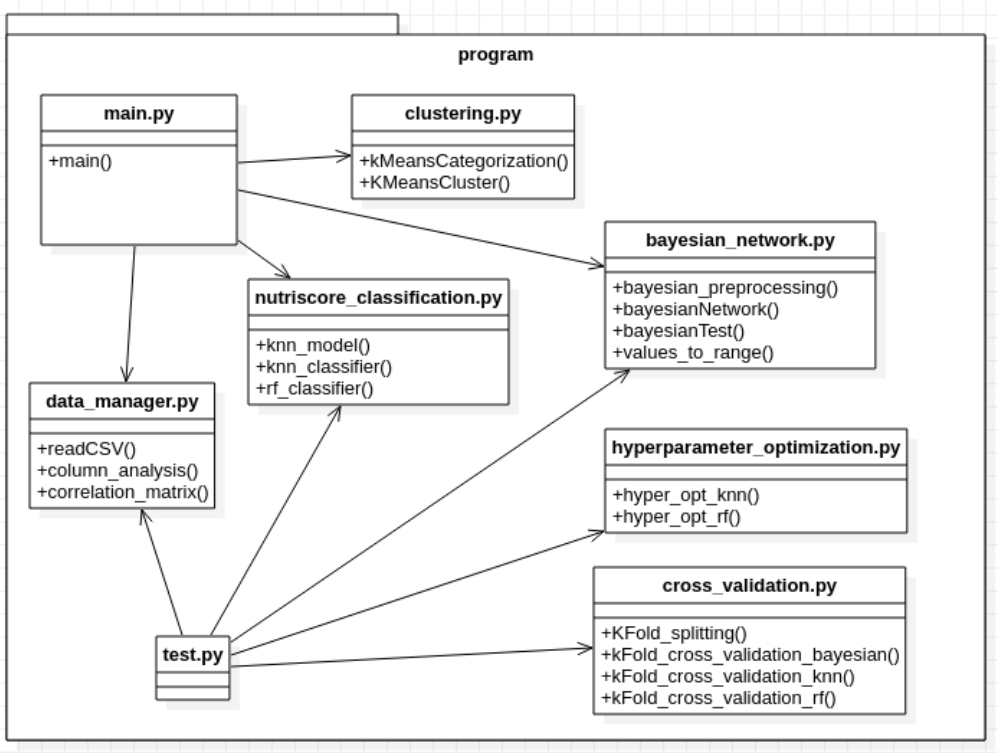
Food_Machine_Learning project [Pierpaolo Ventrella, Giuseppe Sancesario]



La figura mostra l'utilizzo dei vari moduli da parte dei moduli main e del test, con l'elenco dei metodi ad ognuno associati.

Di seguito sono riportate i vari metodi e la rispettiva descrizione.

Modulo **nutriscore_classification.py**

knn_model(df, col_list, hypers, values):
```
"""
predizione tramite classificatore KNN
:param df: dataframe in input
:param col_list: nomi delle features da considerare
:param hypers: iperparametri ottimizzati
:param values: valori da predire
:return: nutriscore predetto
"""
```

knn_classifier(df, col_list, folds, hyp_opt: bool = False):
```
"""
testa un classificatore KNN
:param df: dataframe in input
:param col_list: lista di nomi di features da considerare
:param folds: numero di fold
:param hyp_opt: True se ottimizzazione degli iperparametri richiesta, False altrimenti
:return: valore medio di accuracy su tutte le fold
"""
```

rf_classifier(df, col_list, folds, hyp_opt: bool = False):
```
"""
testa un classificatore RF
:param df: dataframe in input
:param col_list: lista di nomi di features da considerare
:param folds: numero di fold
:param hyp_opt: True se ottimizzazione degli iperparametri richiesta, False altrimenti
:return: valore medio di accuracy su tutte le fold
```

