# Machine Learning Package

*Portfolio de algoritmos de Machine Learning*

# Sumário

- Um modelo consiste numa função que estima um valor ou classe para cada exemplo/amostra tendo como base as *features*.
- No nosso portefólio, os modelos podem seguir a estrutura de um ***Model***.
- Arquitetura de um ***Model*** :
  - Parâmetros – conjunto de parâmetros definidos pelo utilizador
  - Parâmetros estimados – conjunto de parâmetros/atributos estimados a partir dos dados
  - Fit – método responsável por estimar parâmetros a partir dos dados
  - Predict – método responsável por prever valor ou classe para novas amostras
  - Score – método responsável por calcular a métrica de erro

# Sumário

- Machine learning workflow:
  1. Divisão do dataset em dataset de treino e teste (previsão)
  2. Treino do modelo com o dataset de treino
  3. Previsão usando o modelo treinado e o dataset de teste

# Datasets

- Os datasets estão disponíveis em:
  - https://www.dropbox.com/sh/oas4yru2r9n61hk/AADpRunbqES44W49gx9deRN5a?dl=0

# *model_selection* sub-package

- Cria o sub-package *model_selection e* adiciona o módulo chamado *split.py.*

- *def train_test_split*
  - assinatura/argumentos:
    - dataset – dataset para dividir em treino e teste
    - test_size – tamanho do dataset de test (e.g., 20%)
    - random_state – seed para gerar permutações
  - ouput esperado:
    - Um tuplo com dataset de treino e dataset de teste
  - algoritmo:
    - Gera permutações usando o *np.random.permutation*
    - Infere o número de amostras no dataset de teste e treino
    - Seleciona treino e teste usando as permutações

# *metrics* sub-package

- Cria o sub-package *metrics* adiciona o módulo chamado *accuracy.py.*
- *def accuracy*
  - assinatura/argumentos:
    - y_true – valores reais de Y
    - Y_pred – valores estimados de Y
  - ouput esperado:
    - O valor do erro entre *y_true* e *y_pred*
  - algoritmo:
    - Calcula a o erro seguindo a formula da accuracy:
    - (VN+VP) / (VN+VP+FP+FN)

# Objeto *KNNClassifier*

- Na pasta *neighbors*, adiciona o modulo *knn_classifier.py* que deve conter o objeto *KNNClassifier.*
- O algoritmo *k-nearest neighbors* estima a classe para uma amostra tendo como base os k exemplos mais semelhantes.
- *class KNNClassifier:*
  - Parâmetros:
    - k – número de k exemplos a considerar
    - distance – função que calcula a distância entre amostra e as amostras do dataset de treino
  - Parâmetros estimados:
    - dataset – armazena o dataset de treino
  - Métodos:
    - fit – armazena o dataset de treino
    - predict – estima a classe para uma amostra tendo como base os k exemplos mais semelhantes
    - score – calcula o erro entre as classes estimadas e as reais

# Objeto *KNNClassifier*

- *KNNClassifier.fit*:
  - Assinatura:
    - Input: dataset – dataset de treino
    - Output: self – ele mesmo - KNNClassifier
  1. Armazena o dataset de treino
     Resultado: dataset

- *KNNClassifier.predict*:
  - Assinatura:
    - Input: dataset – dataset de test
    - Output: previsões – um array de classes estimadas para o dataset de teste (y_pred)
  1. Calcula a distância entre cada amostra e as várias amostras do dataset de treino
  2. Obtém os indexes dos k exemplos mais semelhantes (menor distância)
  3. Usa os indexes anteriores para obter as classes correspondentes em Y (e.g., *self.dataset.y[indexes]*)
  4. Obtém a classe mais comum (c/ maior frequência) nos k exemplos
  5. Aplica o passo 1, 2, 3, e 4 a todas as amostras do dataset de teste

# Objeto *KNNClassifier*

- *KNNClassifier.score*:
  - Assinatura:
    - Input: dataset – dataset de test
    - Output: erro – calculo do erro entre as previsões e os valores reais
  1. Obtém previsões
  2. Calcula a accuracy entre os valores reais e as previsões

# Avaliação

- Exercício 3: Implementar o *KNNRegressor* com RMSE
  - 3.1) Adiciona a métrica RMSE (RMQE em português) ao sub-package *metrics*. Deves criar um módulo chamado *rmse.py.*
  - 3.2) Considera a estrutura da função *rmse* apresentada no diapositivo seguinte
  - 3.2) Adiciona o objeto *KNNRegressor* ao sub-package *neighbors*. Deves criar um módulo chamado *knn_regressor.py* para implementar este objeto.
  - 3.3) Considera a estrutura do objeto *KNNRegressor* apresentada no diapositivo seguinte.
  - 3.4) Podes testar o objeto *KNNRegressor* num jupyter notebook usando o dataset cpu.csv (regressão)

# *rmse* função

- No sub-package *metrics* adiciona o módulo chamado *rmse.py* que deve conter a seguinte função:

- *def rmse*
  - assinatura/argumentos:
    - y_true – valores reais de Y
    - Y_pred – valores estimados de Y
  - ouput esperado:
    - O valor do erro entre *y_true* e *y_pred*
  - algoritmo:
    - Calcula a o erro seguindo a formula da RMSE (RMQE em português):
    - $\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_{true_i} - y_{pred_i})^2}{N}}$
    - N representa o número de amostras
    - y_true é a variável y_true
    - y_pred é a variável y_pred

# Objeto *KNNRegressor*

- Na pasta *neighbors*, adiciona o modulo *knn_regressor.py* que deve conter o objeto *KNNRegressor.*
- O algoritmo *do KNNRegressor* é semelhante ao *KNNClassifier*. No entanto, o *KNNRegressor* é indicado para problemas de regressão. Portanto, este estima um valor médio dos k exemplos mais semelhantes.
- *class KNNClassifier:*
  - Parâmetros:
    - k – número de k exemplos a considerar
    - distance – função que calcula a distância entre amostra e as amostras do dataset de treino
  - Parâmetros estimados:
    - dataset – armazena o dataset de treino
  - Métodos:
    - fit – armazena o dataset de treino
    - predict – estima a classe para uma amostra tendo como base os k exemplos mais semelhantes
    - score – calcula o erro entre as classes estimadas e as reais

# Objeto *KNNRegressor*

- *KNNRegressor.fit*:
  - Assinatura:
    - Input: dataset – dataset de treino
    - Output: self – ele mesmo - KNNClassifier
  1. Armazena o dataset de treino
     Resultado: dataset

- *KNNRegressor.predict*:
  - Assinatura:
    - Input: dataset – dataset de test
    - Output: previsões – um array de classes estimadas para o dataset de teste (y_pred)
  1. Calcula a distância entre cada amostra e as várias amostras do dataset de treino
  2. Obtém os indexes dos k exemplos mais semelhantes (menor distância)
  3. Usa os indexes anteriores para obter os valores correspondentes em Y (e.g., *self.dataset.y[indexes]*)
  4. Calcula a média dos valores obtidos no passo 3.
  5. Aplica o passo 1, 2, 3, e 4 a todas as amostras do dataset de teste

# Objeto *KNNClassifier*

- *KNNRegressor.score*:
  - Assinatura:
    - Input: dataset – dataset de test
    - Output: erro – calculo do erro entre as previsões e os valores reais
  1. Obtém previsões
  2. Calcula a ***rmse*** entre os valores reais e as previsões