Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity

Predictia performantei academice a studentiilor din invatamantul universitar folosind Machine Learning

Tool: Python 2.7.3, Excel

Name:Luca Andrei & Salamon Adrian
Group:30241 & 30241
Email:luka_atat@yahoo.com & salamon.adrianpaul@gmail.com

Contents

1 Descrierea proiectului si setul de date 5

2 Decision Tree 8
2.1 Decision Tree on our data set 8
2.2 Experimental results 9
2.2.1 Full data set 9
2.2.2 Exclude G1 and G2 from data set 9
2.2.3 Select only G1 and G2 from data set 9
2.3 Statistics of the results 10
2.4 Python code 10

3 Support Vector Machine 11
3.1 Support Vector Machine on our data set 11
3.2 Experimental results 11
3.2.1 Full data set 11
3.2.2 Exclude G1 and G2 from data set 12
3.2.3 Select only G1 and G2 from data set 12
3.3 Statistics of the results 12
3.4 Python code 13

4 K-Nearest Neighbor 14
4.1 K-Nearest Neighbor on our data set 14
4.2 Experimental results 15
4.2.1 Full data set n_neighbors=3 15
4.2.2 Exclude G1 and G2 from data set n_neighbors=3 15
4.2.3 Select only G1 and G2 from data set n_neighbors=3 15
4.2.4 Full data set m =650 n_neighbors=k 15
4.3 Statistics of the results 16
4.4 Python code 17

5 Gradient Tree Boosting 18
5.1 Gradient Tree Boosting on our data set 18
5.2 Experimental results 18
5.2.1 Full data set 18
5.2.2 Exclude G1 and G2 from data set 18
5.2.3 Select only G1 and G2 from data set 19
5.3 Statistics of the results 19
5.4 Python code 19

6 Neural Networks **20**
6.1 Neuronal Networks on our data set . . . 20
6.2 Experimental results . . . 20
6.2.1 Full data set . . . 20
6.2.2 Exclude G1 and G2 from data set . . . 21
6.2.3 Select only G1 and G2 from data set . . . 21
6.3 Statistics of the results . . . 21
6.4 Python code . . . 22

7 Neural Networks with feature scaling **23**
7.1 Neural Networks with feature scaling on our data set . . . 23
7.2 Experimental results . . . 23
7.2.1 Full data set . . . 23
7.2.2 Exclude G1 and G2 from data set . . . 23
7.2.3 Select only G1 and G2 from data set . . . 24
7.3 Statistics of the results . . . 24
7.4 Python code . . . 24

8 AdaBoost ensemble method voting **25**
8.1 Adaboost on our data set . . . 25
8.1.1 Voting process . . . 25
8.2 Experimental results . . . 25
8.3 Statistics of the results . . . 26
8.4 Python code . . . 27

9 Conclusions **28**
9.1 Compararea performantei algoritmilor in cazul setului de date intreg . . . 28
9.2 Compararea performantei algoritmilor in cazul lipsurilor de date . . . 29
9.3 Compararea performantei algoritmilor cu feature selection . . . 30
9.4 Final thoughts . . . 30

10 Appendix **31**
10.1 Decision Tree implementation . . . 31
10.2 Support Vector Machine implementation . . . 32
10.3 K-Nearest Neighbor implementation . . . 33
10.4 Gradient Tree Boosting implementation . . . 34
10.5 Neuronal Network implementation . . . 35
10.6 Neural Networks with feature scaling implementation . . . 36
10.7 Neural Networks with feature scaling implementation . . . 37
10.8 AdaBoost voting implementation . . . 38

11 Bibliografie **40**

Task: Apply the following algorithms/techniques on your own example[1]	Available points	Your own estimation
Data preparation (e.g, outliers, feature selection, missing data)	1	
Decision/Regression Tree	1	
Support Vector Machine	1	
K-Nearest-Neighbour	1	
Multi-Layer Perceptron (ANN)	1	
Convolutional Neural Networks	1	
Recurrent Neural Networks	1	
Inductive Logic Programming	1	
Version Spaces	1	
Ensemble Learning (e.g. Ada Boost, voting)	1	
Showing performance metrics and graphs (split data into training set, test set and validation set)	1	
K-means and Bisecting K-means	1	
Hierarchical Clustering	1	
Pattern Mining/Rule Mining (e.g. Apriori)	1	
Sequential Pattern Mining/Sequential Rule Mining	1	
Applying Explainable AI (XAI) techniques on your own example	1	
Taking simple decisions (e.g. decision diagrams, expected utility)	1	
Taking Complex decisions (e.g. Bellman equation, policy iteration)	1	
Reinforcement learning	1	
Other algorithm of your choice	1	
Points	Max=20	Grade ≤ 10

Table 1: Tasks available. Select tasks of your choice such that to obtain the desired grade (10 points equals grade 10). Complete here what tasks have you tackled. To obtain the available points for **each** algorithm you have to: 1. show your python code, 2. tune some parameters, 3. explain what happens, 4. analyse the results.

Chapter 1

Descrierea proiectului si setul de date

Setul de date a fost furnizat de UCI Machine Learning Repository. Pe baza acestui set de date vom folosi diverse metode de tip machine learning pentru a prezice performantele studentiilor.

Coloana	Tip	Descriere
School	Valoare binară: 'GP' -Gabriel Pereira sau 'MS' -Mousinho da Silveira	Scoala la care este inrolat studentull
Gen	Valoare binara: F=Feminin sau G=Masculin	genul studentului
Varsta	Valoare numerica din intervalul [15,22]	Varsta studentului
Adresa	Valoare binara: U= Urban sau R=rural	Localitatea resedintei studentului
Famsize	Valoare binara: LE3= cel lumt 3 membrii sau GT3- cel putin 4 membrii	Numarul membrilor familiei
Pstatu	Valoare binara: T= casatoriti sau traiesc impreuna A= necasatoriti sau despartiti	Starea casniciei parintilor
Medu	Valoare numerică ıntre 0 si 4: 0 - fără educatie, 1 - scoală primară (4 clase), 2 - gimnaziu (8 clase), 3 - liceu (12 clase) sau 4 - ınvtământ superi	Educatia mamei
Fedu	Valoare numerică ıntre 0 si 4: 0 - fără educatie, 1 - scoală primară (4 clase), 2 - gimnaziu (8 clase), 3 - liceu (12 clase) sau 4 - ınvtământ superi	Educatia tatalui

Mjob	Valoare nominală: 'teacher' = profesoară, 'health' = domeniul medical, 'services'=administrativ, 'at home' = casnică sau 'other' - altele	Meseria mamei
Fjob	Valoare nominală: 'teacher' = profesor, 'health' = domeniul medical, 'services'=administrativ, 'at home' = casnic sau 'other' - altele	Meseria tatalui
Reason	Valoare nominală: 'home' - scoala e aproape de casă, 'reputation' - reputatia scolii, 'course' - preferinta topicului, 'other' - altele	Motivul alegerii scolii
Guardian	Valoare nominală: 'mother' - mama detine tutela, 'father' - tatăl detine tutela, 'other' - altele	Tutela studentului
TravelTime	Valoare numerică ıntre 1 si 4: 1 - mai putin de 15 minute, 2 - ıntre 15 si 30 minute, 3 - ıntre 30 minute si o oră, 4 - mai mult de o oră	timpul de dplasare pana la scoala
StudyTime	Valoare numerică ıntre 1 si 4: 1 - mai putin de 2 ore, 2 - ıntre 2 si 5 ore, 3 - ıntre 5 si 10 ore, 4 - mai mult de 10 ore	Timpul alocat invatatului pe săptămână
Failures	Valoare numerică ıntre 1 si 4: 1 - o resanta, 2 - 2 restante, 3 - 3 restante, 4 - mai mult de 3 restante	numarul de restante
Schoolsup	valoare binara(Yes/No)	Daca studentul primeste meditatii la scoala
Famsup	valoare binara(Yes/No)	Daca studentul este indrumat de membrii familiei
Paid	Valoare binara(Yes/No)	Daca studentul participa la meditatii platite
Activities	Valoare binara(Yes/No)	Daca studentul participa la activitati extrascolare
Nursery	Valoare binara(Yes/No)	Daca tudentul are cunostinte de prim ajutor

