# 05. Avaliação de Modelos

IC 22/23

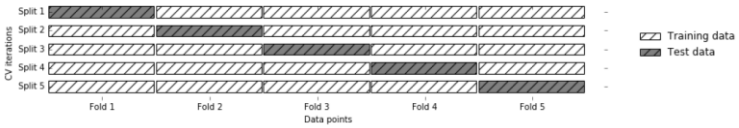
ISEC - Cpereira

# Metodologias

- Treino/Teste
  - Para avaliação de modelos supervisionados, segue-se habitualmente a seguinte metodologia:
    - Dividimos o conjunto de dados em dois conjuntos: conjunto de **treino** e conjunto de **teste;**
    - construímos um modelo no conjunto de treino, avaliamos no conjunto de teste, calculando a percentagem de amostras classificadas corretamente;
  - A avaliação no conjunto de teste permite medir a capacidade de generalização do modelo para dados novos, nunca antes vistos.
    - Não estamos interessados em quão bem o nosso modelo se ajusta ao conjunto de treino, mas sim em quão bem ele pode fazer previsões para dados que não foram observados durante o treino.

# Metodologias

- Cross-Validation – Validação cruzada
  - Método estatístico - mais fiável e completo do que usar apenas uma divisão em treino e teste.
    - » Os dados são divididos em pastas e vários modelos são treinados.
    - » A versão mais comum é a validação cruzada de k-fold, onde k define o número de pastas, definido pelo utilizador, usualmente um valor entre 5 e 10.
      - exemplo para k=5 [1]:

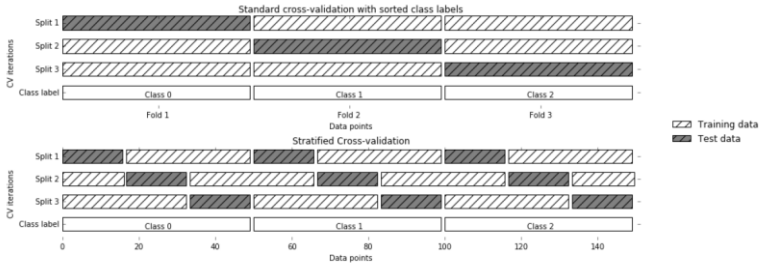



# Metodologias

- …
  - Vantagens
    - Cada amostra estará presente no conjunto de treino e também em uma das pastas para validação.
      - Assim, o modelo deve generalizar para todas as amostras no conjunto de dados, para que todas as métricas de validação cruzada (e a média) sejam elevadas.
    - Utilizamos todos dados de forma mais eficaz (todos são usados para treino, não apenas uma fração como na divisão treino/validação/teste).
    - Fornece informações sobre como o modelo é sensível à seleção do conjunto de dados de treino.
  - Desvantagens
    - Aumento do custo computacional - treinamos k modelos, em vez de um único modelo.
    - Não devolve propriamente um modelo específico, já treinado para posterior “deployment”.

# Metodologias

- …
  - "Stratified cross-validation"
    - Dividimos os dados de forma a que as proporções entre as classes sejam as mesmas em cada pasta e em todo o conjunto de dados. Exemplo [1]:

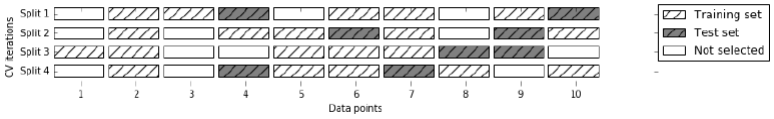



# Metodologias

- …
  - Outras Variantes
    - Leave-one-out
      - Caso especial de validação cruzada em que o número de pastas é igual ao número de amostras no conjunto de dados.
      - O algoritmo de aprendizagem é aplicado uma vez por cada instância, usando todas as outras instâncias como um conjunto de treino e a instância selecionada como teste.
        - » Trata-se de uma situação limite de cross-validation.

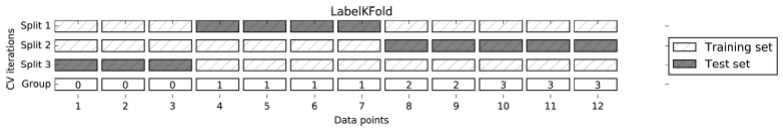
# Metodologias

- ...
  - ShuffleSplit
    - Seleciona aleatoriamente conjunto de treino e teste durante cada iteração. Cada divisão contém "*train_size*" amostras para treino "*test_size*" amostras (conjuntos disjuntos) para o teste. Esta divisão é repetida "n_iter" vezes.
      - » Exemplo [1], para *train_size=5, test_size=2, and n_iter=4:*




# Metodologias

- ...
  - **GroupKFold**
    - Aplicada quando existem grupos altamente relacionados.
      - » Por exemplo - um sistema para reconhecer emoções (classe) a partir de fotografias, com 100 pessoas, onde cada pessoa é capturada várias vezes, mostrando várias emoções. Pretende-se que a mesma pessoa não conste no treino e teste [1].



# Ajuste de híper-parâmetros

- Pesquisa em Grelha - Grid Search
  - Como ajustar os parâmetros de configuração do modelo (híper-parâmetros)?
    - Tentar todas as combinações possíveis dos parâmetros de interesse.
    - Exemplo de uma "grid" para um classificador SVC [1]:

| | C = 0.001 | C = 0.01 | … | C = 10 |
|---|---|---|---|---|
| gamma=0.001 | SVC(C=0.001, gamma=0.001) | SVC(C=0.01, gamma=0.001) | … | SVC(C=10, gamma=0.001) |
| gamma=0.01 | SVC(C=0.001, gamma=0.01) | SVC(C=0.01, gamma=0.01) | … | SVC(C=10, gamma=0.01) |
| … | … | … | … | … |
| gamma=100 | SVC(C=0.001, gamma=100) | SVC(C=0.01, gamma=100) | … | SVC(C=10, gamma=100) |

# Ajuste de híper-parâmetros

- …
  - Código [1]

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(
      X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

# Ajuste de híper-parâmetros

- …
  - Deve-se manter o conjunto de teste completamente isolado do treino - **usado apenas para a avaliação final**.
    - Todas as eventuais escolhas feitas com base na performance para o conjunto de teste "fará verter" informação para o modelo, o que não deve acontecer!


    - Podemos igualmente usar a validação cruzada para avaliar o desempenho de cada combinação de parâmetros.

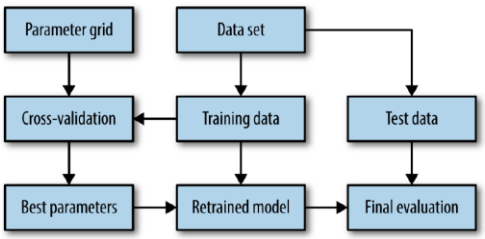
# Ajuste de híper-parâmetros

- …
  - Code [1]

```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```

# Ajuste de híper-parâmetros

- ...
  - O scikit-learn fornece a classe GridSearchCV
    - depois de encontrar o melhor conjunto de híper-parâmetros usando cross-validation, treina novamente o modelo para todo o conjunto de treino [1]:




# Ajuste de híper-parâmetros

- ...
  - Code [3]

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC()
>>> clf = GridSearchCV(svc, parameters)
>>> clf.fit(iris.data, iris.target)
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
>>> sorted(clf.cv_results_.keys())
['mean_fit_time', 'mean_score_time', 'mean_test_score',...
 'param_C', 'param_kernel', 'params',...
 'rank_test_score', 'split0_test_score',...
 'split2_test_score', ...
 'std_fit_time', 'std_score_time', 'std_test_score']
```

# Métricas de Avaliação

- Consideremos problemas de Classificação Binária
  - Conjuntos de dados não balanceados
    - Uma das duas classes é muito mais frequente que a outra.
      - Este acontecimento é bastante comum no mudo real. Exemplos (*spam filter, intrusion detection, diagnosis*, …)
    - Neste caso a "accuracy" – taxa de acerto será uma boa métrica?
      - Se 99% dos dados for da classe positiva, uma taxa de acerto de 99% pode não significar qualquer aprendizagem.

# Métricas de Avaliação

- …
  - Matriz de Confusão

| | | Predicted values | | |
|---|---|---|---|---|
| | | **Positive** | **Negative** | **Totals** |
| Actual Values | **Positive** | TP | FN | **P = (TP + FN ) = Actual Total Positives** |
| | **Negative** | FP | TN | **N = (FP + TN ) = Actual Total Negatives** |
| | **Totals** | Predicted Total Positives | Predicted Total Negatives | |

- Taxa de acerto:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

# Métricas de Avaliação

- ...
  - Existem várias outras formas de resumir a matriz de confusão, sendo as mais comuns as métricas de "precisão" e "recall".
    - Precisão
      - » Mede quantas das amostras previstas como positivas são realmente positivas.
        - usada como uma métrica de desempenho quando o objetivo é limitar o número de falsos positivos

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

# Métricas de Avaliação

- ...
  - Recall (sensitivity ou True Positive Rate)
    - mede quantas das amostras positivas são "capturadas" pelas previsões positivas.
      - Usada quando precisamos de identificar todas as amostras positivas; ou seja, quando é importante evitar falsos negativos.

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

      - » Considere um problema de diagnóstico medico. Qual a melhor métrica para avaliação de desempenho?
    - Existe um "trade-off" entre otimizar "recall" e otimizar a "precisão"!

