Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2021-2022

Machine Learning
Project title: Banknote Authentication

Name:Sirbu Oana-Bianca
Group:30234
Email:sirbuoanabianca@gmail.com

Asst. Prof. Eng. Roxana Ramona Szomiu
Roxana.Szomiu@cs.utcluj.ro

Contents

Chapter 1

Project description

1.1 Introduction

Tema proiectului consta in prezentarea metodelor de clasificare asupra setului de date "autenticitatea bancnotelor" si determinarea parametrilor optimi pentru diversi algoritmi de Machine Learning cu scopul de a obtine modele performante in prezicerea tipului unei bancnote(autentica sau falsa).

Setul de date pentru bancnote implica prezicerea daca o anumita bancnota este autentica, avand in vedere anumite proprietati masurate dintr-o fotografie.

Datele sunt extrase din imagini care au fost capturate folosind bancnote autentice si falsificate. Pentru digitizare s-a folosit o cameră industrială folosită de obicei pentru inspectia tiparirii. Imaginile finale au o rezolutie de 400x400 pixeli. Datorită lentilei obiectului si distanței față de obiectul investigat, au fost obtinute imagini Gray-scale cu o rezolutie de aproximativ 660 dpi. Instrumentul Wavelet Transform a fost folosit pentru a extrage caracteristici din imagini.

1.2 The main goal of the project

Deoarece se doreste prezicerea autenticitatii unei bancnote (fals sau adevarat), aceasta devine o problema de **clasificare** binara.Setul de date contine 1372 de inregistrari cu 4 informatii preluate din masurarea mai multor fotografii si un target:autentica sau nu.

1.3 Dataset

Feature-uri:

- Variance of Wavelet Transformed image (continuous).
- Skewness of Wavelet Transformed image (continuous).
- Kurtosis of Wavelet Transformed image (continuous).
- Entropy of image (continuous).

Target: Class (0 for authentic, 1 for inauthentic).
Sources and examples from the dataset:
https://archive.ics.uci.edu/ml/datasets/banknote+authentication

Chapter 2

Implementation details

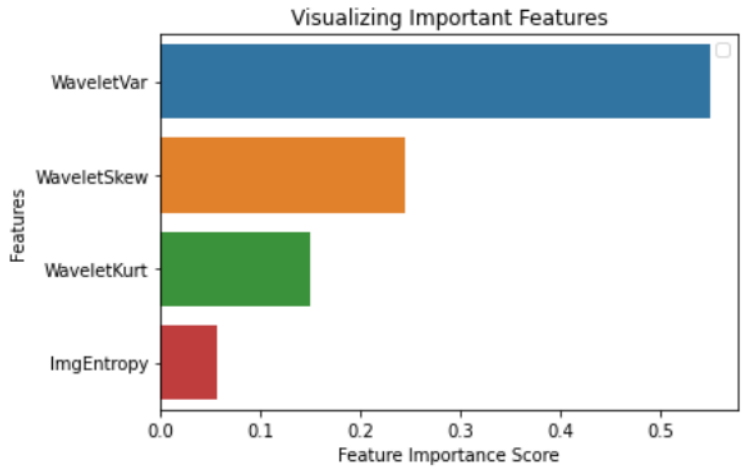
Incarcare dataset

	WaveletVar	WaveletSkew	WaveletKurt	ImgEntropy	Authenticity
0	4.54590	8.1674	-2.45860	-1.46210	0
1	3.86600	-2.6383	1.92420	0.10645	0
2	3.45660	9.5228	-4.01120	-3.59440	0
3	0.32924	-4.4552	4.57180	-0.98880	0
4	4.36840	9.6718	-3.96060	-3.16250	0
...	...	...	...	...	...
1365	-2.41000	3.7433	-0.40215	-1.29530	1
1366	0.40614	1.3492	-1.45010	-0.55949	1
1367	-1.38870	-4.8773	6.47740	0.34179	1
1368	-3.75030	-13.4586	17.59320	-2.77710	1
1369	-3.56370	-8.3827	12.39300	-1.28230	1

Feature-uri:

- Variance of Wavelet Transformed image(in matematica, o serie de wavelet este o reprezentare a unei functii integrabile în patrat (real sau complex) printr-o anumita serie ortonormala generata de o wavelet(oscilatie))
- Skewness of Wavelet Transformed image(deformarea imaginii wavelet transformate)
- Kurtosis of Wavelet Transformed image
- Entropia imaginii pe care s-au facut masuratorile de mai sus

Data exploration - caracteristica ce influenteaza cel mai mult decizia de clasificare este varianta Wavelet, avand o pondere mai mare de 50%.



2.1 Algorithm 1 : Decision Tree Classifier

Estimatorul Decision Tree se bazeaza pe creerea unui model care prezice valoarea unuei variabile target prin invatarea unor reguli de decizie deduse din feature-uri.Pentru a prezice clasa din care face parte o inregistrare se incepe de la radacina arborelui de decizie.Se compara valorile atributului radacina cu atributul inregistrarii. Pe baza comparatiei, urmarim ramura corespunzatoare acelei valori si sarim la următorul nod.