Higher	Valoare binară (Yes/No)	Daca studentul dorete sa se inroleze in invatamantul superior
Internet	Valoare binară (Yes/No)	Daca studentul are acces la internet la domiciliu
Romantic	Valoare binară (Yes/No)	Daca studentul este intr-o relatie
Famrel	Valoare numerică (o evaluare) ıntre 1 si 5: 1 - minim , 5 - maxim	Calitatea relatiei familiale
Freetime	Valoare numerică (o evalu- are) ıntre 1 si 5: 1 - minim , 5 - maxim	Timp liber
Gout	Valoare numerică (o evaluare) ıntre 1 si 5: 1 - minim , 5 - maxim	cat de des iese in oras
Dalc	Valoare numerică (o evaluare) ıntre 1 si 5: 1 - minim , 5 - maxim	Consumul de alcool afara weekendului
Walc	Valoare numerică (o evaluare) ıntre 1 si 5: 1 - minim, 5 - maxim	Consumul de alcool in weekend
Healt	Valoare numerică (o evaluare) ıntre 1 si 5: 1 - minim, 5 - maxim	Starea curenta de sanatare
Absences	Valoare numerică (o evaluare) ıntre 1 si 93	Numarul de absente
G1	Valoare numerica cuprinsa in intervalul [0,20]	Nota pentru prima evaluare
G2	Valoare numerica cuprinsa in intervalul [0,20]	Nota pentru a doua evaluare
G3	Valoare numerica cuprinsa in intervalul [0,20]	Nota pentru evaluarea finala

Pentru reducerea complexitatii am decis sa translatam valoarea lui G3 din valoare intreaga in valoare binara, astfel incat numarul de clase posibile se reduce de la 20 la 2. Notele G1 si G2 au cea mai mare pondere pentru determinarea notei finale G3, dar **G3** $\neq \frac{G1+G2}{2}$

$$G3 = \begin{cases} 1 & G3 < 12 \\ 0 & G3 \geq 12 \end{cases}$$

Coloana	Tip	Descriere
G1	Valoare numerica cuprinsa in intervalul [0,20]	Nota pentru prima evaluare
G2	Valoare numerica cuprinsa in intervalul [0,20]	Nota pentru a doua evaluare
G3	Valoare binara(1/0)	Nota pentru evaluarea a studentului finala 1= pass 0 = fail

Table 1.1: Tabelul utilizat pentru feature selection

Chapter 2

Decision Tree

2.1 Decision Tree on our data set

Decision Tree este un clasificator de tip arbore. Radacina arborelui reprezinta entitatea entitatea pe care dorim sa o clasificam, ramurile dintre radacina si frunze reprezinta decizii pe care luate de clasificator, iar frunzele sunt rezultatul obtinut in urma clasificarii.

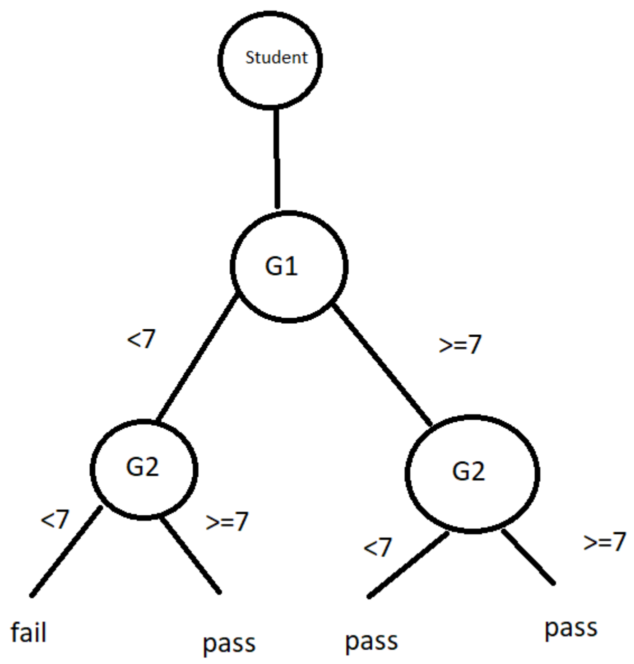


Figure 2.1: Exemplu unui model posibil generat de clasificatorul Decision Tree pentru clasificarea setului de date cu feature selection

2.2 Experimental results

2.2.1 Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	86.83	89.40	90.12	90.38	90.66
Precision	87.74	89.03	91.43	90.75	90.93

Table 2.1: Full data set

2.2.2 Exclude G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	56.00	61.14	64.15	63.68	63.88
Precision	57.76	63.40	67.40	66.27	67.27

Table 2.2: Performance after removing G1 and G2 from data set

2.2.3 Select only G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	90.66	92.00	92.62	93.31	93.69
Precision	92.53	94.72	95.34	97.18	98.43

Table 2.3: Performance after feature selection only G1 and G2

2.3 Statistics of the results

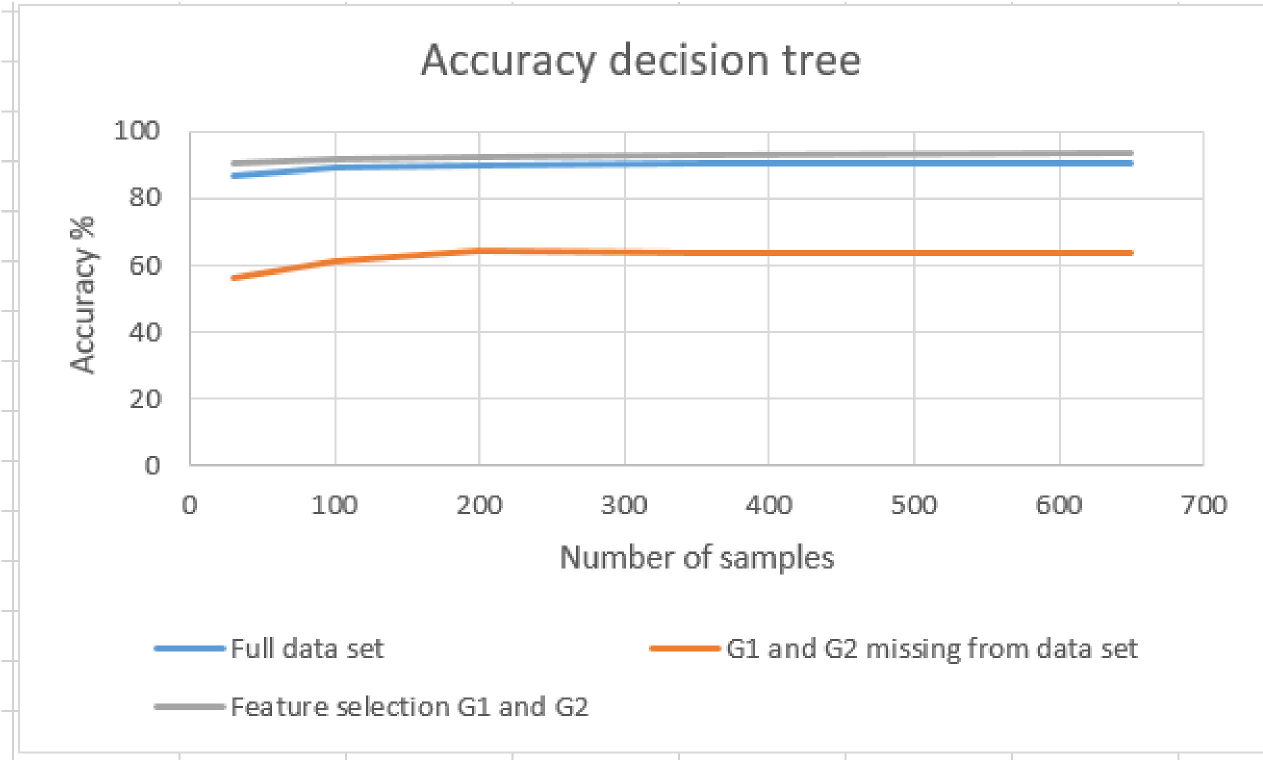


Figure 2.2: Analiza performantei clasificatorului Decision Tree, invatare supervizata, dataset= 80% antrenare si 20% testare

2.4 Python code

See 10.1

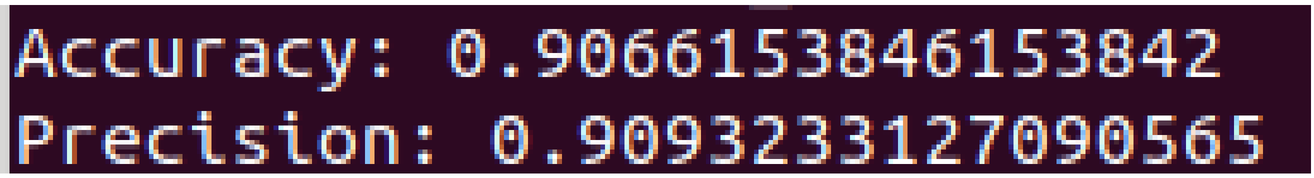


Figure 2.3: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 3

Support Vector Machine

3.1 Support Vector Machine on our data set

Datorita faptului ca avem doar 2 clase posibile pentru clasificare, Support Vector Machine este cea mai performanta solutie pe care o putem utiliza.
Acest clasificator are ca scop delimitarea datelor celor doua clase printr-o granita astfel incat:
Fie P0 punctul care apartine clasei 0, dar este cel mai aproape de punctele din clasa 1
Fie P1 punctul care apartine clasei 1, dar este cel mai aproape de punctele din clasa 0
Granita definita de SVM trece prin mijlocul distantei dintre P0 si P1 si separa complet punctele din clasa 1 de cele din clasa 0.
Clasificatorul in urma procesului de invatatre instantiaza granita care separa cele 2 clasa ca fiind o functie continua G(p).
In cadrul testarii, se reprezinta fiecare punct din setul de test si se verifica daca acest punct se afla pe partea clasei 1 sau pe partea clasei 0 conform granitei definite in urma procesului de invatare.

3.2 Experimental results

3.2.1 Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	84.16	89.50	89.95	90.98	91.36
Precision	86.54	90.49	91.72	92.90	93.33

Table 3.1: Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	60.66	66.65	68.65	70.51	72.11
Precision	56.20	67.85	70.02	71.84	72.68

Table 3.2: Performance after removing G1 and G2 from data set

3.2.2 Exclude G1 and G2 from data set

3.2.3 Select only G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	88.16	91.00	92.72	93.57	93.83
Precision	87.86	92.57	96.73	97.82	98.35

Table 3.3: Performance after feature selection only G1 and G2

3.3 Statistics of the results

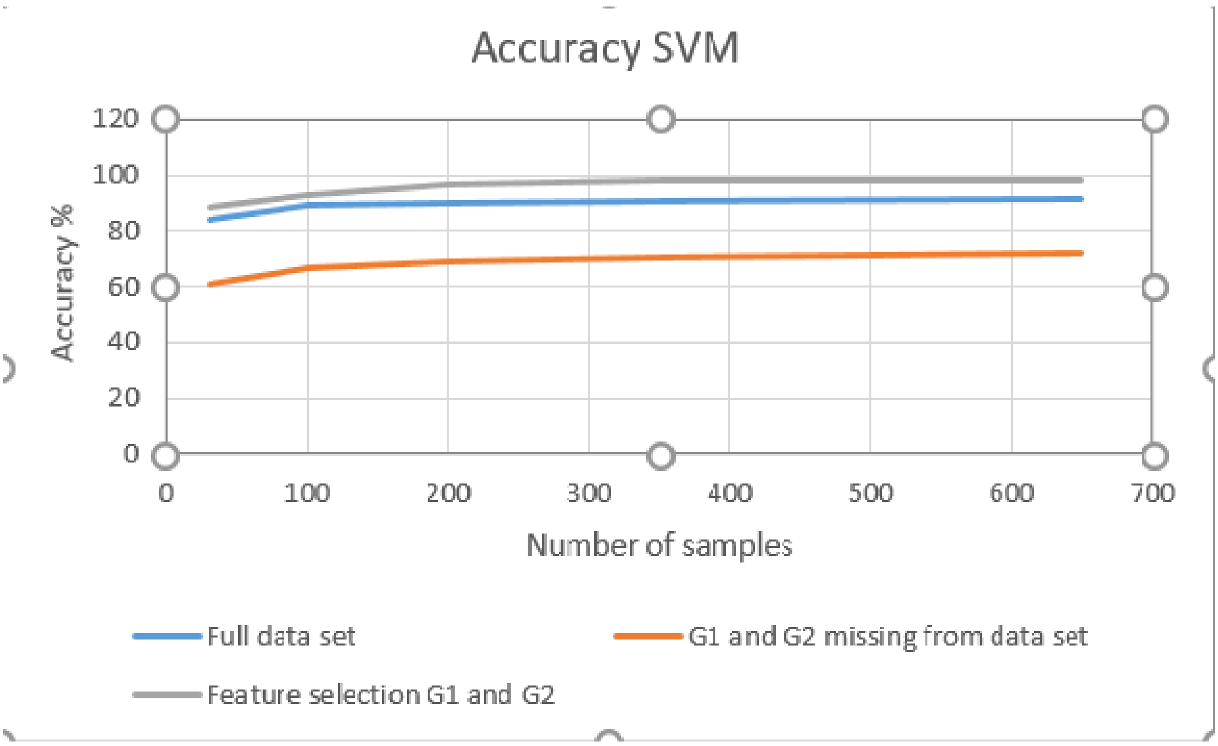


Figure 3.1: Analiza performantei clasificatorului SVM, invatare supervizata, dataset= 80% antrenare si 20% testare

3.4 Python code

See 10.2

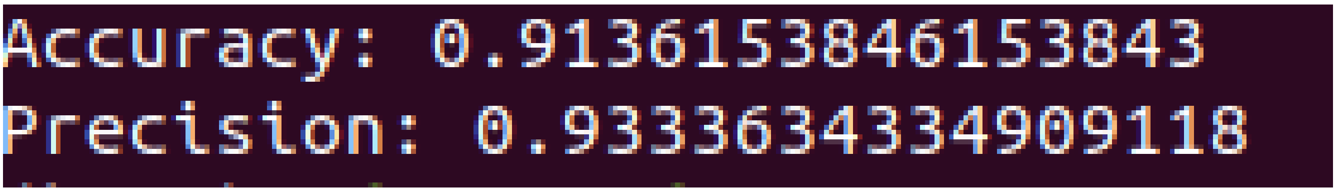


Figure 3.2: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 4

K-Nearest Neighbor

4.1 K-Nearest Neighbor on our data set

K-Nearest Neighbor este printre primii clasificatori inventati. Acest clasificator reprezinta datele intr-un spatiu multidimensional si foloseste pentru masurarea distantei in exemplul nostru metoda distantei euclidiene.
Pentru majoritatea rezultatelor am utilizat numarul de vecini considerati ca fiind 3, dar am efectuate experimente si cu k=5,7,9,11, fapt care a crescut performantele clasificatorului.
Vom exemplifica functionarea algoritmului in cazut in care efectuam feature selection in cadrul setului de date, asadar vom reprezenta datele intr-un spatiu 2D, fiecare punct avand coordonatele G1 si G2, iar clasa punctului va fi determinata de G3(clasa 0(nepromovat) pentru G3=0 si clasa 1 (promovat) pentru G3=1)

In uruma efectuarii procesului de antrenare clasificatorul va contine un numar relativ egal de puncte care corespund claselor 0 si 1.

Pentru fiecare entitate din setul de test se va reprezenta proiectia acesteia in spatiul definit in urma procesului de invatare. Se vor calcula primele k cele mai mici dinstante dintre data de test si modelul obtinut in urma procesului de invatare. Clasa care obtine numarul mai mare de vecini apropiati din cele k distante ii este asignata punctului testat.

Exemplu:
k=3
TP= punctul de test
CP= punct care apartine clasei de promovare
CNP= punct care apartine clasei de nepromovare

TP are vecinii cei mai apropiati CNP,CP, CP = TP va fi clasificat ca fiind

CP(promovat)

TP are vecinii cei mai apropiati CNP, CP, CNP = TP va fi clasificat ca fiind CNP(nepromovat)

4.2 Experimental results

4.2.1 Full data set n_neighbors=3

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	85.49	86.04	86.85	87.27	87.48
Precision	83.41	85.17	85.78	86.50	86.63

Table 4.1: Full data set

4.2.2 Exclude G1 and G2 from data set n_neighbors=3

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	58.83	63.32	63.89	64.03	63.04
Precision	59.64	62.87	62.11	63.27	63.80

Table 4.2: Performance after removing G1 and G2 from data set

4.2.3 Select only G1 and G2 from data set n_neighbors=3

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	90.49	91.65	91.77	91.69	92.02
Precision	93.41	93.88	93.50	93.22	93.76

Table 4.3: Performance after feature selection only G1 and G2

4.2.4 Full data set m =650 n_neighbors=k

	k = 3	k=5	k=7	k=9	k=11
Accuracy	88.06	88.56	89.44	89.45	90.58
Precision	87.82	88.27	89.01	88.94	90.51

Table 4.4: Full data set

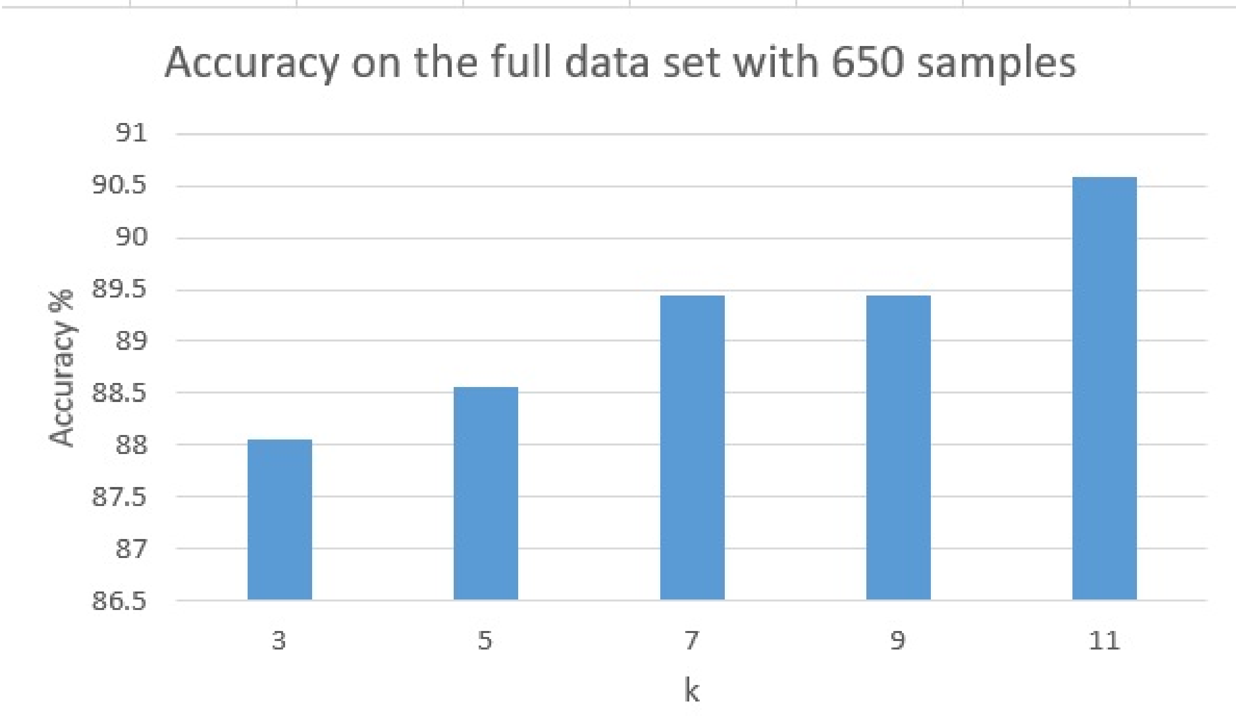


Figure 4.1: Evolutia acuratetei in functie de k, invatare supervizata, dataset= 80% antrenare si 20% testare

4.3 Statistics of the results

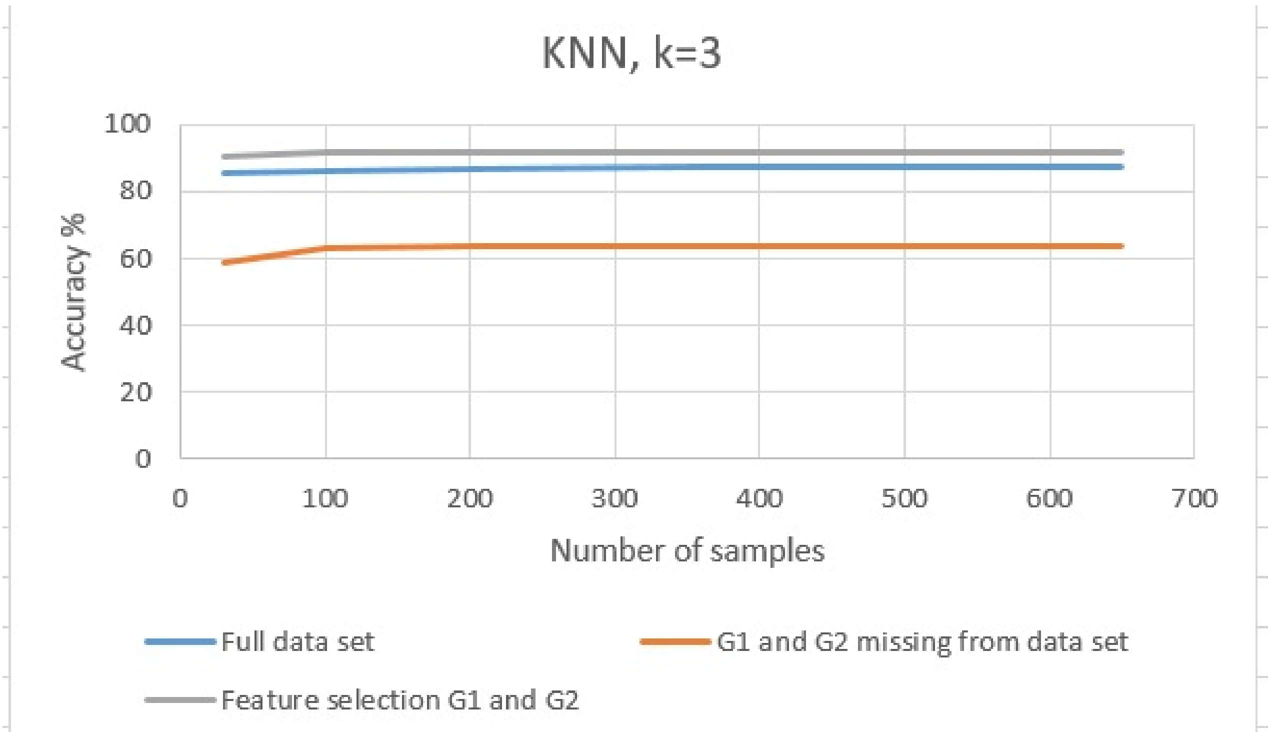


Figure 4.2: Analiza perfomantei clasificatorului KNN with k=3, invatare supervizata, dataset= 80% antrenare si 20% testare

4.4 Python code

See 10.3

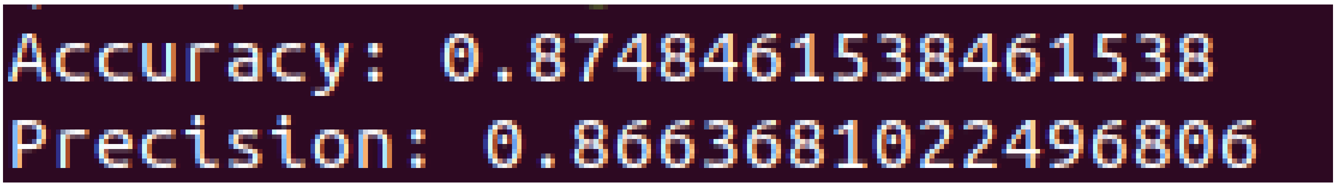


Figure 4.3: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare K=3

Chapter 5

Gradient Tree Boosting

5.1 Gradient Tree Boosting on our data set

Acest clasificator imbunatateste clasificatorul prezentat in capitolul 2(Decision Tree).
Clasificatorul Gradient Tree Boosting are ca scop imbunatatirea performantei clasificatorului Decision Tree. In cadrul acestei metode sunt creati mai multi arbori de decizie, dar spre deosebire de algoritmul random forest, se foloseste doar arborele care returneaza cele mai bune rezultate. Implementarea si exemplele sunt similare cu cele descrise in capitolul Decision Tree.
Se poate observa ca rezultatele acestui clasificator sunt afectate nesemnificativ de feature selection.

5.2 Experimental results

5.2.1 Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	88.33	91.05	91.24	91.29	92.95
Precision	89.06	91.97	93.38	94.18	94.28

Table 5.1: Full data set

5.2.2 Exclude G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	56.33	66.09	66.89	68.29	69.08
Precision	62.61	66.00	67.93	69.16	69.77

Table 5.2: Performance after removing G1 and G2 from data set

5.2.3 Select only G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	91.66	91.45	92.59	93.09	93.36
Precision	92.66	93.80	95.50	96.94	98.50

Table 5.3: Performance after feature selection only G1 and G2

5.3 Statistics of the results

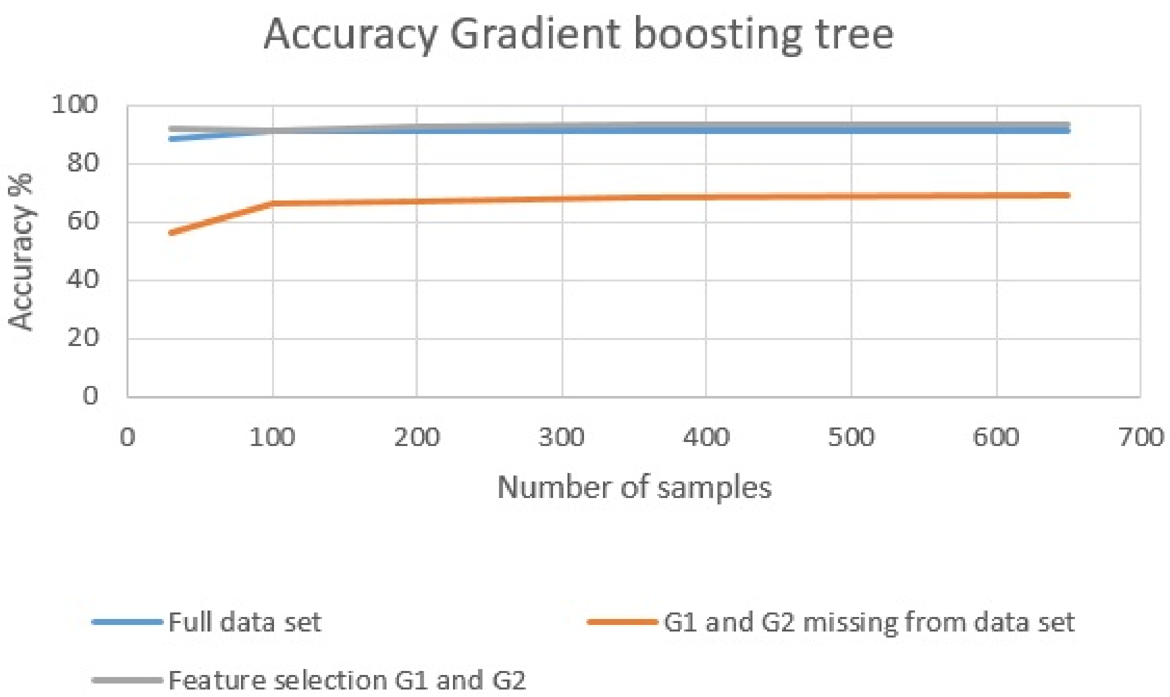


Figure 5.1: Analiza performantei clasificatorului gradient boosting tree, invatare supervizata, dataset= 80% antrenare si 20% testare

5.4 Python code

See 10.4

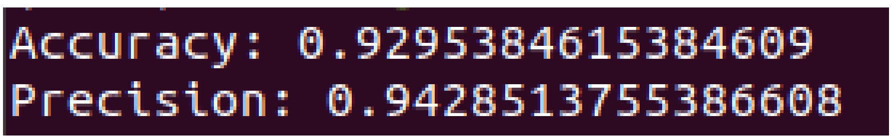


Figure 5.2: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 6

Neural Networks

6.1 Neuronal Networks on our data set

Pentru implementarea retelei neuronale am decis sa folosim 10 straturi de neuroni complet conectati. Fiecare dintre acesti neuroni analizeaza o parte din datele de intrale si se calibreaza pentru obtinerea clasificarii rezultatului obtinul la examenul final.
Daca in cadrul procesului de invatare rezultatul obtinut de reteaua neuronala este gresit se declanseaza o serie de schimbari a ponderilor care au indicat raspunsul eronat. In consecinta ponderile care au semnalat raspunsul gresit scad, iar cele care ar fi semnalat un raspuns corect cresc.
De exemplu, ponderile neuronilor care semnaleaza ca studentul nu ar promova examenul final analizeaza campurile "FJob", "MJob", "studytime", "scholarship" si "nursery" vor scadea in timp ce celelalte vor creste.

6.2 Experimental results

6.2.1 Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	81.50	84.39	84.45	85.38	86.95
Precision	78.36	84.93	85.93	86.67	87.48

Table 6.1: Full data set

6.2.2 Exclude G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	58.00	63.15	65.25	66.47	67.20
Precision	58.98	66.95	67.54	68.66	69.13

Table 6.2: Performance after removing G1 and G2 from data set

6.2.3 Select only G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	88.66	92.15	92.37	93.05	93.12
Precision	89.06	94.49	94.99	95.93	96.98

Table 6.3: Performance after feature selection only G1 and G2

6.3 Statistics of the results

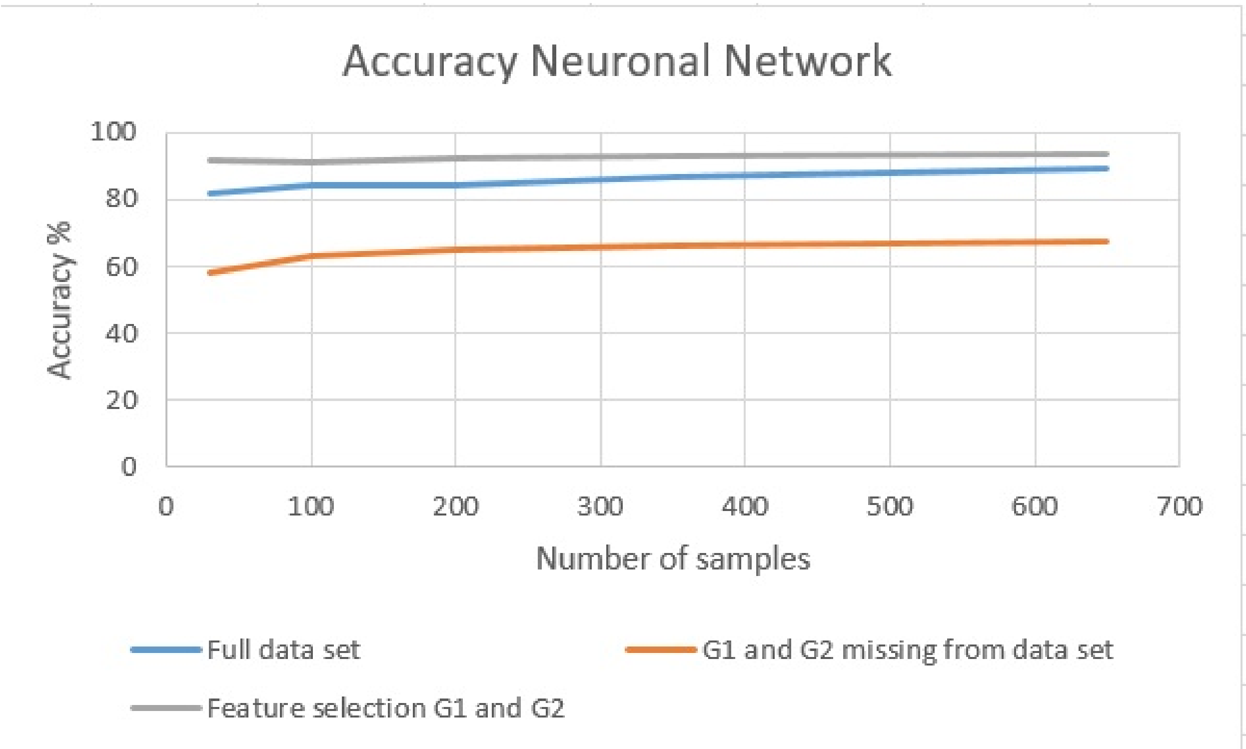


Figure 6.1: Analiza performantei retelei neuronale, invatare supervizata, dataset= 80% antrenare si 20% testare

6.4 Python code

See 10.5

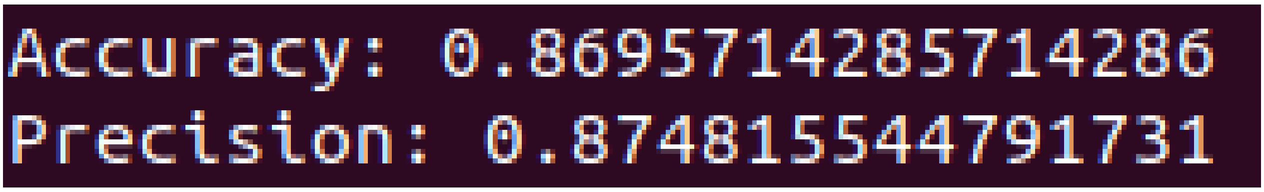

Accuracy: 0.8695714285714286
Precision: 0.874815544791731

Figure 6.2: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 7

Neural Networks with feature scaling

7.1 Neural Networks with feature scaling on our data set

Feature scaling este o metoda des folosita in cadrul clasificatorilor pentru a elimina caracteristicile care cele mai putin relevante in desemnarea rezultatului. Comparativ cu exemplul retelei neuronale prezentate anterior putem observa ca performanta acestui clasificator a crescut in toate cele 3 cazuri considerate, diferenta dintre acuratetea obtinuta din setul intreg de date si setul care utilizeaza feature selection este mai mica (comparativ cu reteaua neuronala prezentata anterior), iar timpul de rulare a fost mai scazut. Proprietatile cele mai putin relevante au fost: "Job-ul mamei", "Job-ul tatalui" si "travel time"

7.2 Experimental results

7.2.1 Full data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	81.66	85.24	86.39	86.67	87.48
Precision	82.63	85.73	87.25	88.56	86.63

Table 7.1: Full data set

7.2.2 Exclude G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	61.50	66.55	66.82	66.87	66.92
Precision	64.34	70.80	68.56	69.48	68.62

Table 7.2: Performance after removing G1 and G2 from data set,

7.2.3 Select only G1 and G2 from data set

	m = 30	m = 100	m = 200	m = 350	m = 650
Accuracy	92.83	92.82	92.48	92.64	93.57
Precision	98.52	97.86	98.37	98.22	98.30

