Cuprins

1. Contribuții 2
2. Implementare 2
2.1. Extragerea unor caracteristici utile în predicția notelor 2
2.2. Prelucrarea seturilor de date 5
2.3. Identificarea tehnicilor de învățare automată 7
2.4. Antrenarea unui model de învățare automată 8
2.5. Testarea și analizarea rezultatelor 8
3. Rezultate 8
4. Bibliografie 12

1. Contribuții

Pasii pentru rezolvarea acestei probleme au fost stabiliti la comun. Am incercat sa lucram modular impartind proiectul in 3 bucati principale: implementarea caracteristilor pentru analiza proiectelor de POO, prelucrarea seturilor de date obtinute din evaluarea caracteristicilor mentionate anterior, realizarea si antrenarea modelului.

Pentru a evita aparitia erorilor, am verificat fiecare bucata de cod/ implementare noua la fiecare pas, iar in cazul intampinarii anumitor probleme am incercat sa le rezolvam impreuna.

2. Implementare

Pentru partea de implementare sunt explicate functiile din fisierul „interpreter.py".

2.1. Extragerea unor caracteristici utile în predicția notelor

Pentru a analiza proiectele, ne-am gandit la cateva caracteristici generale ce tin de orice implementare orientata pe obicte (existenta unor interfete, modul de implementare al acestora prin metode virtuale, numarul de clase pentru modularizare) si caracteristici care tin de redactarea unui proiect (existenta si consistenta unui fisier README cu intructiuni pentru anumite particularitati ale proiectului, prezenta unei diagrame ca poza sau fisier .cd pentru proiectarea solutiei si de asemnea, am luat in considerare si numarul de linii din tot proiectul).

Explicatii pentru implementare:

- **Parametri initiali**
  - PATH = path-ul in care se afla lucrarile studentilor (ori cele de train ori cele de test)
  - GRADES_PATH = in ce folder se afla labels.txt
  - PATH_*_CSV = numele fisierului .csv de output, unde * poate fi de test sau de train
  - CSV_HEADER = capete de tabel pentru fisierul .csv: Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines
- **check_readme()**

  Functia este apelata cu calea returnata de find_readme_path() in care cautam in folderul studentului curent daca exista un fisier .txt care contine in denumirea sa „read" sau „raed" indiferent de case sau alte caractere folosite. Am recurs la aceasta metoda, deoarece analizand folderele cu datele de input am intampinat mai multe probleme: nu toate directoarele corespunzatoare studentilor au aceeasi structura, diferenta de locatie pentru anumite documente (diagrame atat in format .jpg cat si .cd, readme) si greseli gramaticale in ceea ce priveste readme (ex: studentul de test nr. 53 -> RaedMe!!.txt).

```
def find_readme_path(student):
    for subdir, dirs, files in os.walk(PATH + "\\" + student):
        for file in files:
            # we discovered that some readmes are named RAED
            if "read" in file.lower() and ".txt" in file.lower() or "raed" in file.lower():
                return subdir
```

Aceasta functie returneaza 0 daca nu exista si 1 daca exista un fisier in formatul mentionat mai sus. De asemenea, va returna numarul de caractere in locul existentei fisierului, daca secventa de cod care incepe cu „with open....” este necomentata.

```
def check_readme(path):  # returneaza 1 daca are fisier de readme
    readme = 0
    for std in os.listdir(path):
        if "read" in std.lower() or "raed" in std.lower():
            readme = 1
            # Pentru normalizarea numarului de caractere din readme. Daca comentam partea asta, vom avea numai
            #       0 sau 1, daca are fisier de readme sau nu
            with open(path + "\\" + std, "r") as file:
                filename = file.read()
                readme = sum(len(word) for word in filename)
            #### Comenteaza pana aici

    return readme
```

- **trace_files()**

Este functia care cauta in toate directoarele studentului curent pentru a gasi fisierele dorite in functie de extensie. Am recurs la aceasta metoda, deoarece structura de directoare nu este la fel la toti studentii.