Avantaje pentru DecisionTree Classifier:simplu si necesita mai putin efort in ceea ce priveste pregatirea datelor in timpul preprocesarii (nu necesita normalizarea/scalarea datelor).Deoarece arborele de decizie nu este potrivit pentru probleme de regresie, clasificarea dataset-ului prezentat este indeplinit de DecisionTree.Dezavantajul algoritmului este timpul mare pentru antrenamentul modelului si tendinta de overfitting.

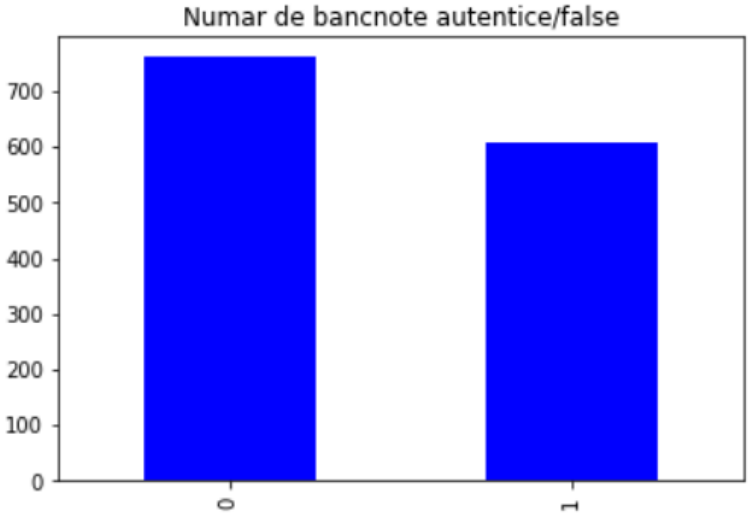
Preprocesarea datelor

DecisionTree nu necesita normalizare,scalare,standardizare sau encodare a datelor (valorile sunt intregi, deci nu trebuie encodate).

Cross-validation - evalueaza performanta modelului, cu scopul de a evita overfitting-ul.Setul de date de training a fost impartit in 5 parti egale si, iterativ, se face antrenamentul pe 4 din parti, evaluearea finala facandu-se pe partea ramasa. Rezultatele fiecarei evaluari sunt: [0.98636364, 0.97260274, 0.98630137, 0.98173516, 0.98630137] Pentru ca scorul este bun, nu se pune problema overfitting-ului.

Data exploration

Din grafic, se observa ca datasetul contine un numar echilibrat de inregistrari de bancnote autentice si falsificate.



Prezentare set de training si set de test

Setul de date a fost impartit automat (train-test-split) in set de training si set de test,dintre care 20% din dataset este alocat pentru testare.

Aplicare algoritm pe dataset

Pentru antrenarea modelului, am folosit atat parametrii impliciti ai arborelui de decizie, cat si GridSearch care a gasit valorile cele mai bune pentru parametrii pentru o performanta mai mare.

Parametrii impliciti:

-criterion=”gini”- functie care masoara calitatea unei impartiri in arbore

-splitter=”best” - strategie utilizata la alegerea impartirii unui nod

-max depth=None - adancimea maxima a arborelui.Daca nu exista, atunci nodurile sunt extinse pana cand toate frunzele sunt pure

Parametrii gasiti de GridSearch:

-criterion=”gini”

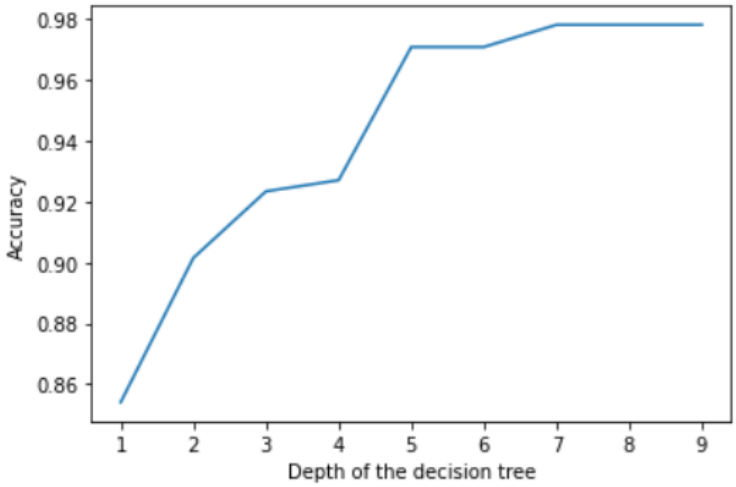
-splitter=”random”

-max depth=12

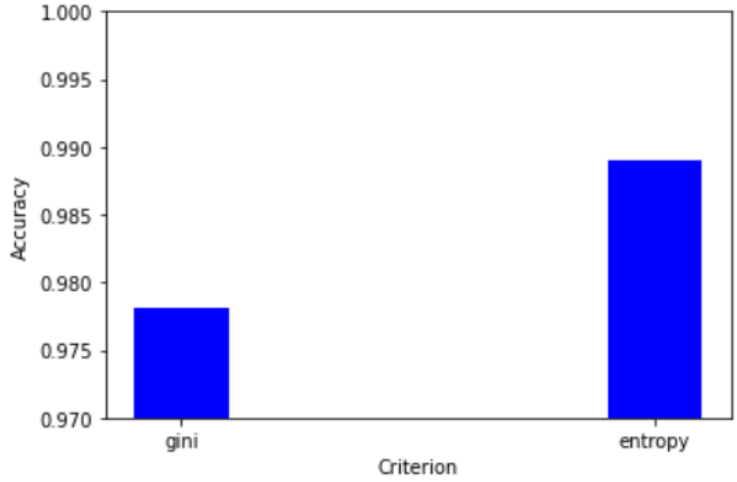
Imbunatatirea performantei prin varierea parametrilor

