s e 2s, outro para 1s e 2s e assim sucessivamente.
  - Necessários $N \times (N - 1) / 2$ classificadores.
    - Para o MNIST: 45 classificadores!!!
    - Vantagem: cada classificador precisa ser treinado somente na parte do conjunto de treinamento para as duas classes que ele deve distinguir.

Classificação Multiclasse

- Para alguns algoritmos como o SVM, o OvO é preferido porque é mais rápido treinar muitos classificadores em pequenos conjuntos de treinamento do que treinar poucos classificadores em grandes conjuntos de treinamento.
- Entretanto, para a maioria dos algoritmos de classificação binária, prefere-se o OvR.
- O Scikit executa automaticamente o OvR ou o OvO, dependendo do algoritmo. No exemplo abaixo usa-se OvO com 45 classificadores:

```
>>> from sklearn.svm import SVC
>>> svm_clf = SVC()
>>> svm_clf.fit(X_train, y_train) # y_train, not y_train_5
>>> svm_clf.predict([some_digit])
array([5], dtype=uint8)
```

Classificação Multiclasse

- Ele retorna 10 scores por instância, sendo um score por classe.
- O score mais alto é de fato o que corresponde à classe 5.

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
>>> some_digit_scores
array([[ 2.92492871,  7.02307409,  3.93648529,  0.90117363,  5.96945908,
         9.5       ,  1.90718593,  8.02755089, -0.13202708,  4.94216947]])

>>> np.argmax(some_digit_scores)
5
>>> svm_clf.classes_
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
>>> svm_clf.classes_[5]
5
```

Classificação Multiclasse

- Pode-se forçar o Scikit a usar OvR ou OvO.
  - Basta usar OneVsOneClassifier ou OneVsRestClassifier.

```
>>> from sklearn.multiclass import OneVsRestClassifier
>>> ovr_clf = OneVsRestClassifier(SVC())
>>> ovr_clf.fit(X_train, y_train)

>>> ovr_clf.predict([some_digit])
array([5], dtype=uint8)
>>> len(ovr_clf.estimators_)
10
```

Classificação Multiclasse

- Alguns algoritmos não precisam de OvR ou OvO, pois sabem lidar diretamente com classificação multiclasse.
  - Nesse caso, o método decision_function() retorna um valor por classe.

```
>>> sgd_clf.fit(X_train, y_train)
>>> sgd_clf.predict([some_digit])
array([5], dtype=uint8)

>>> sgd_clf.decision_function([some_digit])
array([[-15955.22628, -38080.96296, -13326.66695,   573.52692, -17680.68466,
         2412.53175, -25526.86498, -12290.15705, -7946.05205, -10631.35889]])

>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.8489802 , 0.87129356, 0.86988048])

>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
>>> cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
array([0.89707059, 0.8960948 , 0.90693604])
```

Log Loss

- Logarithmic Loss or **Log Loss**, works by **penalising the false classifications**.
- It works well for multi-class classification.
- When working with Log Loss, the classifier must assign probability to each class for all the samples.

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

  - N samples
  - M classes
  - y_{ij} indicates whether sample i belongs to class j or not
  - p_{ij} indicates the probability of sample i belonging to class j

Log Loss

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

- In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class.
- Log Loss has no upper bound and it exists on the range [0, ∞).
- Log Loss nearer to 0 indicates higher accuracy.
- If the Log Loss is away from 0 then it indicates lower accuracy.
- In general, minimising Log Loss gives greater accuracy for the classifier.

Log loss, aka logistic loss or cross-entropy loss