Aici sunt apelate functiile pentru gasirea numarului de interfete, functii virtuale, clase, linii si implementata o verificare pentru existenta diagramelor. Verifica folderul in care avem fisiere .cpp si fisiere .h, iar pentru fiecare fisier in parte sunt apelate functiile corespunzatoare.

Functia este apelata cu doi parametrii: path (calea catre folderul cu fisiere .cpp si .h) si diagram_path (calea returnata de find_diagram_path() care cauta in toate subdirectoarele studentului respectiv prezenta unor fisiere cu extensia .cd sau .jpg, .png etc.)

```
def trace_files(path, diagram_path):
    interfaces = 0
    virtual_functions = 0
    classes = 0
    diagrams = 0
    lines = 0
    _path = path + "\\"
    for std in os.listdir(path):  # parcurge folderul pentru student cu .cpp si .h
        try:
            interfaces, virtual_functions = check_interface(_path + std, interfaces, virtual_functions)
            classes = check_classes(_path + std, classes)

            lines += get_lines(_path + std)
        except:
            pass

    if diagram_path is not None:
        diagrams = 1

    return {"interfaces": interfaces, "vfuncs": virtual_functions, "classes": classes, "diagrams": diagrams,
            "lines": lines}
```

- **check_interface()**

Functia returneaza numarul de interfete din directorul care contine clasele. Verficare se va face dupa existenta literei „i” (indiferent de case-ul lui i) urmata de o majuscula. In aceasta functie se apeleaza si functia check_virtual_functions() care numara metodele virtuale.

```
def check_interface(std, interfaces, virtual_functions):
    if ".h" in std:  # verificam numai dupa headere
        # Daca prima litera este i si urmatoarea este majuscula (ex. IDrawable sau iDraw)
        if std.split("\\")[-1][0].lower() == "i" and std.split("\\")[-1][1].isupper():
            virtual_functions += check_virtual_functions(std)
            interfaces += 1
    return interfaces, virtual_functions
```

- **check_virtual_functions()**

Verificam in headers dupa cuvantul „virtual” pentru a identifica metodele virtuale. Nu luam in considerare si destructorul virtual al clasei.

```
def check_virtual_functions(path):  # returneaza numarul functiilor virtuale
    virtual_functions = 0
    f = open(path, "r")
    for line in f.readlines():
        if "virtual" in line and "~" not in line:
            virtual_functions += 1
    return virtual_functions
```

- **check_classes()**

Functie care returneaza numarul de clase in functie de cate fisiere .cpp avem. Am presupus ca exista o singura clasa intr-un fisier .cpp

```
def check_classes(std, classes):
    if ".cpp" in std:
        classes += 1
    return classes
```

- **get_lines()**

  Functie care returneaza numarul de linii din fisierele .cpp sau .h. Acesta au fost adunate in trace_files() pentru a analiza dimensiunea proiectului.

```
def get_lines(std):
    if ".h" in std or ".cpp" in std:
        f = open(std, "r")
        return len(f.readlines())
    return 0
```

2.2. Prelucrarea seturilor de date

-despre rezultatele concentrate in csv si normalizare

```
class Student:
    def __init__(self, name, grade, readme, interfaces, vfuncs, classes, diagrams, lines):
        self.name = name
        if grade is None:
            self.grade = -1
        else:
            self.grade = grade
        self.readme = readme
        self.interfaces = interfaces
        self.vfuncs = vfuncs
        self.classes = classes
        self.diagrams = diagrams
        self.lines = lines
```

Clasa Student este implementata ca sa usureze manipularea datelor si scrierea in fisierul .csv. In functia get_students_as_obj() se creeaza o lista de studenti care au ca membri ai clasei parametrii: Student, Grade, Readme, Interfaces, Virt_functions, Classes, Diagrams, Lines.

