**Отчет для ИДЗ**

**по предмету «Кросплатформене програмування на мові Java»**

**Выполнил:**

**Носов Максим**

**Одесса 2020**

**Задание:**

Требуемый функционал:

1. Поддержка возможности загрузки входных данных из файла формата csv.
2. Отображение импортированных данных в таблице с возможностями редактирования и сортировки
3. Расчет и вывод в другой таблице статистических показателей данных (по примеру, как это выводится в дедукторе или рапидмайнере), среднее, медиана, мода, максимум, минимум…
4. Возможность выбора нужного алгоритма МО и его конфигурации (выбор параметров алгоритма, у каждого есть свои входные, так называемые, гиперпараметры)
5. Возможность выбора метрики оценки качества модели (например, MSE, подробнее тут <https://habr.com/ru/company/ods/blog/328372/>)
6. Возможность разбития импортированной выборки на входную и тестовую.
7. Возможность запуска алгоритма и построения графика его ошибки + выдачи данных в текстовом виде (в файл или текстовое поле)

8. Возможность сохранения полученных данных в виде отчета в файл, хотя бы текстовый.

Для данной работы были выбраны 3 алгоритма обучения:

1. Линейная регрессия

2. Логистическая регрессия

3. К-ближайших соседей, в варианте алгоритма классификации

Пример работы программы представлен в прикрепленном видео.

Вывод:

Была проделана работа по разработке программы для сравнения трех алгоритмов обучения модели. Программа имеет возможность выбора алгоритма обучения, загрузки csv либо xlsx файлов, выбора метрики, выбора сравниваемых параметров, установки гиперпараметров для модели, возможность вывода статистики обучения и записи ее в файл.

Листинг:

**\_\_init\_\_.py**

import sys

import random

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

from datetime import date

from fpdf import FPDF, HTMLMixin

from PySide2.QtWidgets import QApplication, QMainWindow, QFileDialog, QListWidgetItem, QTableWidgetItem

from UIs.mainwindow import Ui\_MainWindow

from StatDialog import StatDialog

from myRow import MyRow

class MyFPDF(FPDF, HTMLMixin):

pass

class Base(QMainWindow):

def \_\_init\_\_(self):

QMainWindow.\_\_init\_\_(self)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

self.\_hiperNames = []

self.\_X = []

self.\_y = []

self.\_dataSet = []

self.\_train\_X = []

self.\_train\_y = []

self.\_test\_X = []

self.\_test\_y = []

self.\_predictions = []

self.\_method = []

self.\_metric = []

self.\_statDialog = StatDialog()

self.\_scoreResult = 0

self.\_metricResult = 0

self.\_isSplit = True

self.\_isStatSaved = False

self.\_isFileParsed = False

self.\_classMetricLabels = ('accuracy', 'balanced\_accuracy', 'average\_precision', 'neg\_brier\_score', 'f1', 'precision', 'precision', 'recall', 'jaccard')

self.\_regressionMetricLabels = ('explained\_variance', 'max\_error', 'neg\_mean\_absolute\_error', 'neg\_mean\_squared\_error', 'neg\_median\_absolute\_error', 'r2', 'neg\_mean\_poisson\_deviance', 'neg\_mean\_gamma\_deviance')

self.ui.cb\_method.addItems(('Linear regression', 'Logistic regression', 'K-nearest neighbors'))

self.ui.cb\_method.currentIndexChanged.connect(self.\_setMethod)

self.ui.cb\_metric.addItems(self.\_regressionMetricLabels)

self.ui.cb\_metric.currentIndexChanged.connect(self.\_setMetric)

self.ui.cb\_parametr1.currentIndexChanged.connect(self.\_setParametr1)

self.ui.cb\_parametr2.currentIndexChanged.connect(self.\_setParametr2)

self.ui.lw\_parametrs.itemChanged.connect(self.\_updateHiperparamets)

self.ui.chb\_twice.stateChanged.connect(self.\_updateTwiceData)

self.ui.spb\_k\_count.editingFinished.connect(self.\_setK)

self.ui.btn\_setFile.clicked.connect(self.\_parseFile)

self.ui.btn\_start.clicked.connect(self.\_startTraining)

self.ui.btn\_stat.clicked.connect(self.\_showStatistic)

def \_startTraining(self):

self.\_isStatSaved = False

print('start train')

self.\_train\_X, self.\_test\_X, self.\_train\_y, self.\_test\_y = train\_test\_split(self.\_X, self.\_y, test\_size=0.25, random\_state=42)

self.\_method.fit(self.\_train\_X, self.\_train\_y)

self.\_predictions = self.\_method.predict(self.\_test\_X)

print(self.\_predictions)

print('stop train')

self.\_scoreResult = self.\_method.score(self.\_test\_X, self.\_test\_y)

