ДОДАТКИ

Код 1: Побудова базових моделей та вибір найкращої з них

```
import pandas
import click
import numpy

from pandas.tools.plotting import scatter_matrix
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import
   ↪ LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

from rule_fit import RuleFitBuilder


class ModelsEvaluator(object):
    VALIDATION_SIZE = 0.2
    SEED = 7
    SCORING = 'accuracy'

    def __init__(self):
        self.dataset = None
        self.dataset_url = 'https://archive.ics.uci.edu/ml/machine-
           ↪ learning-databases/iris/iris.data'
        self.X_train = None
        self.Y_train = None
        self.X_validation = None
        self.Y_validation = None

    def load_dataset(self):
        names = ['sepal-length', 'sepal-width', 'petal-length', '
           ↪ petal-width', 'class']
        _info('Loading␣dataset␣into␣memory...')
        self.dataset = pandas.read_csv(self.dataset_url, names=names
           ↪ )
```

```
def split_train_and_validation(self):
    # Split-out validation dataset
    array = self.dataset.values
    X = array[:,0:4]
    Y = array[:,4]
    X_train, X_validation, Y_train, Y_validation = \
        model_selection.train_test_split(X, Y, test_size=self.
            ↪ VALIDATION_SIZE, random_state=self.SEED)
    self.X_train = X_train
    self.Y_train = Y_train
    self.X_validation = X_validation
    self.Y_validation = Y_validation
    print('Train and validation partitions were created')

@staticmethod
def build_models():
    _info('Building models...')
    models = {
        'LR': LogisticRegression(),
        'LDA': LinearDiscriminantAnalysis(),
        'KNN': KNeighborsClassifier(),
        'NB': GaussianNB(),
        'SVM': SVC(),
        'CART': DecisionTreeClassifier(),
    }
    print('{} models built'.format(len(models)))
    return models

@staticmethod
def choose_best_model(results):
    highest_accuracy = 0
    best_model_name = None
    for model_name, result in results.items():
        if result.mean() > highest_accuracy:
            highest_accuracy = result.mean()
            best_model_name = model_name
    _info('Best model is {}'.format(best_model_name))
    return best_model_name

def train_models(self, models):
    # evaluate each model in turn
    results = {}
```

```
    _info('Training␣models...')
    for name, model in models.items():
        kfold = model_selection.KFold(n_splits=10, random_state=
            ↪ self.SEED)
        cv_results = model_selection.cross_val_score(model,
            self.X_train, self.Y_train, cv=kfold, scoring=self.
                ↪ SCORING)
        results[name] = cv_results
    return results

def get_prediction_accuracy(self, model):
    model.fit(self.X_train, self.Y_train)
    predictions = model.predict(self.X_validation)
    accuracy = accuracy_score(self.Y_validation, predictions)
    print(confusion_matrix(self.Y_validation, predictions))
    print(classification_report(self.Y_validation, predictions))
    return accuracy

def compare_to_rule_fit(self, model):
    approximation_builder = RuleFitBuilder(model)
    _note('Building␣RuleFit␣approximation...')
    rule_fit_model = approximation_builder.
        ↪ build_rule_fit_approximation()

    accuracy_model = self.get_prediction_accuracy(model)
    _info('Predictions␣for␣the␣best␣model:␣{}'.format(
        ↪ accuracy_model))
    accuracy_rule_fit = self.get_prediction_accuracy(
        ↪ rule_fit_model)
    _info('Prediction␣for␣the␣rule␣fit␣model:␣{}'.format(
        ↪ accuracy_rule_fit))

    _note('Deviation:␣{}%'.format(numpy.std([accuracy_model,
        ↪ accuracy_rule_fit])))


def show_results(self, results):
    from terminaltables import AsciiTable
    table_data = [
        ['Model␣name', 'Validation', 'Standard␣deviation']
    ]
    for model_name, cv_results in results.items():
        table_data.append([model_name, cv_results.mean(),
```

```
                ↪ cv_results.std()])
        table = AsciiTable(table_data)

        print(table.table)


def _info(text):
    click.secho(text, fg='yellow')


def _note(text):
    click.secho(text, fg='green')


def main():
    evaluator = ModelsEvaluator()
    evaluator.load_dataset()
    evaluator.split_train_and_validation()
    models = evaluator.build_models()
    train_results = evaluator.train_models(models)
    evaluator.show_results(train_results)
    best_model_name = evaluator.choose_best_model(train_results)
    best_model = models[best_model_name]
    evaluator.compare_to_rule_fit(best_model)
```