De precizat ca in cazul in care se lucreaza pe datele de train, nu se iau in considerare intrarile NaN, dar la datele de test, pe coloana grades, notele sunt „-1”.

```
def get_students_as_obj(student_list, grades_dict):
    obj_student_list = []
    diagram_path = find_diagram_path(student_list[_INDEX])  # unii studenti aveau doar poze, altii  doar .cd
    for student in student_list:
        readme_path = find_readme_path(student)
        path = find_classes_path(student)
        trace = trace_files(path, diagram_path)
        if PATH.split("\\")[-1] == "test":  # studentii din test nu au nota, asa ca ii tratez diferit
            if not student in grades_dict:
                grade = None
            elif student in grades_dict:
                grade = grades_dict[student]
        elif PATH.split("\\")[-1] == "train":
            if student not in grades_dict:
                continue
            else:
                grade = grades_dict[student]
        obj_student_list.append(Student(student, grade, check_readme(readme_path), trace["interfaces"],
                                        trace["vfuncs"], trace["classes"], trace["diagrams"],
                                        trace["lines"]))
    return obj_student_list
```

Am facut normalizarea numarului de linii din tot proiectul si din fisierul de readme (in cazul in care bucata cu „with open...” din functia check_readme() este decomentata). Normalizarea se face in intervalul [0,1] prin intermediul Min-Max Normalization.

```
def normalize_lines(obj_std):  # Rescaling (min-max normalization)
    max_lines = -1
    min_lines = 1000000
    # Comenteaza aici pentru a nu normaliza liniile din readme(in cazul in care ma intereseaza doar existenta readme)
    max_readme_chars = -1
    min_readme_chars = 1000000
    for obj in obj_std:
        if obj.lines > max_lines:
            max_lines = obj.lines
        if obj.lines < min_lines:
            min_lines = obj.lines
        # Comenteaza aici
        if obj.readme > max_readme_chars:
            max_readme_chars = obj.readme
        if obj.readme < min_readme_chars:
            min_readme_chars = obj.readme
        # Pana aici
    for obj in obj_std:
        obj.lines = (obj.lines - min_lines) / (max_lines - min_lines)
        # Comenteaza aici
        obj.readme = (obj.readme - min_readme_chars) / (max_readme_chars - min_readme_chars)
        # Pana aici
```

Am adunat datele intr-un fisier .csv pentru a putea face predictia notelor prin intermediul turicreate.

```
def write_to_csv(obj_std):
    if PATH.split("\\")[-1] == "test":
        filename = PATH_TEST_CSV
    elif PATH.split("\\")[-1] == "train":
        filename = PATH_TRAIN_CSV
    else:
        raise Exception("Wrong filename")

    f = open(filename, "w")
    f.write(CSV_HEADER)
    f.write("\n")
    for obj in obj_std:
        f.write(str(obj.name))
        f.write(",")
        f.write(str(obj.grade))
        f.write(",")
        f.write(str(obj.readme))
        f.write(",")
        f.write(str(obj.interfaces))
        f.write(",")
        f.write(str(obj.vfuncs))
        f.write(",")
        f.write(str(obj.classes))
        f.write(",")
        f.write(str(obj.diagrams))
        f.write(",")
        f.write(str(obj.lines))
        f.write("\n")
```

- Main

Se ia lista de studenti din directorul PATH (care poate fi ori test ori train). Se citesc notele si se pun intr-un dictionar sub forma „student: nota”. In cazul in care variabila PRINT==True, se va afisa un student cu toate caracteristicile.

```
if __name__ == "__main__":

    student_list = get_student_list()
    grades_dict = read_grades()

    if PRINT:
        path = find_classes_path(student_list[_INDEX])
        diagram_path = find_diagram_path(student_list[_INDEX])  # unii studenti aveau doar poze, altii  doar .cd
        trace = trace_files(path, diagram_path)
        readme_path = find_readme_path(student_list[_INDEX])

        print(student_list)
        print("Path: ", path)
        print(student_list[_INDEX])
        print("README: ", check_readme(readme_path))
        print("Interfaces: ", trace["interfaces"])
        print("Virt funcs: ", trace["vfuncs"])
        print("Classes: ", trace["classes"])
        print("Diagrams : ", trace["diagrams"])
        print("Lines : ", trace["lines"])

    obj_std = get_students_as_obj(student_list, grades_dict)
    normalize_lines(obj_std)
    write_to_csv(obj_std)
```