Antrenarea modelului pe alti parametri decat cei default a fost facuta pe cei obtinuti de GridSearch prezentati mai sus, obtinandu-se acuratetea=0.9963.. si precizia=0.9915.. De asemenea, modelul a fost antrenat atat pe clasificatorul cu parametrii default,cei obtinuti de GridSearch, cat si pe variatia parametrului ”criterion” si ”max depth”.Mai jos avem o reprezentare a rezultatelor obtinute variind ”criterion”=”gini” sau ”entropy” si max depth=[1,10)

Vizualizare rezultate obtinute prin variatia parametrilor



Acest grafic a fost obtinut prin variatia parametrului max depth = inaltimea maxima a arborelui in intervalul [1,10).Se observa faptul ca acuratetea estimatorului cu atat este mai buna cu cat adancimea arborelui e mai mare, la adancimea maxima stabilizandu-se la acuratetea 0.98..



Acest grafic a fost generat dupa variatia parametrului criterion, iar rezultate mai bune dau modelele care au setat parametrul criterion ca "entropy",apropiindu-se de 0.990..

Testare model pe 2 exemple concrete

```
my_test= [[3.6216,8.6661,-2.8073,-0.44699],[-2.5419,-0.65804,2.6842,1.1952]]
Y_prediction_test= model.predict(my_test)
print("Prediction form my given test:",Y_prediction_test)
```

```
Prediction form my given test: [0. 1.]
```

Modelul a fost testat initial pe 2 exemple din setul de date care au fost scoase ulterior.Rezultatul asteptat pentru prima inregistrare=0 (bancnota autentica), iar pentru a 2-a inregistrare=1 (bancnota falsificata).

Rezultate obtinute pe baza metricilor

Metrici obtinute pe clasificatorul cu parametri default:

-Accuracy: 0.9781

-Confusion matrix is : [[154 3] [3 114]]

-Recall score is : 0.97435

-Precision Score is : 0.97435

-f1 Score is : 0.97435

Metrici obtinute pe clasificatorul cu parametri obtinuti de GridSearch:

-Accuracy: 0.9963

-Confusion matrix is : [[156 1] [0 117]]

-Recall score is : 1.0

-Precision Score is : 0.9915

-f1 Score is : 0.9957

Pe baza rezultatelor, concluzia dupa aplicarea algoritmului DecisionTree, variind parametrii, este ca performanta cea mai buna este gasita de GridSearch si nu schimband parametrii aleator.Confusion matrix arata cate predictii gresite au fost facute:cu parametrii optimi o singura predictie este gresita, iar cu parametrii aleatori sunt 6 predictii gresite.

2.2 Algorithm 2 : KNearestNeighbors

Atat pentru clasificare, cat si pentru regresie, o tehnica utila poate fi atribuirea de ponderi contributiilor vecinilor, astfel încat vecinii mai apropiati să contribuie mai mult la medie decat cei mai îndepartati. K din numele acestui clasificator reprezintă k vecini cei mai apropiati. Prin

urmare, asa cum sugereaza si numele, acest clasificator implementeaza învatarea bazata pe cei mai apropiati k vecini. Alegerea valorii lui k depinde de dataset.

Avantajul acestui algoritm: este foarte usor de implementat. Este nevoie de doar doi parametri pentru a implementa KNN, adica valoarea lui K si functia de distanta (de exemplu, Euclideana sau Manhattan).Nu exista o perioada de antrenament pentru KNN. Stochează setul de date de antrenament si învata din el doar în momentul realizarii predictiilor în timp real. Acest lucru face ca algoritmul KNN sa fie mult mai rapid decât alti algoritmi care necesita antrenament.

Dezavantajul lui KNN este ca necesita neaparat scalarea datelor(standardizare si normalizare) inainte de aplicare, altfel va face predictii gresite.De asemenea,algoritmul devine semnificativ mai lent pe masura ce volumul de date creste.

Preprocesarea datelor

Standardizarea/normalizarea unui set de date este o cerinta comuna pentru multi estimatori de Machine Learning: acestia pot da predictii gresite daca datasetul nu arata intr-o forma standard cunoscuta de ei.

Cross-validation - Setul de date de training a fost impartit in 5 parti egale si, iterativ, se face antrenamentul pe 4 din parti, evaluearea finala facandu-se pe partea ramasa. Rezultatele fiecarei evaluari sunt maxime: [1., 1., 1., 1., 1.]

Prezentare set de training si set de test

Setul de date a fost impartit automat (train-test-split) in set de training si set de test,dintre care 30% din dataset este alocat pentru testare.Inainte de aplicarea modelului, s-a realizat scalarea si normalizarea atat pe setul de training, cat si pe cel de test.

Aplicare algoritm pe dataset

Pentru antrenarea modelului, am folosit atat parametrii impliciti ai Kneighbors, cat si GridSearch care a gasit valorile cele mai bune pentru parametrii pentru o performanta mai mare.