print(self.\_scoreResult)

self.\_metricResult = self.\_metric(self.\_test\_y, predictions)

print(self.\_metricResult)

self.\_statDialog.setData(self.\_train\_X, self.\_train\_y, self.\_test\_X, self.\_test\_y, predictions)

self.\_statDialog.show()

if not self.\_isStatSaved:

self.\_statDialog.save()

self.\_isStatSaved = True

def \_updateTwiceData(self):

if self.ui.chb\_twice.isChecked():

self.\_isSplit = True

else:

self.\_isSplit = False

def \_setMethod(self):

if self.ui.cb\_method.currentIndex() == 0:

self.\_method = LinearRegression()

self.ui.cb\_metric.clear()

self.ui.cb\_metric.addItems(self.\_regressionMetricLabels)

self.\_metric = []

elif self.ui.cb\_method.currentIndex() == 1:

self.\_method = LogisticRegression()

self.ui.cb\_metric.clear()

self.ui.cb\_metric.addItems(self.\_classMetricLabels)

self.\_metric = []

elif self.ui.cb\_method.currentIndex() == 2:

self.\_method = KNeighborsClassifier(n\_neighbors=5)

self.ui.cb\_metric.clear()

self.ui.cb\_metric.addItems(self.\_classMetricLabels)

self.\_metric = []

def \_showStatistic(self):

self.\_statDialog.show()

if not self.\_isStatSaved:

self.\_statDialog.save()

self.\_isStatSaved = True

# -------------------------------

def \_setParametr1(self):

if self.\_isFileParsed:

if not isinstance(self.\_y, list):

self.\_X = self.\_dataSet[self.ui.cb\_parametr1.currentText()]

self.\_X = self.\_X.values

self.\_X = self.\_X.reshape(-1, 1)

print(self.\_X)

else:

self.\_X = self.\_dataSet[self.ui.cb\_parametr1.currentText()]

self.\_X = self.\_X.values

self.\_X = self.\_X.reshape(-1, 1)

print(self.\_X)

def \_setParametr2(self):

if self.\_isFileParsed:

if not isinstance(self.\_X, list):

self.\_y = self.\_dataSet[self.ui.cb\_parametr2.currentText()]

self.\_y = self.\_y.values

print(self.\_y)

else:

self.\_y = self.\_dataSet[self.ui.cb\_parametr2.currentText()]

self.\_y = self.\_y.values

print(self.\_y)

def \_updateHiperparamets(self):

self.\_hiperNames = []

for i in range(self.ui.lw\_parametrs.count()):

item = self.ui.lw\_parametrs.item(i)

element = self.ui.lw\_parametrs.itemWidget(item)

if isinstance(element, MyRow) and element.isChecked():

self.\_hiperNames.append(element.getText())

print(self.\_hiperNames)

def \_parseFile(self):

fileInfo = QFileDialog.getOpenFileName()

fileName = fileInfo[0].split('.')

result = 0

if fileName[-1] == 'csv':

result = pd.read\_csv(fileInfo[0], error\_bad\_lines=False)

else:

result = pd.read\_excel(fileInfo[0], error\_bad\_lines=False)

self.\_dataSet = result

self.\_dataSet = self.\_dataSet.fillna(0)

self.\_fillTable()

self.\_setHiperparametrs(self.\_dataSet.columns)

self.ui.cb\_parametr1.addItems(list(self.\_dataSet.columns))

self.ui.cb\_parametr2.addItems(list(self.\_dataSet.columns))

self.\_isFileParsed = True

def \_setMetric(self):

if self.ui.cb\_method.currentIndex() == 0:

if self.ui.cb\_metric.currentIndex() == 0:

self.\_metric = metrics.explained\_variance\_score

elif self.ui.cb\_metric.currentIndex() == 1:

self.\_metric = metrics.max\_error

elif self.ui.cb\_metric.currentIndex() == 2:

self.\_metric = metrics.mean\_absolute\_error

elif self.ui.cb\_metric.currentIndex() == 3:

self.\_metric = metrics.mean\_squared\_error

elif self.ui.cb\_metric.currentIndex() == 4:

self.\_metric = metrics.median\_absolute\_error

elif self.ui.cb\_metric.currentIndex() == 5:

self.\_metric = metrics.r2\_score

elif self.ui.cb\_metric.currentIndex() == 6:

self.\_metric = metrics.mean\_poisson\_deviance

elif self.ui.cb\_metric.currentIndex() == 7:

self.\_metric = metrics.mean\_gamma\_deviance

else:

if self.ui.cb\_metric.currentIndex() == 0:

self.\_metric = metrics.accuracy\_score

elif self.ui.cb\_metric.currentIndex() == 1:

self.\_metric = metrics.balanced\_accuracy\_score

elif self.ui.cb\_metric.currentIndex() == 2:

self.\_metric = metrics.average\_precision\_score

elif self.ui.cb\_metric.currentIndex() == 3:

self.\_metric = metrics.brier\_score\_loss

elif self.ui.cb\_metric.currentIndex() == 4:

self.\_metric = metrics.f1\_score

elif self.ui.cb\_metric.currentIndex() == 5:

self.\_metric = metrics.precision\_score

elif self.ui.cb\_metric.currentIndex() == 6:

self.\_metric = metrics.recall\_score

elif self.ui.cb\_metric.currentIndex() == 7:

self.\_metric = metrics.jaccard\_score

def \_setK(self):

self.\_method = self.\_method = KNeighborsClassifier(n\_neighbors=int(self.ui.spb\_k\_count.value()))

def \_setHiperparametrs(self, names):

for name in names:

item = QListWidgetItem(self.ui.lw\_parametrs)

self.ui.lw\_parametrs.addItem(item)

row = MyRow(name)

item.setSizeHint(row.minimumSizeHint())

self.ui.lw\_parametrs.setItemWidget(item, row)

def \_fillTable(self):

for i in range(self.ui.tbw\_data.rowCount(), -1, -1):

self.ui.tbw\_data.removeRow(i)

i = 0

j = 0

self.ui.tbw\_data.setColumnCount(len(list(self.\_dataSet.columns)))

self.ui.tbw\_data.setHorizontalHeaderLabels(list(self.\_dataSet.columns))

tmp = self.\_dataSet.values

for record in self.\_dataSet.values:

self.ui.tbw\_data.insertRow(i)

for item in record:

if item == np.nan:

continue

else:

self.ui.tbw\_data.setItem(i, j, QTableWidgetItem(str(item)))

j += 1

j = 0

i += 1

def \_makeFile(self):

pdf = MyFPDF()

pdf.add\_page()

html = '<h1>Data info</h1><table border="1" width="90%"><thead><tr><th width="20%">train\_X</th><th width="20%">train\_y</th><th width="20%">test\_X</th><th width="20%">test\_y</th><th width="20%">predictions</th></tr></thead><tbody>'

sum = 0

for i in range(len(self.\_train\_X)):

if i >= len(self.\_test\_X):

html += '<tr><td>{}</td><td>{}</td><td>{}</td><td>{}</td><td>{}</td></tr>'.format(self.\_train\_X[i], self.\_train\_y[i], 0, 0, 0)

else:

html += '<tr><td>{}</td><td>{}</td><td>{}</td><td>{}</td><td>{}</td></tr>'.format(self.\_train\_X[i], self.\_train\_y[i], self.\_test\_X[i], self.\_test\_y[i], self.\_predictions[i])

html += '<tr><td colspan=4 >score answer </td><td>{}</td></tr>'.format(self.\_scoreResult)

html += '<tr><td colspan=4 >Metric answer </td><td>{}</td></tr>'.format(self.\_metricResult)

html += '</tbody></table>'

pdf.write\_html(html)

d = date.today()

t = d.timetuple()

pdf.output('data\_{}-{}-{}.pdf'.format(t.tm\_hour, t.tm\_min, t.tm\_sec))

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

window = Base()

window.show()

sys.exit(app.exec\_())

**StatDialog.py**

from UIs.statDialog import Ui\_Dialog

from PySide2.QtWidgets import QDialog, QTableWidgetItem, QTableWidget

import matplotlib.pyplot as plt

import numpy as np

from scipy import stats

from datetime import date

from fpdf import FPDF, HTMLMixin

class MyFPDF(FPDF, HTMLMixin):

pass

class StatDialog(QDialog):

def \_\_init\_\_(self, train\_X=[], train\_y=[], test\_X=[], test\_y=[], predictions=[]):

QDialog.\_\_init\_\_(self)

self.ui = Ui\_Dialog()

self.ui.setupUi(self)

self.\_train\_X = train\_X

self.\_train\_y = train\_y

self.\_test\_X = test\_X

self.\_test\_y = test\_y

self.\_predictions = predictions

self.\_stats = []

self.\_targets = [self.\_train\_X, self.\_train\_y, self.\_test\_X, self.\_test\_y, self.\_predictions]

self.ui.tbw\_stat.setColumnCount(5)

self.ui.tbw\_data.setColumnCount(5)

self.ui.tbw\_stat.setHorizontalHeaderLabels(('train\_X', 'train\_y', 'test\_X', 'test\_y', 'predictions'))

self.ui.tbw\_data.setHorizontalHeaderLabels(('train\_X', 'train\_y', 'test\_X', 'test\_y', 'predictions'))

self.ui.tbw\_stat.setVerticalHeaderLabels(('Mean', 'Max', 'Min', 'Median', 'Moda'))