2.3. Identificarea tehnicilor de învățare automată

Am folosit asa cum ne-a fost aratat la laborator, regresia liniara. Ii furnizam aumite date dupa care invata si in urma acestora obtinem anumiti coeficienti. In urma acestei „invatari” putem face predictii pe un set de date fara note doar cu aceeasi parametri pentru a afla notele.

Datele furnizate reprezinta un set de parametri (structura .csv de mai sus) cu o anumita nota.

2.4. Antrenarea unui model de învățare automată

Antrenarea se face in fisierul „regression.py” si urmeaza modelul din link-ul furnizat in bibliografie.

```
import turicreate

data = turicreate.SFrame('/content/training_*.csv')  # unde * va fi inlocuita cu numarul csv-ului respectiv
tests = turicreate.SFrame('/content/test_*.csv')
feats = ["Readme", "Interfaces", "Virt_functions", "Classes", "Diagrams", "Lines"]
model = turicreate.linear_regression.create(data, target='Grade', features=feats)
coefficients = model.coefficients
predictions = model.predict(data)
results = model.evaluate(data)
```

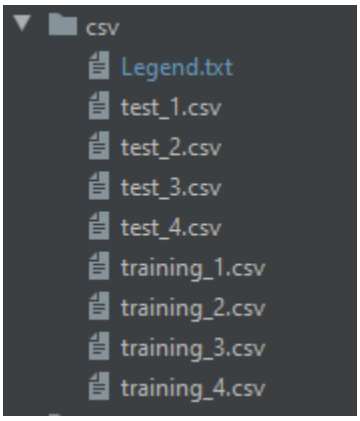
2.5. Testarea și analizarea rezultatelor

Dupa ce am obtinut modelul, facem o predictie pe baza acestuia. Vom obtine un set de note pentru studentii din setul de test. Aceste rezultate sunt analizate in *Rezultate.*

```
res_tests = model.predict(tests)
```

3. Rezultate

Am analizat datele de intrare pentru mai multe situatii. In fiecare situatie ne folosim de alte fisiere .csv. Nu am avut o metrica de verificare, asadar nu am putut alege o singura varianta. Pentru a alege o varianta, avem nevoie de notele studentilor de test.



Pentru fiecare varianta se vor folosi fisiere .csv de train si de test specifice. Acestea au urmatoarea structura:

- **Test**

```
Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines
student_51,-1,1,0,0,15,1,0.5582507687051589
student_52,-1,1,0,0,12,1,0.7116501537410318
student_53,-1,1,0,0,29,1,0.38811069354287664
student_54,-1,0,0,0,17,1,0.24461906388793986
student_55,-1,1,2,2,24,1,0.2302699009224462
student_56,-1,1,2,5,20,1,0.4185172531602323
student_74,-1,1,0,0,23,1,0.0942944994875299
```

- **Train**

```
Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines
student_1,8.55,1,0,0,30,1,0.09736831499617137
student_10,6.16,0,0,0,7,1,0.02595413694434369
student_12,5.2,1,0,0,13,1,0.059041631402893646
student_13,6.9,1,0,0,20,1,0.06605408455245235
student_14,4.41,1,0,0,14,1,0.1048240841494378
student_15,8.55,1,3,5,25,1,0.08959013420384476
student_17,5.7,0,1,2,12,1,0.052150082617982506
student_2,7.16,0,2,2,25,1,0.07729819046467577
student_22,7.36,0,0,0,22,1,0.02365695401603998
student_23,9.45,1,1,0,23,1,0.11348889694917987
student_24,10,1,10,55,41,1,0.13009309636077862
student_25,5.7,1,0,0,7,1,0.037077338491919555
student_26,6.92,1,0,0,19,1,0.058759521218716
student_27,5.01,0,0,0,12,1,0.020997057993793575
```

Am avut in vedere urmatoarele situatii pe masura ce am lucrat la caracteristicile de evaluare:

➢ **Cazul 1**: doar existenta readme
Ne intereseaza -> Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines
Unde readme.txt va fi analizat doar din punctul de vedere al existentei.

Coeficienti:

Highest Positive Coefficients

(intercept) : 6.0731
Interfaces : 0.4372
Readme : 0.3226
Lines : 0.0002

Lowest Negative Coefficients

Virt_functions : -0.0203
Diagrams : -0.0

Rezultate:

{'student_51': 6.25, 'student_52': 5.82, 'student_53': 8.42, 'student_54': 6.47, 'student_55': 7.43, 'student_56': 6.91, 'student_74': 7.57, 'student_75': 7.5, 'student_76': 6.83, 'student_77': 6.86, 'student_78': 6.04, 'student_79': 5.82}

- **Cazul 2**: existenta readme + normalizare nr_linii_proiect

Ne intereseaza -> Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines

Unde readme.txt va fi analizat doar din punctul de vedere al existentei, iar numarul de linii din intreg proiectul va fi normalizat.

Coeficienti:

Highest Positive Coefficients

(intercept) : 3.5745
Lines : 2.519
Readme : 0.183
Classes : 0.1574
Virt_functions : 0.0222

Lowest Negative Coefficients

Interfaces : -0.1096
Diagrams : -0.0

Rezultate:

{'student_51': 7.52, 'student_52': 7.44, 'student_53': 9.3, 'student_54': 6.87, 'student_55': 7.94, 'student_56': 7.85, 'student_74': 7.62, 'student_75': 8.19, 'student_76': 6.95, 'student_77': 9.11, 'student_78': 6.08, 'student_79': 5.87}

- **Cazul 3**: nr_caractere_readme + normalizare nr_linii_proiect

Ne intereseaza -> Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines

Unde readme.txt va fi analizat in functie de numarul de caractere existente, iar numarul de linii din intreg proiectul va fi normalizat.

Coeficienti:

Highest Positive Coefficients

(intercept) : 3.5984
Lines : 2.3326
Classes : 0.1576
Virt_functions : 0.0302
Readme : 0.0001

Lowest Negative Coefficients

Interfaces : -0.1638
Diagrams : -0.0

Rezultate:

{'student_51': 7.31, 'student_52': 7.22, 'student_53': 9.4, 'student_54': 6.85, 'student_55': 7.74, 'student_56': 8.01, 'student_74': 7.47, 'student_75': 8.26, 'student_76': 6.9, 'student_77': 8.91, 'student_78': 5.99, 'student_79': 5.85}

- **Cazul 4**: normalizare_nr_caractere_readme + normalizare nr_linii_proiect
  Ne intereseaza -> Student,Grade,Readme,Interfaces,Virt_functions,Classes,Diagrams,Lines
  Unde readme.txt va fi analizat in functie de numarul de caractere existente, iar numarul de linii din intreg proiectul si din readme va fi normalizat.

Coeficienti:

Highest Positive Coefficients

(intercept) : 3.5984
Lines : 2.3326
Readme : 1.1135
Classes : 0.1576
Virt_functions : 0.0302

Lowest Negative Coefficients

Interfaces : -0.1638
Diagrams : -0.0

Rezultate:

{'student_51': 7.38, 'student_52': 7.31, 'student_53': 9.85, 'student_54': 6.85, 'student_55': 7.86, 'student_56': 8.66, 'student_74': 7.51, 'student_75': 8.69, 'student_76': 6.9, 'student_77': 9.1, 'student_78': 6.13, 'student_79': 6.05}

4. Bibliografie

https://en.wikipedia.org/wiki/Feature_scaling

https://apple.github.io/turicreate/docs/api/generated/turicreate.linear_regression.LinearRegression.html