```
sklearn.metrics.log_loss(y_true, y_pred, eps=1e-15,
normalize=True, sample_weight=None, labels=None)
```

- loss function used in (multinomial) logistic regression and extensions of it

Análise de Erro

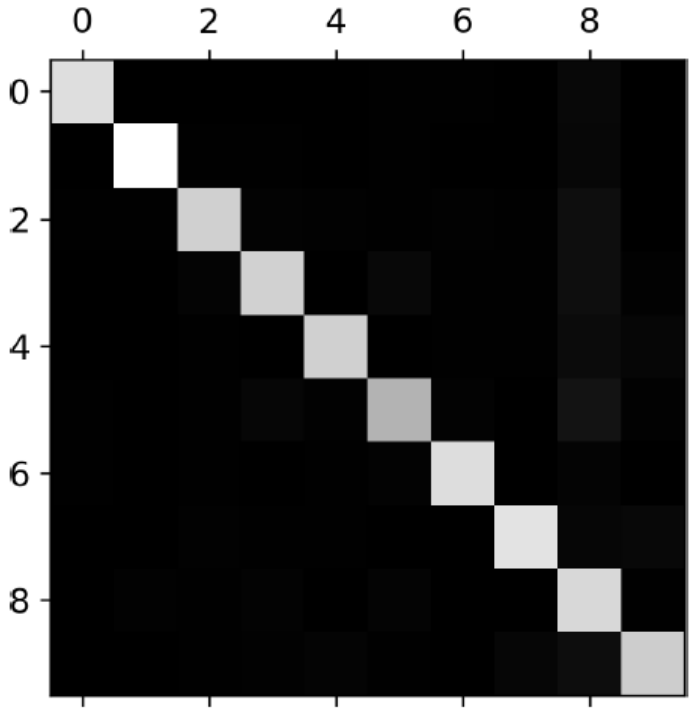
- Analise a matriz de confusão

```
>>> y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
>>> conf_mx = confusion_matrix(y_train, y_train_pred)
>>> conf_mx
array([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [   0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [  28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [  23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [  11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [  26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [  31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [  20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [  17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [  24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```

Análise de Erro

- Analise a matriz de confusão

```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
```



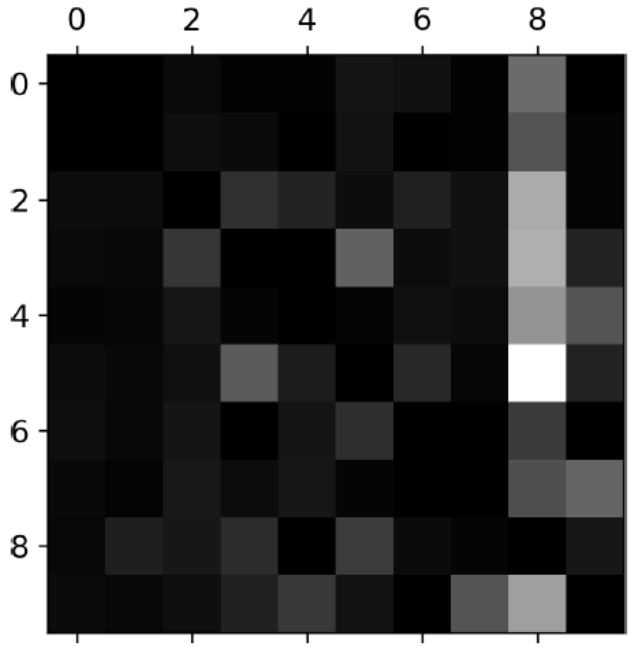
Análise de Erro

- Analise a matriz de confusão
  - Grande parte das imagens está na diagonal principal, o que significa que foram classificadas corretamente.
  - Os 5s são um pouco mais escuros do que os outros algarismos, o que pode indicar que há menos imagens de 5s no conjunto de dados ou que o classificador não funciona tão bem nos 5s quanto nos outros algarismos.