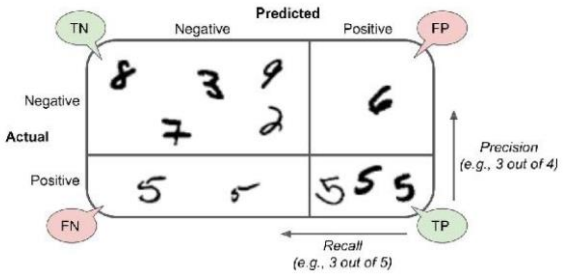
# Métricas de Avaliação

- ...
  - f-score
    - Uma forma de resumir o trade-off entre precisão e *recall* é a medidad "f-score" ou "f-measure", que representa a média harmónica entre as duas métricas
      - » Como tem em consideração a precisão e a *recall*, é mais adequada do que apenas a precisão para conjuntos de dados não balanceados

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

# Métricas de Avaliação

- ...
  - Exemplo [2]:

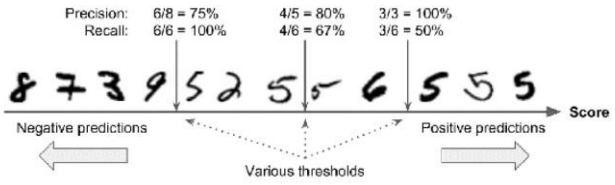


# Métricas de Avaliação

- ...
  - Considerar a "incerteza" na predição
    - a maioria dos classificadores fornece uma função de decisão ou um método de previsão de probabilidade para avaliar os graus de certeza sobre as previsões.
      - Como padrão, o "threshold"de 0,5 significa que se o modelo tiver mais de 50% de "certeza" de que um ponto é da classe positiva, ele será classificado como tal.
      - Aumentar este limite significa que o modelo precisa estar mais confiante para tomar uma decisão positiva.
    - Alterar o limite que é usado para tomar uma decisão de classificação é uma forma de ajustar o compromisso entre precisão e recall para um determinado classificador

# Métricas de Avaliação

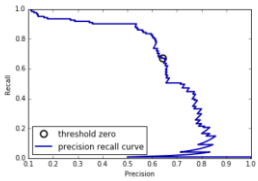
- ...
  - Este exemplo representa alguns dígitos posicionados da pontuação (probability_score) mais baixa à esquerda até a pontuação mais alta à direita [2]:



# Métricas de Avaliação

- Curva de Precisão-Recall
  - Definir um ponto de operação – por exemplo colocar como requisito um classificador com 90% de recall.
    - sklearn.metrics
      - » O threshold de zero é valor por defeito para a função de decisão do classificador.
      - » Qual a precisão para um recall de 90%?

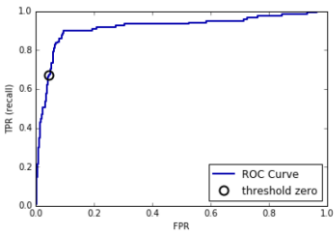
```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(
    y_test, svc.decision_function(X_test))
```




# Métricas de Avaliação

- ...
  - Curva de ROC - *Receiver operating characteristics* e AUC
    - Compromisso entre *false positive rate* (FPR) e *true positive rate* (TPR) – *recall*
      - » A curva ideal é próxima do canto superior esquerdo - recall elevado mantendo um "false positive rate" baixo [1]:

$$\text{FPR} = \frac{\text{FP}}{\text{FP+TN}}$$



# Métricas de Avaliação

- ...
  - Métricas para problemas “Multiclasse”
    - Digits dataset [1]

```
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

**Out[63]:**

```
Accuracy: 0.953
Confusion matrix:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

# Métricas de Avaliação

- ...

**In[65]:**

```
print(classification_report(y_test, pred))
```

**Out[65]:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 37 |
| 1 | 0.89 | 0.91 | 0.90 | 43 |
| 2 | 0.95 | 0.93 | 0.94 | 44 |
| 3 | 0.90 | 0.96 | 0.92 | 45 |
| 4 | 0.97 | 1.00 | 0.99 | 38 |
| 5 | 0.98 | 0.98 | 0.98 | 48 |
| 6 | 0.96 | 1.00 | 0.98 | 52 |
| 7 | 1.00 | 0.94 | 0.97 | 48 |
| 8 | 0.93 | 0.90 | 0.91 | 48 |
| 9 | 0.96 | 0.94 | 0.95 | 47 |
| avg / total | 0.95 | 0.95 | 0.95 | 450 |

# Métricas de Avaliação

- ...
  - Versão multiclasse de “f-score”:
    - calcular o “f-score” por classe, com essa classe sendo a classe positiva e as outras classes constituindo as classes negativas. Em seguida, essas pontuações por classe são combinadas usando uma das seguintes estratégias:
      - » "macro" averaging - a média das pontuações por classe.
      - » "weighted" averaging - a média das pontuações por classe ponderada pelo seu suporte.
      - » "micro" averaging - calcula o número total de falsos positivos, falsos negativos e verdadeiros positivos em todas as classes

# Referências


- [1] Müller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.".
- [2] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.
- [3] https://scikit-learn.org