Table 7.3: Performance after feature selection only G1 and G2

7.3 Statistics of the results

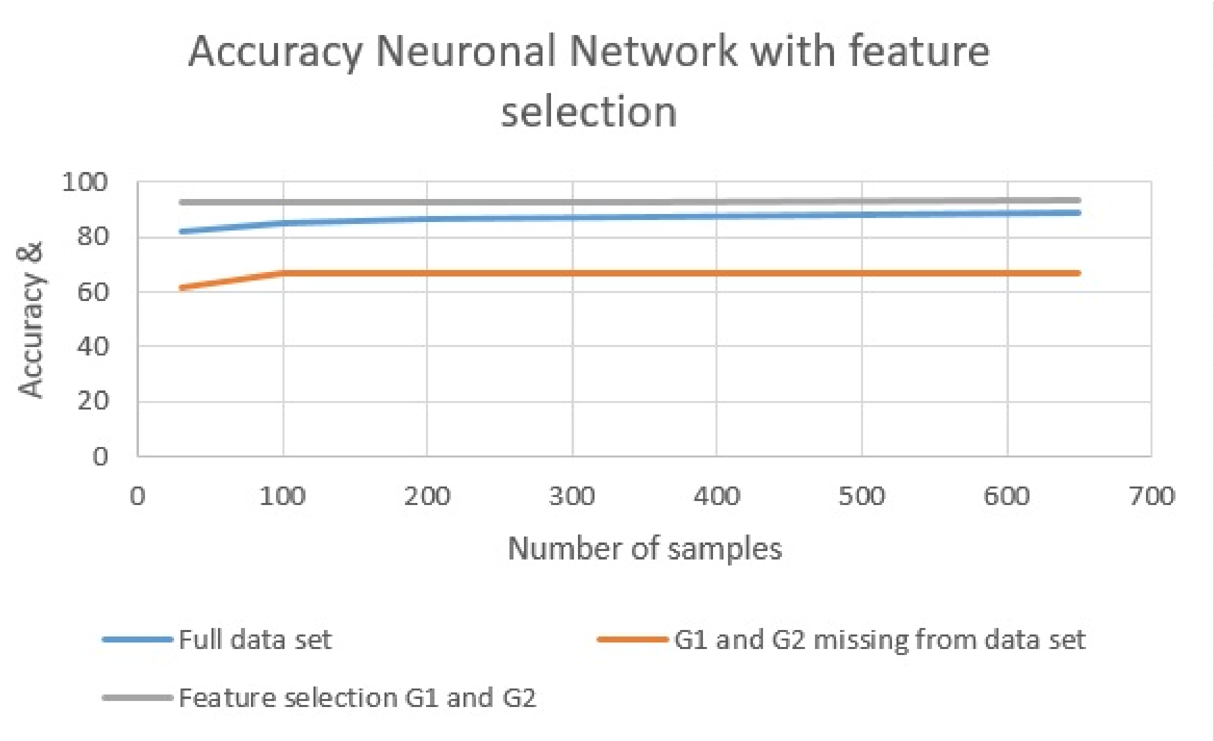


Figure 7.1: Analiza rezultatelor Retelei neuronale with feature scaling, invatare supervizata, dataset= 80% antrenare si 20% testare

7.4 Python code

See 10.7

Figure 7.2: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 8

AdaBoost ensemble method voting

In acest capitol vom experimenta tehnica ensemble method prin procesul de votare din cadrul unui clasificator de tip AdaBoost

8.1 Adaboost on our data set

Setul de date este divizat in mai multe blocuri de lungime egala. Primul bloc este folosit ca bloc de testare, iar restul sunt blocuri de antrenare.

8.1.1 Voting process

Pentru asamblare clasificatorului am decis sa folosim clasificatorii "Logistic Regression", "Random Forest" si "Naive Bayes".
Cei trei clasificatori se antreneaza cu intreg setul de date pentru a lua decizii independente.
In cadrul setului datelor de testare fiecare dintre cei 3 clasificatori clasifica datele de test, iar decizia finala a clasificatorului adaBoost este clasificarea care a obtinut cel mai mare numar de voturi.

Din figura 8.1 se poate observa ca acuratetea algoritmului este apropiata de cel mai performant clasificator, dar nu o depaseste deoarece numarul cazurilor in care doi clasificatori gresesc este considerabil. In concluzie am observat ca in cadrul acestei metode daca decizia gresita castiga votul majoritar aceasta va fi returnata de clasificatorul adaBoost, chiar daca o componenta a clasificat corect data de test.

8.2 Experimental results

Acestea sunt valorie obtinute prin rularea algoritmului AdaBoost prin varierea numarului de blocuri in care este impartit setul de date.
Pentru kfolds=2 setul de date este impartit in 50% date de andrenare si 50% date de testare.

Pentru kfolds=5 avem cazul standard de 80% date de antrenare si 20% date de testare

kfolds	2	3	5	6	10
Logistic Regression Accuracy	87.98	91.24	91.24	91.69	91.24
Random Forest Accuracy	90.91	91.84	91.97	92.05	92.15
Naive Bayes Accuracy	90.45	91.22	91.73	91.93	91.85
Ensemble (AdaBoost) Accuracy	90.14	91.84	92.46	92.57	92.00

Table 8.1: Performance after feature selection only G1 and G2

8.3 Statistics of the results

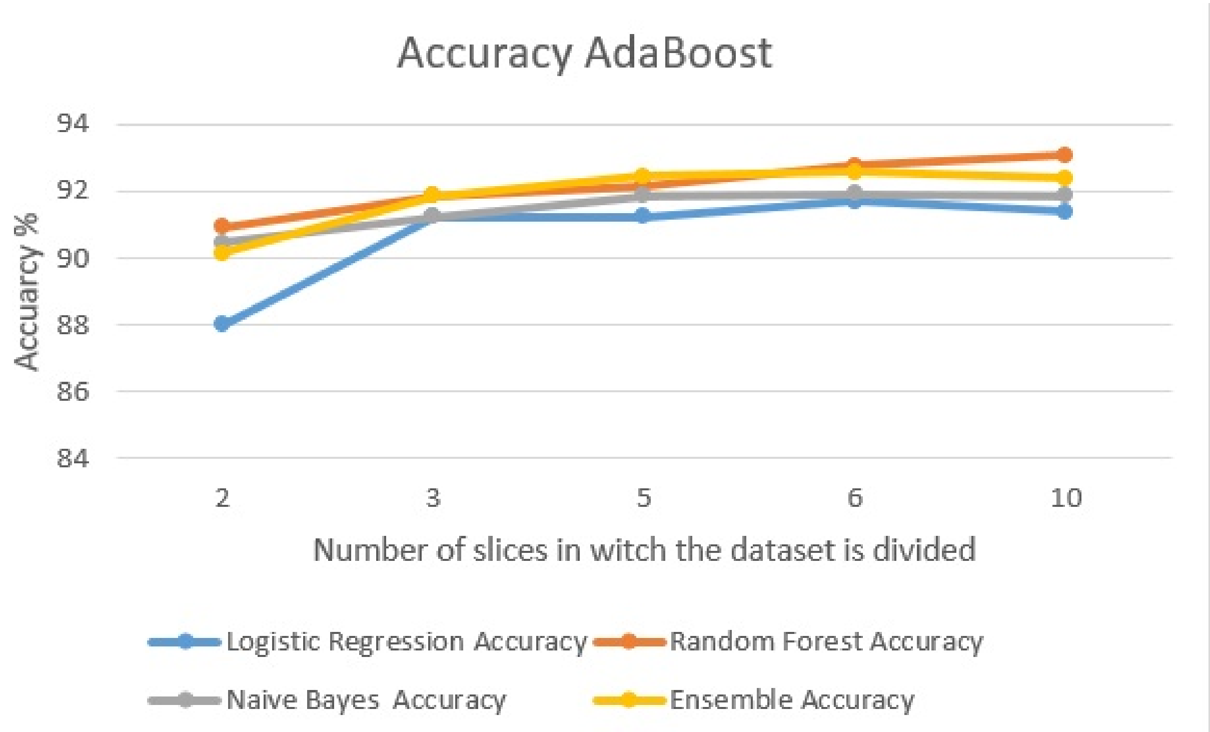
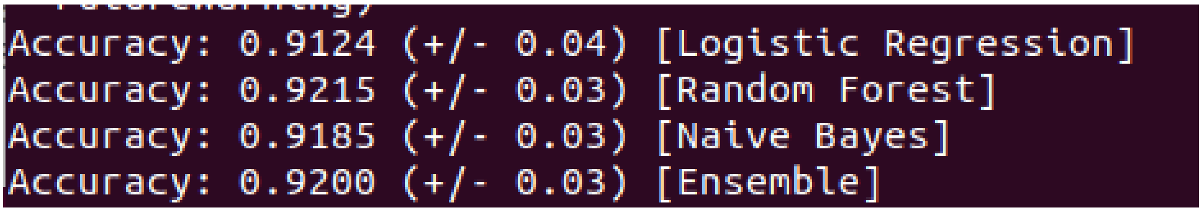


Figure 8.1: Analiza rezultatelor clasificatorului AdaBoost

8.4 Python code

See 10.8



```
Accuracy: 0.9124 (+/- 0.04) [Logistic Regression]
Accuracy: 0.9215 (+/- 0.03) [Random Forest]
Accuracy: 0.9185 (+/- 0.03) [Naive Bayes]
Accuracy: 0.9200 (+/- 0.03) [Ensemble]
```

Figure 8.2: Acuratetea si precizia pentru setul intreg de date (650 de exemple), invatare supervizata, dataset= 80% antrenare si 20% testare

Chapter 9

Conclusions

9.1 Compararea performantei algoritmilor in cazul setului de date intreg

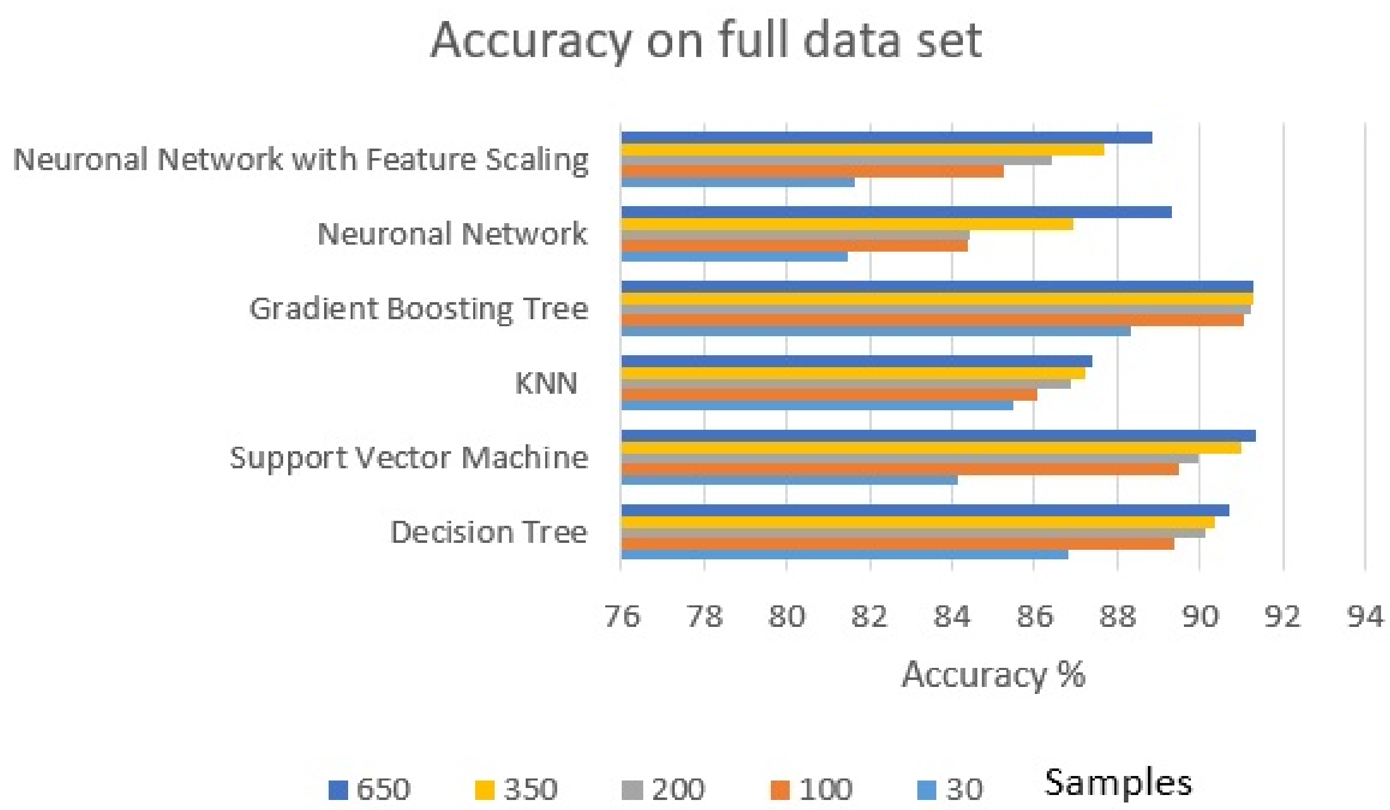


Figure 9.1: Analiza rezultatelor invatare supervizata, dataset= 80% antrenare si 20% testare

9.2 Compararea performantei algoritmilor in cazul lipsurilor de date

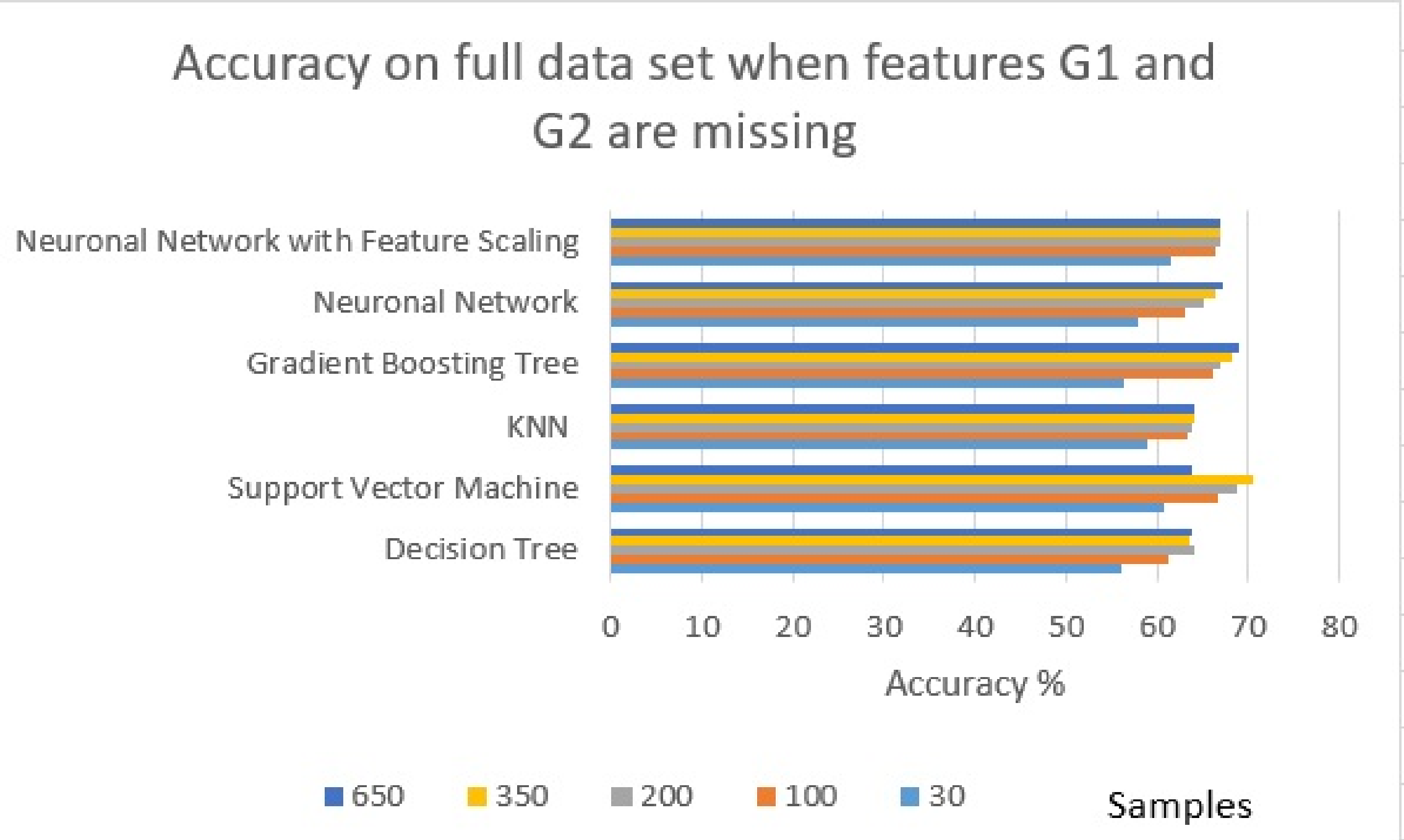


Figure 9.2: Analiza rezultatelor invatare supervizata, dataset= 80% antrenare si 20% testare

9.3 Compararea performantei algoritmilor cu feature selection

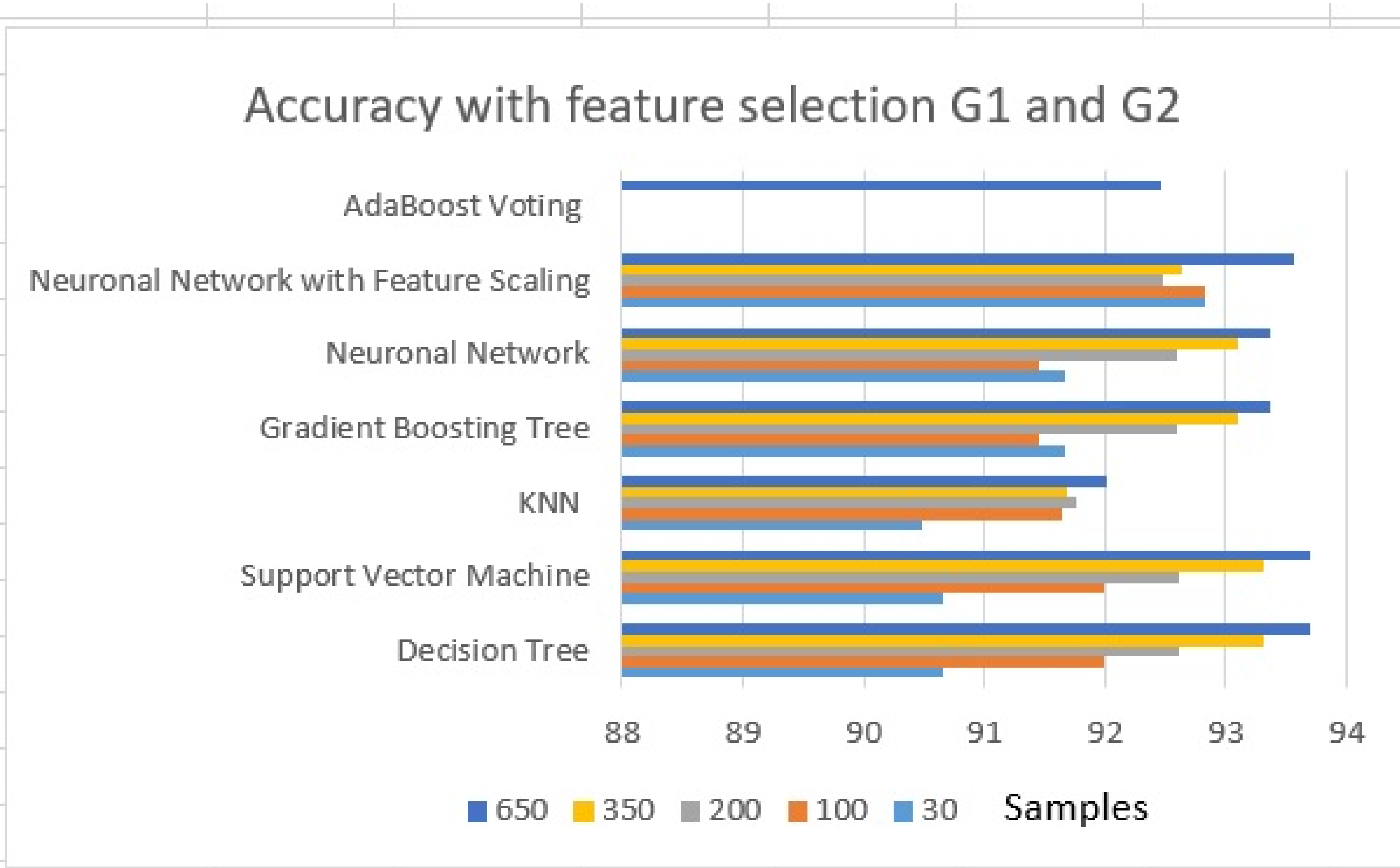


Figure 9.3: Analiza rezultatelor: invatare supervizata, dataset= 80% antrenare si 20% testare

9.4 Final thoughts

Graficele obtinute sugereaza faptul ca Support Vector Machine (SVM) ofera cea mai buna predictie a promovarii examenului final (G3 cel putin 12). In figura 9.1 s-a observat cum Decions Trees with gradient Boosting reusea sa surclaseze SVM-ul pe datele preluate ın forma bruta, cu toate acestea prin feature selection SVM-ul reuseste sa obtina nu doar o predictie mai buna, ci chiar cea mai buna acuratete din toate cazurile.

Chapter 10

Appendix

10.1 Decision Tree implementation

```python
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0

  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that
   contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
   contains all the features
    clf = tree.DecisionTreeClassifier()
    le = LabelEncoder()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]

    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']
```

```

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)

  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS

evaluate()
```

Listing 10.1: Decision Tree implementation

10.2 Support Vector Machine implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn import svm
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0

  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that
   contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
   contains all the features
    clf = svm.SVC()
    le = LabelEncoder()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]

    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']

    clf.fit(x_train, y_train)
```

```

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)


  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS

evaluate()
```

Listing 10.2: Support Vector Machine implementation

10.3 K-Nearest Neighbor implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0


  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that
   contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
   contains all the features

    clf = KNeighborsClassifier(n_neighbors=3)
    le = LabelEncoder()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]


    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']
```

```

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)

  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS


evaluate()
```

Listing 10.3: K-Nearest Neighbor implementation

10.4 Gradient Tree Boosting implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingClassifier

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0


  for i in range(0, NUMBER_OF_ITERATIONS):
    #input_file = "student_grades.csv"  this line reads the file that
   contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
   contains all the features
    clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
   max_depth=5, random_state=0)
    le = LabelEncoder()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]
```

```

    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)


  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS


evaluate()
```

Listing 10.4: Gradient Tree Boosting implementation

10.5 Neuronal Network implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0


  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that contains all the features
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10, 10), random_state=1)
    le = LabelEncoder()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
```

```
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]

    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)


  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS

evaluate()
```

Listing 10.5: Neuronal Network implementation

10.6 Neural Networks with feature scaling implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0


  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that
   contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
   contains all the features
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,
   10), random_state=1)
    le = LabelEncoder()
    scaler = StandardScaler()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)
```

```

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]
    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']

    scaler.fit(x_train)
    x_train = scaler.transform(x_train)
    x_test = scaler.transform(x_test)
    clf.fit(x_train, y_train)
    predicted = clf.predict(x_test)

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)


  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS

evaluate()
```

Listing 10.6: Neural Networks with feature scaling implementation

10.7 Neural Networks with feature scaling implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle

NUMBER_OF_ITERATIONS = 100
NUMBER_OF_EXAMPLES = 650

def evaluate():

  accuracy = 0
  precision = 0


  for i in range(0, NUMBER_OF_ITERATIONS):
    # input_file = "student_grades.csv"  this line reads the file that
```

```
    contains only the grades set
    input_file = "student_dataset.csv" # this line reads the file that
    contains all the features
    clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,
    10), random_state=1)
    le = LabelEncoder()
    scaler = StandardScaler()

    data = pd.read_csv(input_file, header = 0)
    data = shuffle(data)

    data = data[0:NUMBER_OF_EXAMPLES]
    data = data.apply(le.fit_transform)
    delim = int(len(data) * 0.8)
    # 80% training data
    data_train = data[0:delim]
    # 20% test data
    data_test = data[delim:len(data)]
    x_train = data_train[data_train.columns.drop('G3')]
    y_train = data_train['G3']
    x_test = data_test[data_test.columns.drop('G3')]
    y_test = data_test['G3']

    scaler.fit(x_train)
    x_train = scaler.transform(x_train)
    x_test = scaler.transform(x_test)
    clf.fit(x_train, y_train)
    predicted = clf.predict(x_test)

    clf.fit(x_train, y_train)

    predicted = clf.predict(x_test)

    accuracy += accuracy_score(y_test, predicted)
    precision += precision_score(y_test, predicted)


  print "Accuracy:", accuracy / NUMBER_OF_ITERATIONS
  print "Precision:", precision / NUMBER_OF_ITERATIONS

evaluate()
```

Listing 10.7: Neural Networks with feature scaling implementation

10.8 AdaBoost voting implementation

```
import os
import subprocess
import pandas as pd
import numpy as np

from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier


input_file = "student_grades.csv"
```

```

data = pd.read_csv(input_file, header = 0)

X, y = data[data.columns.drop('G3')], data['G3']


clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=15, random_state=1)
clf3 = GaussianNB()


eclf = VotingClassifier(
    estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)],
    voting='hard')

for clf, label in zip([clf1, clf2, clf3, eclf], ['Logistic Regression', 'Random Forest', 'naive Bayes', 'Ensemble']):
  scores = cross_val_score(clf, X, y, scoring='accuracy', cv=2)
  print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

Chapter 11

Bibliografie

1. https://archive.ics.uci.edu/ml/datasets/student+performance
2. http://scikit-learn.org/stable/
3. Artificial Intelligence: A Modern Approach
4. /wiki/Decision$_tree/wiki/Support_vector_machine$
5. /wiki/K-nearest$_neighbors_algorithm/wiki/Gradient_boosting$
6. /wiki/Artificial$_neural_network/wiki/Feature_scaling$
7. /wiki/AdaBoost
8. C.M. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2006

Intelligent Systems Group