Análise de Erro

- Analise a matriz de confusão
  - Focar na representação dos erros

```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```



A coluna da classe 8 é bastante clara, o que indica que muitas imagens são classificadas erroneamente como 8s.
3s e 5s geralmente ficam confusos (em ambas direções).

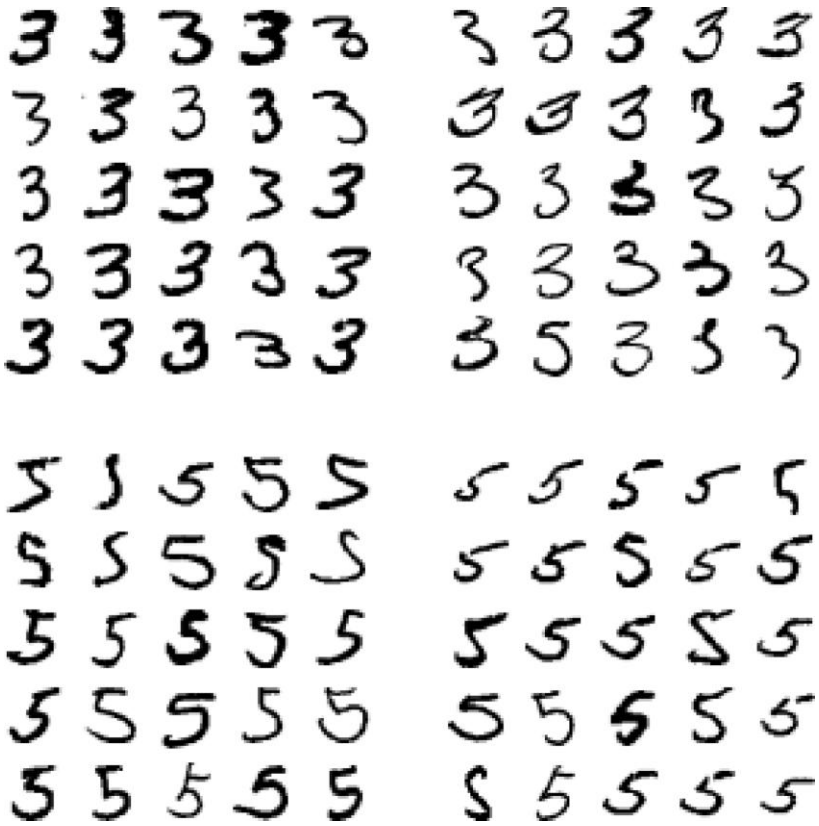
Análise de Erro

- Analise a matriz de confusão
  - Focar na representação dos erros

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```

Análise de Erro

- Analise a matriz de confusão
  - Focar na representação dos erros



Esse classificador é bastante sensível à mudança e à rotação da imagem. Portanto, uma forma de reduzir a confusão de 3/ 5 seria pré-processar as imagens para assegurar que elas estejam bem centralizadas, sem muita rotação.

Classificação Multirrótulo (Multilabel)

- Sistema de classificação que gera a saída de vários rótulos binários.
- Gera a saída de várias classes para cada instância.
- Exemplo:
  - Classificador de reconhecimento facial para reconhecer diversas pessoas na mesma imagem.
    - Classificador treinado para reconhecer três rostos: Alice, Bob e Charlie.
      - Foto de Alice e Charlie [1, 0, 1]: “Alice sim, Bob não, Charlie sim”).

Classificação Multirrótulo (Multilabel)

```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

Classificação Multirrótulo (Multilabel)

- Se você tiver muito mais fotos de Alice do que de Bob ou Charlie, convém dar mais peso ao score do classificador nas fotos de Alice.
- Uma opção simples é atribuir a cada rótulo um peso igual ao seu support (ou seja, o número de instâncias com esse rótulo-alvo).
- Para fazer isso, basta definir
  - average = "weighted"

Classificação Multioutput

- É uma generalização da classificação multirrótulo, em que cada rótulo pode ser multiclasseado, ou seja, pode ter mais de dois valores possíveis.
- É uma generalização da classificação multirrótulo, em que cada rótulo pode ser multiclasseado, ou seja, pode ter mais de dois valores possíveis.
- Exemplo:
  - Sistema que remove o ruído das imagens.
  - Tem como entrada uma imagem de algarismo com ruído e gerará uma saída de uma imagem de algarismo limpo.
  - A saída do classificador é multirrotulada (um rótulo por pixel) e cada rótulo pode ter vários valores (a intensidade do pixel varia de 0 a 255).

Classificação Multioutput

```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test

knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC