Mitocaru Irina

343C1

Analiza problemei

Contents

[Introducere 2](#_Toc35780498)

[Analiza setului de date 2](#_Toc35780499)

[1. Dimensiune 2](#_Toc35780500)

[2. Clase 2](#_Toc35780501)

[3. Atribute 2](#_Toc35780502)

[Analiza clasificatorilor 3](#_Toc35780503)

[K-means 3](#_Toc35780504)

[1. Baseline 3](#_Toc35780505)

[2. Imbunatatiri 3](#_Toc35780506)

[3. Grafice intra-algoritmi 3](#_Toc35780507)

[Random Forest Classifier 4](#_Toc35780508)

[1. Baseline 4](#_Toc35780509)

[2. Imbunatatiri 4](#_Toc35780510)

[3. Grafice intra-algoritmi 6](#_Toc35780511)

[XGBoost Classifier 6](#_Toc35780512)

[1. Baseline 6](#_Toc35780513)

[2. Imbunatatiri 6](#_Toc35780514)

[3. Grafice intra-algoritmi 8](#_Toc35780515)

[SVM (SVC) Classifier 8](#_Toc35780516)

[1. Baseline 8](#_Toc35780517)

[2. Imbunatatiri 8](#_Toc35780518)

[3. Grafice intra-algoritmi 9](#_Toc35780519)

[Grafice inter-algoritmi 10](#_Toc35780520)

[1. Acuratete 10](#_Toc35780521)

[2. Precizie 10](#_Toc35780522)

[3. Recall 11](#_Toc35780523)

[4. F1 11](#_Toc35780524)

[5. Matricea de confuziune 12](#_Toc35780525)

[Concluzie 13](#_Toc35780526)

# Introducere

Acest document ilustreaza atat analiza amanuntita a setului de date cu care se lucreaza, cat si analiza 4 modele folosite la clasificarea unui set de date dupa genul muzical.

# Analiza setului de date

### Dimensiune

Setul de date e impartit in 2000 de exemple pentru setul de invatare si 400 de exemple pentru setul de testare si validare. In total sunt 2400 de exemple.

### Clase

Facand o analiza a distributiei claselor, s-a observat ca toate 4 (Hip-Hop, Electronic, Folk si Rock) sunt impartite in mod egal, fiecare avand 600 de exemple, lucru util deoarece se evita fenomenul de overfitting.

### Atribute

#### Echonest.csv

* audio\_features: 8 atribute, valori numerice si fara NaN
* social\_features: 5 atribute, valori numerice si fara NaN
* ranks: 5 atribute, valori numerice si NaN
* metadata: 7 atribute, valori numerice, categorice si NaN
* temporal\_features: 224 atribute, valori numerice si fara NaN

#### Features.csv

* chroma\_cens
* chroma\_cqt
* chroma\_stft
* mfcc
* rmse
* spectral\_bandwidth
* spectral\_centroid
* spectral\_contrast
* spectral\_rolloff
* tonnetz
* zcr

Toate sunt in combinatie cu: 'kurtosis', 'max', 'mean', 'min', 'skew' si 'std', avand atat valori numerice, categorice, cat si NaN.

# Analiza clasificatorilor

## K-means

### Baseline

KMeans(n\_clusters=n\_clusters)

### Imbunatatiri

#### Hiperparametri

KMeans(n\_clusters=n\_clusters)

**n\_clusters:**

S-a ales numarul optim de clustere folosind silhouette.

**init:**

* ‘k-means++’: nu a adus imbunatatiri acuratetei
* ‘random’: nu a adus imbunatatiri acuratetei

S-a ales default: ‘k-means++’

**tol:**

S-au incercat mai multe valori fara a aduce imbunatiri. S-a ales default.

**precompute\_distances:**

**S-a incercat ‘True’, fara imbunatatiri. S-a ales default.**

**algorithm:**

* **‘full’: fara imbunatatiri**
* **‘elkan’: fara imbunatatiri**

**S-a ales default: ‘auto’.**

#### Preprocesare

preprocessing.StandardScaler().fit\_transform(x\_train)

S-a aplicat standardizarea.