Modulo **bayesian_network.py**

bayesian_preprocessing(food_df, values=None):

```
"""
operazioni preliminari da effettuare sul dataframe per renderlo idoneo ad una rete bayesiana
:param food_df: dataframe in input
:param values: None se non c'é bisogno di predizione
:return: dataset idoneo ad una rete bayesiana
"""
```

bayesianNetwork(food_df, values):

```
"""
previsione tramite rete bayesiana
:param food_df: dataframe in input
:param values: valori da predire
:return: stringa decisionale
"""
```

bayesianTest(food_df, folds):

```
"""
test di una rete bayesiana tramite cross-validation
:param food_df: dataframe in input
:param folds: numero di folds
:return: accuracy media
"""
```

values_to_range(new_food_df, f_old, f_val, i, cont, step):

```
"""
converte in range da 0 a 4 i valori di un dataframe
:param new_food_df: dataframe
:param f_old: nome precedente delle features
:param f_val: nome aggiornato delle features
:param i: percentuale
:param cont: contatore
:param step: passo
:return: dataframe trasformato
"""
```

Modulo **clustering.py**

kMeansCategorization(data, col_list):

```
"""
clustering del dataframe secondo features in input
:param data: dataframe su cui effettuare il clustering
:param col_list: nome delle features da considerare
:return: valori contenuti in un cluster
"""
```

kMeansCluster(df, col_list, values):

```
"""
predizione cluster dei valori in input
:param df: dataframe in input
:param col_list: nomi delle features per il clustering
:param values: valori in input
:return: stringa contenente valori appartenenti al cluster predetto
"""
```

Modulo **data_manager.py**

readCSV(path, separ):

```
"""
legge un file .csv e lo incapsula in un dataframe pandas
:param path: percorso file .csv da leggere
:param separ: separatore (carattere)
:return: dataframe contenente i dati estratti
"""
```

column_analisys(df, col):

```
"""
stampa a video il numero di valari nella feature
:param df: dataframe
:param col: nome feature
:return:
"""
```

correlation_matrix(df):

```
"""
stampa la matrice di correlazione di un dataframe
:param df:
:return:
"""
```

Modulo **hyperparameter_optimization.py**

hyper_opt_knn(X, y, folds, classifier: bool = True):

```
"""
ottimizzazione dei parametri di un modello KNN
:param X: X dataframe - valori noti
:param y: y column(s) - valori da predire
:param folds: numero di folds per la cross-validation
:param classifier: True se classificatore KNN, False se regressore KNN
:return: parametri ottimizzati
"""
```

hyper_opt_rf(X, y, folds):

```
"""
ottimizzazione dei parametri di un modello RF
:param X: X dataframe - valori noti
:param y: y column(s) - valori da predire
:param folds: numero di folds per la cross-validation
:return: parametri ottimizzati
"""
```

Modulo **cross_validation.py**

KFold_splitting(X, y, splits=10):

```
"""
divisione del dataset in train e test set
:param X: X dataframe - valori noti
:param y: y column(s) - valori da predire
:param splits: numero di folds da utilizzare
:return: lista delle varie combinazioni di folds (train/test sets)
"""
```

kFold_cross_validation_bayesian(X, y, splits=10):

```
"""
cross-validation per la rete bayesiana
:param X: X dataframe - valori noti
:param y: y column(s) - valori da predire
```

```
    :param splits: numero di folds da utilizzare
    :return: valore medio di accuracy
    """

kFold_cross_validation_knn(X, y, hypers, classifier: bool = True, splits=10):
    """
    esegue cross validation utilizzando la tecnica K-fold su un knn Classifier o Regressor
    :param hypers: valori ottimali degli iperparametri
    :param X: X dataframe - valori noti
    :param y: y column(s) - valori da predire
    :param classifier: True se knn Classifier (default), False se knn Regressor
    :param splits: numero di folds
    :return: valore medio di accuracy
    """

kFold_cross_validation_rf(X, y, hypers, splits=10):
    """
    esegue cross validation utilizzando la tecnica K-fold su un Random Forest
    :param hypers: valori ottimali degli iperparametri
    :param X: X dataframe - valori noti
    :param y: y column(s) - valori da predire
    :param splits: numero di folds
    :return: valore medio di accuracy
    """
```