def setData(self, train\_X=[], train\_y=[], test\_X=[], test\_y=[], predictions=[]):

self.\_train\_X = train\_X

self.\_train\_y = train\_y

self.\_test\_X = test\_X

self.\_test\_y = test\_y

self.\_predictions = predictions

self.\_targets = [self.\_train\_X, self.\_train\_y, self.\_test\_X, self.\_test\_y, self.\_predictions]

self.\_update()

def show(self):

if not isinstance(self.\_train\_X, list) and not isinstance(self.\_train\_y, list) and not isinstance(self.\_test\_X, list) and not isinstance(self.\_test\_y, list) and not isinstance(self.\_predictions, list):

self.\_showInfo()

else:

pass

def save(self):

if not isinstance(self.\_train\_X, list) and not isinstance(self.\_train\_y, list) and not isinstance(self.\_test\_X, list) and not isinstance(self.\_test\_y, list) and not isinstance(self.\_predictions, list):

pdf = MyFPDF()

pdf.add\_page()

html = '<h1>Data info</h1><table border="1" width="90%"><thead><tr><th width="20%">train\_X</th><th width="20%">train\_y</th><th width="20%">test\_X</th><th width="20%">test\_y</th><th width="20%">self.\_predictions</th></tr></thead><tbody>'

sum = 0

for i in range(len(self.\_stats[0])):

html += '<tr><td>{}</td><td>{}</td><td>{}</td><td>{}</td><td>{}</td></tr>'.format(self.\_stats[0][i], self.\_stats[1][i], self.\_stats[2][i], self.\_stats[3][i], self.\_stats[4][i])

html += '</tbody></table>'

pdf.write\_html(html)

d = date.today()

t = d.timetuple()

pdf.output('stat\_{}-{}-{}.pdf'.format(t.tm\_hour, t.tm\_min, t.tm\_sec))

else:

return False

def \_update(self):

if self.ui.tbw\_data.rowCount() > 0:

for i in range(self.ui.tbw\_data.rowCount()):

self.ui.tbw\_data.removeRow(i)

if self.ui.tbw\_stat.rowCount() > 0:

for i in range(self.ui.tbw\_stat.rowCount()):

self.ui.tbw\_stat.removeRow(i)

j = 0

for i in range(len(self.\_targets[0])):

self.ui.tbw\_data.insertRow(i)

for i in range(5):

self.ui.tbw\_stat.insertRow(i)

for element in self.\_targets:

print(element.dtype)

l = []

if not element.dtype == 'object':

l.append(np.mean(element))

l.append(np.max(element))

l.append(np.min(element))

l.append(np.median(element))

l.append(stats.mode(element))

self.ui.tbw\_stat.setItem(0, j, QTableWidgetItem(str(np.mean(element))))

self.ui.tbw\_stat.setItem(1, j, QTableWidgetItem(str(np.max(element))))

self.ui.tbw\_stat.setItem(2, j, QTableWidgetItem(str(np.min(element))))

self.ui.tbw\_stat.setItem(3, j, QTableWidgetItem(str(np.median(element))))

self.ui.tbw\_stat.setItem(4, j, QTableWidgetItem(str(stats.mode(element))))

else:

l.append('-')

l.append('-')

l.append('-')

l.append('-')

l.append('-')

self.ui.tbw\_stat.setItem(0, j, QTableWidgetItem(str('-')))

self.ui.tbw\_stat.setItem(1, j, QTableWidgetItem(str('-')))

self.ui.tbw\_stat.setItem(2, j, QTableWidgetItem(str('-')))

self.ui.tbw\_stat.setItem(3, j, QTableWidgetItem(str('-')))

self.ui.tbw\_stat.setItem(4, j, QTableWidgetItem(str('-')))

j += 1

self.\_stats.append(l)

i = 0

j = 0

for element in self.\_targets:

for item in element:

if item == np.nan:

continue

else:

self.ui.tbw\_data.setItem(i, j, QTableWidgetItem(str(item)))

i += 1

i = 0

j += 1

def \_showInfo(self):

plt.figure(figsize=(12, 6))

plt.plot(self.\_test\_X, self.\_predictions) # regression line

plt.plot(self.\_test\_X, self.\_test\_y, 'ro') # scatter plot showing actual data

plt.title('Actual vs Predicted')

plt.xlabel('X')

plt.ylabel('y')

plt.show()

self.exec\_()

if self.ui.tbw\_data.rowCount() > 0:

for i in range(self.ui.tbw\_data.rowCount()):

self.ui.tbw\_data.removeRow(i)

if self.ui.tbw\_stat.rowCount() > 0:

for i in range(self.ui.tbw\_stat.rowCount()):

self.ui.tbw\_stat.removeRow(i)