Parametrii impliciti:

-n neighbors = 5 - numar de vecini pe care sa caute pentru un punct

-weights=uniform - toate punctele din fiecare vecinatate sunt ponderat egale

Parametrii gasiti de GridSearch:

-n neighbors = 5

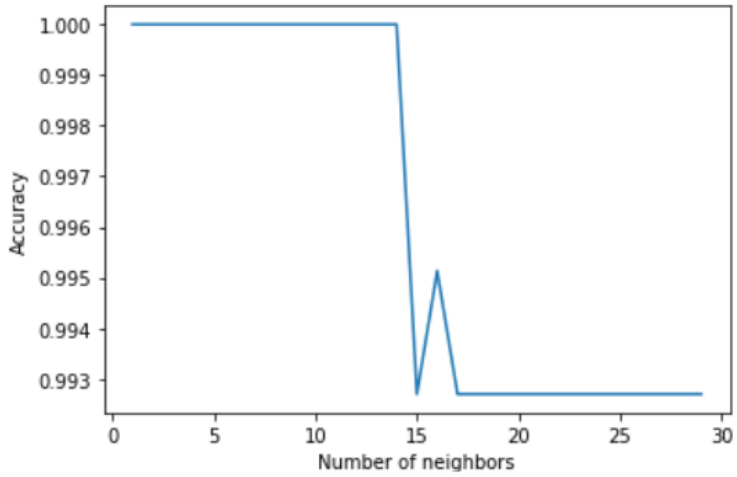
-weights=uniform

Parametrii impliciti raman cei care dau rezultate optime.

Imbunatatirea performantei prin varierea parametrilor

Deoarece parametrii optimi gasiti de GridSearch sunt cei default, antrenarea modelului a fost facuta pe acestia.

Vizualizare rezultate obtinute prin variatia parametrului neighbors



Graficul a fost obtinut prin variatia numarului de vecini intr-o bucla iterativa, luand valori intre 0-30. Din grafic se deduce faptul ca valoarea optima pentru numarul de vecini este in intervalul (0,15) cu o acuratete maxima, insa chiar daca numarul de vecini este mai mare, tot se obtin rezultate bune 0.99..

Rezultate obtinute pe baza metricilor Kneighbors vs RadiusNeighbors

GridSearch a gasit ca si parametri optimi pe cei default de la Kneighbors.Am obtinut urmatoarele scoruri:

Kneighbors Accuracy: 1.0

Confusion matrix is : [[220 0] [0 191]]

Recall score is : 1.0

Precision Score is : 1.0

f1 Score is : 1.0

RadiusNeighbors Accuracy: 0.9927

Confusion matrix is : [[226 3] [0 182]]

Recall score is : 1.0

Precision Score is : 0.9837

f1 Score is : 0.9918

In confusion matrix:

-pentru Kneighbors-se observa cele 2 valori de 0 care indica faptul ca nu a fost facuta nicio predictie gresita.

-pentru radiusNeighbors-au fost prezise gresit 3 inregistrari.

Datele sunt uniform esantionate, iar in acest caz, Kneighbors este alegerea cea mai buna.

2.3 Ensemble Method 1 :Bagging

RandomForestTree Classifier

Clasificatorul Random forest este un algoritm de învatare supervised si construieste un ansamblu de arbori de decizie, de obicei antrenați cu metoda "Bagging".

Algoritmul construiește mai multi arbori de decizie si ii combina pentru a obtine o predictie mai precisa si mai stabilă.In loc sa caute cea mai importanta caracteristică in timp ce divizeaza un nod, cauta cea mai buna caracteristica dintr-un subset aleator de caracteristici. Acest lucru are ca rezultat o diversitate care duce, in general, la un model mai bun.

Un avantaj al acestui algoritm ar fi evitarea overfittingului.Dacă exista destui arbori, clasificatorul nu va face overfitting, deoarece mai multi weaklearners se antreneaza pe bucati din dataset.Dezavantajul sau este ca un numar prea mare de arbori poate face algoritmul ineficient.O predictie mai precisa necesita mai multi arbori, ceea ce duce la un model mai lent.

ExtraTrees Classifier

Extra Trees este ca Random Forest, prin faptul ca se construiesc mai multi arbori si imparte nodurile folosind subseturi aleatoare de feature-uri, dar cu doua diferente: adică esantionează fara înlocuire in setul de date de antrenament, iar nodurile sunt impartite pe bucati aleatoare intre un subset aleatoriu de caracteristici selectate la fiecare nod. Extra Trees este mult mai rapid. Acest lucru se datoreaza faptului ca, in loc sa caute diviziunea optima la fiecare nod, o face aleatoriu.

Preprocesarea datelor

Cross-validation - Setul de date de training a fost impartit in 5 parti egale si, iterativ, se face antrenamentul pe 4 din parti, evaluearea finala facandu-se pe partea ramasa. Rezultatele fiecarei evaluari sunt:

[0.9927 0.9927 0.9890 0.9963 0.9963] -randomForest

[1. 1. 0.9963 0.9963 1.] -extraTrees

Prezentare set de training si set de test

Setul de date a fost impartit automat (train-test-split) in set de training si set de test,dintre care 20% din dataset este alocat pentru testare.

Aplicare algoritm pe dataset

Pentru antrenarea modelului randomForest si extraTrees, am folosit atat parametrii impliciti ai clasificatorilor, cat si GridSearch care a gasit valorile cele mai bune pentru parametrii pentru o performanta mai mare.

Parametrii impliciti:

-n estimators=100 - numarul total al arborilor

-criterion="gini"- functie care masoara calitatea unei impartiri in arbore

-max depth=None - adancimea maxima a arborelui.Daca nu exista, atunci nodurile sunt extinse pana cand toate frunzele sunt pure

Parametrii gasiti de GridSearch pentru RandomForest:

-criterion="entropy"

-n estimators=9 din [5,15]

-max depth=8 din [5,10]

Intervalul de valori pentru numarul de estimatori si adancime este restrans, deoarece timpul de executie creste odata cu numarul de arbori.

Parametrii gasiti de GridSearch pentru ExtraTrees:

-criterion="gini"

-n estimators=12 din [5,15]

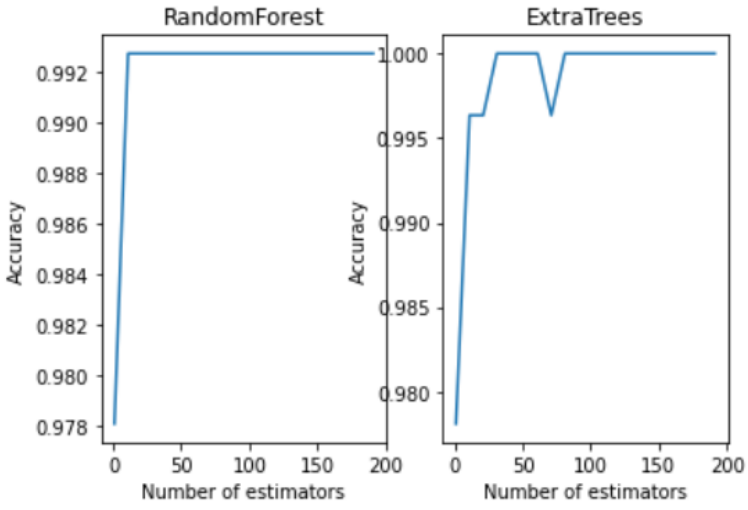
-max depth=9 din [5,10]

Imbunatatirea performantei prin varierea parametrilor

Antrenarea modelului pe alti parametri decat cei default a fost facuta pe cei obtinuti de GridSearch prezentati mai sus, obtinandu-se acuratetea=0.9975.. si precizia=0.9947..

Vizualizare rezultate obtinute prin variatia parametrilor

Graficul urmator a fost obtinut variind numarul de estimatori in intervalul (1-200), reiesind faptul ca randomForest se stabilizeaza la valoarea 0.992.. pe masura ce numarul de estimatori creste.In comparatie cu randomForest, algoritmul ExtraTrees atinge acuratete maxima si ramane stabil la aceasta valoare cand numarul de estimatori depaseste 100.Astfel, ExtraTrees are cele mai bune raspunsuri.



Rezultate obtinute pe baza metricilor RandomForest vs ExtraTrees

In urma gasirii celor mai bune valori pentru parametri cu GridSearch, s-au obtinut scorurile metricilor:

RandomForest Accuracy: 0.9975

Confusion matrix is : [[221 1] [0 189]]

Recall score is : 1.0

Precision Score is : 0.9947
f1 Score is : 0.9973
ExtraTrees Accuracy: 0.9975
Confusion matrix is : [[222 0] [1 188]] Recall score is : 0.9947
Precision Score is : 1.0
f1 Score is : 0.9973

In confusion matrix - ambii algoritmi au prezis gresit o inregistrare.Rezultatele metricilor sunt fin asemanatoare,asadar pe baza metricilor ambii algoritmi dau rezultate optime.

2.4 Ensemble Method 2 :Boosting

AdaBoost Classifier

Este posibil ca un singur clasificator sa nu poata prezice cu precizie clasa unui obiect, dar daca grupam mai multi clasificatori slabi, fiecare invatand progresiv din ceea ce ceilalti au clasificat gresit, putem construi un model puternic.Clasificatorul de baza este decisionTree. Adaboost este mai puțin predispus la overfitting, deoarece parametrii de intrare sunt executati secvential.

Principalul dezavantaj al acestuia este ca are nevoie de un set de date de calitate.De aceea am obtinut acuratete maxima cu acest algoritm pe datasetul prezentat.

GradientBoost Classifier

Clasificatorul gradientBoosting este construit într-un mod pe etape, ca si în alte metode de boosting, dar generalizeaza celelalte metode permitand optimizarea printr-o functie 'loss' arbitrara. Avantaj: ofera acuratete mai mare comparativ cu alti algoritmi si se antreneaza rapid pe dataseturi mari. Dezavantaj:predispus la overfitting

Preprocesarea datelor

K-Folds cross-validation - Setul de date de training a fost impartit in 10 parti egale si, iterativ, se face antrenamentul pe 9 din parti, evaluearea finala facandu-se pe partea ramasa. Media rezultatelor fiecarei evaluari sunt:

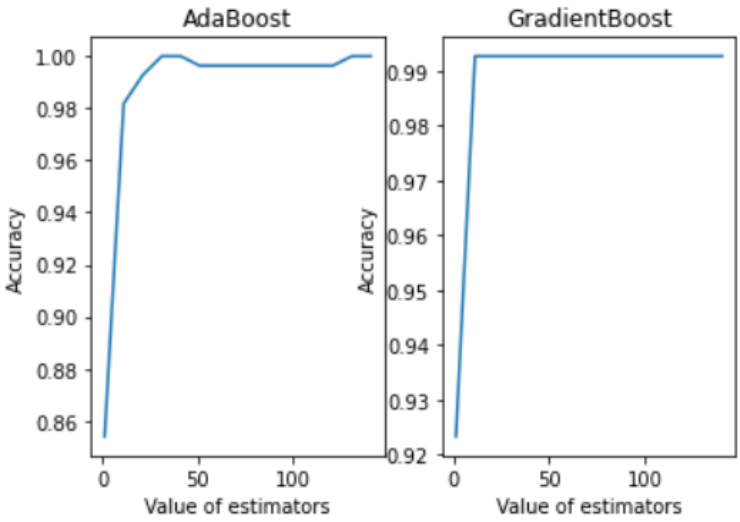
0.9956 - adaBoost
0.9948 - gradientBoost

Rezultate obtinute pe baza metricilor AdaBoost vs GradientBoost

AdaBoost cu parametrii default Accuracy: 1.0
Confusion matrix is : [[157 0] [0 117]]
Recall score is : 1.0
Precision Score is : 1.0
f1 Score is : 1.0
GradientTree cu parametrii default Accuracy: 1.0
Confusion matrix is : [[157 0] [0 117]]
Recall score is : 1.0
Precision Score is : 1.0
f1 Score is : 1.0

Vizualizare rezultate obtinute prin variatia parametrilor

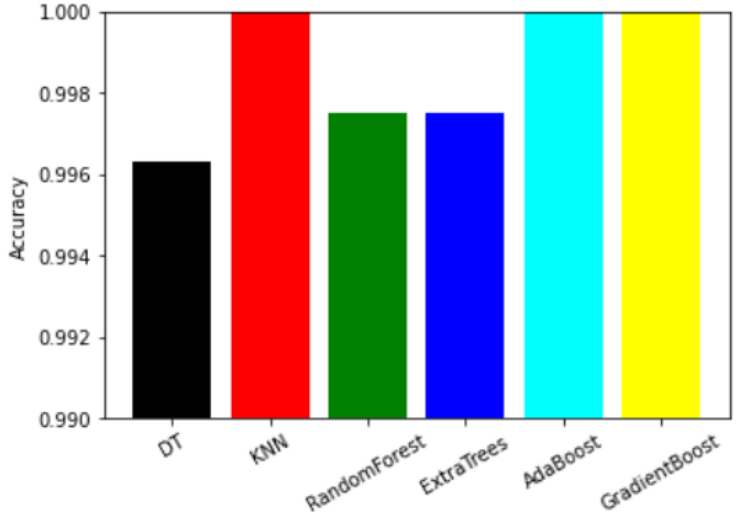
In urma variatiei numarului de estimatori in intervalul (1-150), rezultatele acuratetii sunt aproape la fel de bune, AdaBoost atingand acuratete maxima, iar GradientBoost stabilizandu-se la valoarea 0.99..


AdaBoost
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0
50
100
Value of estimators
GradientBoost
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0
50
100
Value of estimators

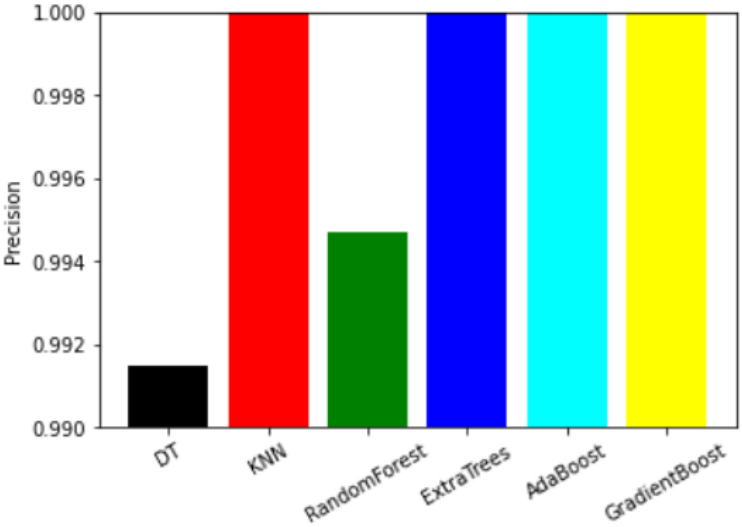
Chapter 3

Analysis of results - Comparisons

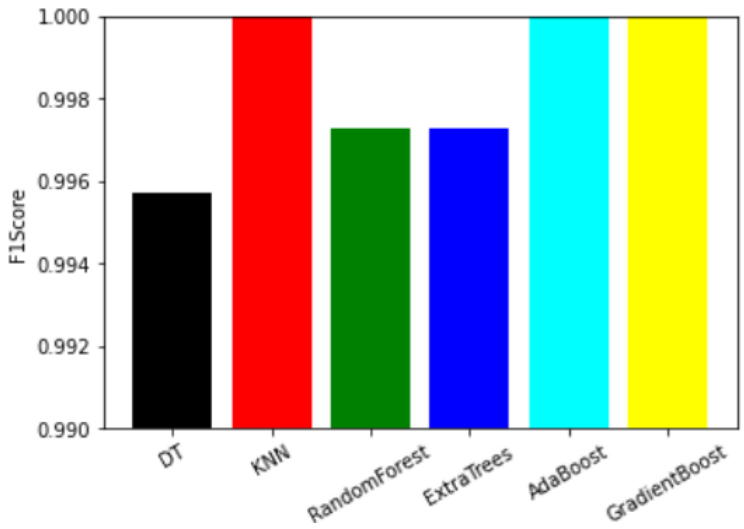
Acuratete maxima:



Precizie maxima:



F1 scor maxim:



Pe baza metricilor, pentru setul de date ales, cei mai buni algoritmi de clasificare cu acuratete maxima sunt KNearestNeighbors, AdaBoost si GradientBoost.

Metode de optimizare folosite:

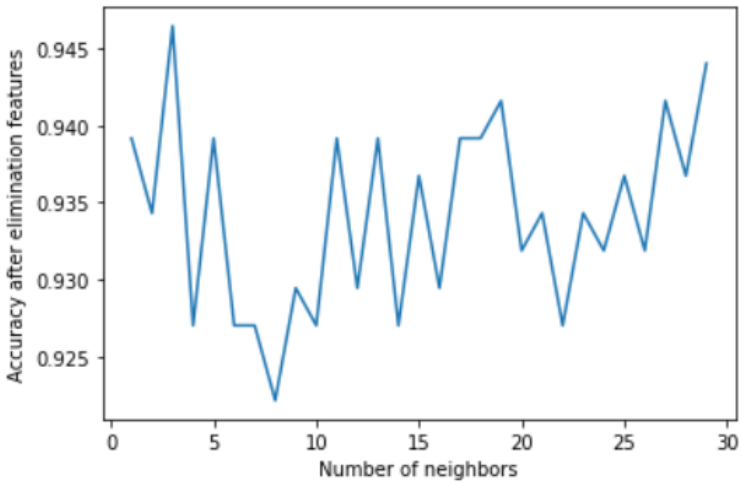
Pentru cei 6 algoritmi am aplicat cele 5 metrici inainte si dupa optimizare(optimizare parametri prin GridSearch).

Exemplu pentru clasificatorul DecisionTree:

- acuratete - inainte:0.9781 — dupa: 0.9963
- recall score - inainte:0.9743 — dupa: 1.0
- precizie - inainte:0.9743 — dupa: 0.9915
- f1 score - inainte:0.9743 — dupa: 0.9957

Rezultate dupa eliminarea celor mai nesemnificative 2 coloane :ImgEntropy si WaveletKurt

KNearest Neighbors:



In graficul prezentat mai sus, pentru clasificatorul KNN, avand toate feature-urile, s-a observat ca un numar de estimatori mai mic decat 15 se apropia de acuratete maxima.Daca se scoate cel mai nesemnificativ feature, putem observa din graficul alaturat ca rezultatele devin tot mai slabe,nu depasesc acuratetea de 0.94.Concluzia:In ciuda faptului ca feature-ul care a fost scos din dataset avea importanta minima conform Data Exploration, omiterea lui a dus la diminuarea performantei.

Chapter 4

Conclusion

In concluzie, antrenand mai multe modele pe setul de date cu informatii despre proprietatile autenticitatii bancnotelor, s-a dovedit ca datele sunt uniform esantionate,iar cei 6 algoritmi testati au afisat rezultate foarte bune, clasificatorii KNearestNeighbors, AdaBoost si Gradient-Boost avand acuratete 100%.
Totusi, daca se incearca verificarea stabilitatii algoritmului in urma eliminarii feature-ului cel mai nesemnificativ regasit in Data Exploration, performanta algoritmului KNearestNeighbor scade.

Appendix A

Your original code

```
#!/usr/bin/env python
# coding: utf-8

## Boosting

# In[31]:


# Boosting Algorithms

import pandas as pd
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import *

#Incarcare dataset
input_file = "data_banknote_authentication.txt"
data = pd.read_csv(input_file, header = 0)

array = data.values
X = array[:,0:4]
Y = array[:,4]
data
seed = 7
num_trees = 30


# In[32]:


X.shape


# In[33]:
```

```
data
```

```
## AdaBoostClassifier
```

```
# In[34]:
```

```
kfold2 = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)
```

```
# In[35]:
```

```
model = c(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
results2 = model_selection.cross_val_score(model, X, Y, cv=kfold2)
print(results)
print(results.mean())
print(results2.mean())
```

```
# In[36]:
```

```
#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
#adaboost = AdaBoostClassifier(n_estimators=100, learning_rate=1)

#Fit
model.fit(X_train,Y_train)

#Predict
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#Testare estimator pe 2 exemple concrete
my_test= [[3.6216,8.6661,-2.8073,-0.44699],[-2.5419,-0.65804,2.6842,1.1952]]
Y_prediction_test= model.predict(my_test)
print("Prediction form my given test:",Y_prediction_test)

# Testare 5 metrici pentru Clasificare
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
```

```
print("Accuracy:",metrics.accuracy_score(Y_test, Y_prediction))

confusion_mat = confusion_matrix(Y_test, Y_prediction)
print("Confusion matrix is :", confusion_mat)

recall_show = recall_score(Y_test, Y_prediction)
print("Recall_score is :",recall_show)

precision_show = precision_score(Y_test, Y_prediction)
print("Precision Score is :", precision_show)


f1_score = f1_score(Y_test, Y_prediction)
print("f1 Score is :",f1_score)


## Gradient Tree Boosting

# In[37]:


num_trees = 100
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())


# In[38]:


#Impartire dataset: train + test
from sklearn.model_selection import train_test_split
X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, tes
my_test= [[3.6216,8.6661,-2.8073,-0.44699],[-2.5419,-0.65804,2.6842,1.1952]]

# Train GradientBoostingClassifier
gboost = GradientBoostingClassifier(n_estimators=num_trees,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict
Y_prediction = gboost.predict(X_test)
# Testare model pe 2 exemple concrete
Y_prediction_test= gboost.predict(my_test)

print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

# Testare 5 metrici pentru Clasificare
```

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score


print("Accuracy:",metrics.accuracy_score(Y_test, Y_prediction))

confusion_mat = confusion_matrix(Y_test, Y_prediction)
print("Confusion matrix is :", confusion_mat)

recall_show = recall_score(Y_test, Y_prediction)
print("Recall_score is :",recall_show)

precision_show = precision_score(Y_test, Y_prediction)
print("Precision Score is :", precision_show)

f1_score = f1_score(Y_test, Y_prediction)
print("f1 Score is :",f1_score)


## Vizualizare rezultate cu parametri diferiti

# In[47]:


#Vizualizare rezultate obtinute cu parametri diferiti
range_k = range(1,151,10)
scoresA = {}
scores_listA = []
scoresG = {}
scores_listG = []

for k in range_k:
    ada = AdaBoostClassifier(n_estimators=k, random_state=seed)
    gboost = GradientBoostingClassifier(n_estimators=k,learning_rate=1)

    ada.fit(X_train, Y_train)
    gboost.fit(X_train, Y_train)
    Y_predictionA = ada.predict(X_test)
    Y_predictionG = gboost.predict(X_test)

    scoresA[k] = metrics.accuracy_score(Y_test,Y_predictionA)
    scores_listA.append(metrics.accuracy_score(Y_test,Y_predictionA))

    scoresG[k] = metrics.accuracy_score(Y_test,Y_predictionG)
    scores_listG.append(metrics.accuracy_score(Y_test,Y_predictionG))
```

```
# In[48]:


get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt

plt.subplot(1, 2, 1)
plt.plot(range_k,scores_listA)
plt.title("AdaBoost")
plt.xlabel("Number_of_estimators")
plt.ylabel("Accuracy")

plt.subplot(1, 2, 2)
plt.plot(range_k,scores_listG)
plt.title("GradientBoost")
plt.xlabel("Number_of_estimators")
plt.ylabel("Accuracy")


## Comparare rezultate obtinute pentru toti algoritmii prezentati

# Acuratete maxima

# In[57]:


import numpy as np
import matplotlib.pyplot as plt

accuracy = [0.9963, 1.0, 0.9975,0.9975, 1.0,1.0]
bars = ('DT', 'KNN', 'RandomForest', 'ExtraTrees', 'AdaBoost','GradientBoost'
x_pos = np.arange(len(bars))

plt.bar(x_pos, accuracy, color=['black', 'red', 'green', 'blue', 'cyan','yell

plt.xticks(x_pos, bars,rotation=30)
plt.ylabel("Accuracy")
plt.ylim((0.99,1))

plt.show()


# Precizie maxima

# In[58]:


import numpy as np
import matplotlib.pyplot as plt
```

```
accuracy = [0.9915, 1.0, 0.9947,1.0, 1.0,1.0]
bars = ('DT', 'KNN', 'RandomForest', 'ExtraTrees', 'AdaBoost','GradientBoost'
x_pos = np.arange(len(bars))

plt.bar(x_pos, accuracy, color=['black', 'red', 'green', 'blue', 'cyan','yell

plt.xticks(x_pos, bars,rotation=30)
plt.ylabel("Precision")
plt.ylim((0.99,1))

plt.show()


# F1 score

# In[59]:


import numpy as np
import matplotlib.pyplot as plt

accuracy = [0.9957, 1.0, 0.9973,0.9973, 1.0,1.0]
bars = ('DT', 'KNN', 'RandomForest', 'ExtraTrees', 'AdaBoost','GradientBoost'
x_pos = np.arange(len(bars))

plt.bar(x_pos, accuracy, color=['black', 'red', 'green', 'blue', 'cyan','yell

plt.xticks(x_pos, bars,rotation=30)
plt.ylabel("F1Score")
plt.ylim((0.99,1))

plt.show()
```

Bibliography

- https://machinelearningmastery.com/standard-machine-learning-datasets/
- https://scikit-learn.org/
- https://tutorialspoint.com/
- https://geeksforgeeks.org/