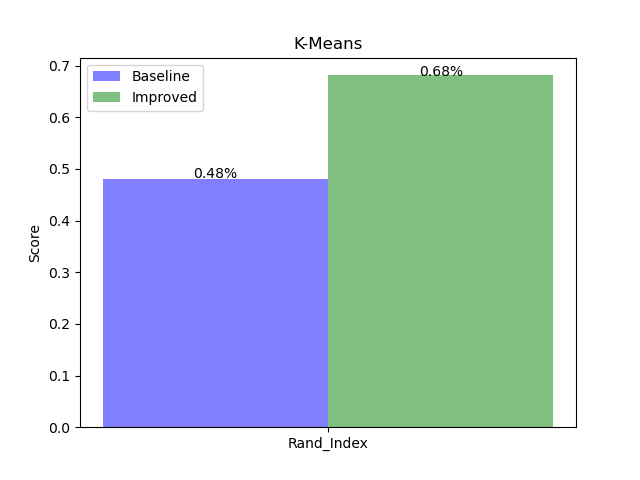
#### Atribute

S-a incercat combinatia dintre audio\_features cu fiecare din celelalte din categoria Echonest si cateva din Features. Cea mai optima solutie a fost folosirea exclusiva de ‘audio-features’.

‘audio\_features’

### Grafice intra-algoritmi

* Rand Index: 0.65% baseline, 0.71% varianta imbunatatita



## Random Forest Classifier

### Baseline

RandomForestClassifier(random\_state=0)

### Imbunatatiri

#### Hiperparametri

RandomForestClassifier(n\_jobs=-1, n\_estimators=200, max\_depth=depth, random\_state=0)

S-au incercat urmatorii hiper-parametri:

**n\_estimators**:

* (100, 500): acuratete cu 3% mai mare
* >= 500: acuratete la metrica f1\_score cu 2% mai mare, iar la restul cu 3% mai mare

Astfel, s-a ales valoarea 200.

**max\_depth**:

* None: acurarete la fel ca la baseline
* > ¾ din numarul de atribute: acuratete cu 1% mai mare
* < ¾ din numarul de atribute: acuratete mai slaba ca la baseline
* = ¾ din numarul de atribute: acuratete cu 4% mai mare

S-a observat ca o combinatie de numar de estimatori mic (100) cu o limita de adancime mica (1/2 din numarul de atribute) rezulta in performante proaste, deoarece algoritmul nu ia in considerare toate atributele (sau mai multe atribute) care aduc un plus acuratetei.

S-a ales max\_depth = 3/4 din numarul de atribute.

**criterion**:

* ‘gini’: acurarete la fel ca la baseline
* ‘entropy’: acuratete mai slaba ca la baseline

S-a ales ‘gini’.

**n\_jobs**:

S-a ales -1 pentru a folosi tot CPU-ul la maxim, daca este nevoie.

**min\_samples\_split & min\_samples\_leaf:**

* >2: nu avem imbunatatiri

S-a ales valoarea default: 1.

**max\_features:**

* int: nu avem imbunatatiri
* alte optiuni mai putin default: performante mai proaste

S-a ales default: auto.

**max\_leaf\_nodes & min\_impurity\_decrease & min\_impurity\_split & bootstrap & cc\_alpha:**

* alte optiuni mai putin default: performante mai proaste

S-a ales default: None.

**random\_state:**

S-a ales valoarea 0 pentru a face modelul determinist.

#### Preprocesare

depth = int(len(x\_train.columns) \* 3 / 4)

x\_train = SelectKBest(f\_classif, k=depth).fit\_transform(x\_train, y\_train)

Pentru preprocesarea setului de invatare, s-a incercat:

* Standardizare: fara rezultat
* Normalizare: fara rezultat
* SelectKBest: singura metoda care a adus un plus de 2-3% la perfomanta. Si aici s-a ales un numar de atribute = ¾ din numarul total

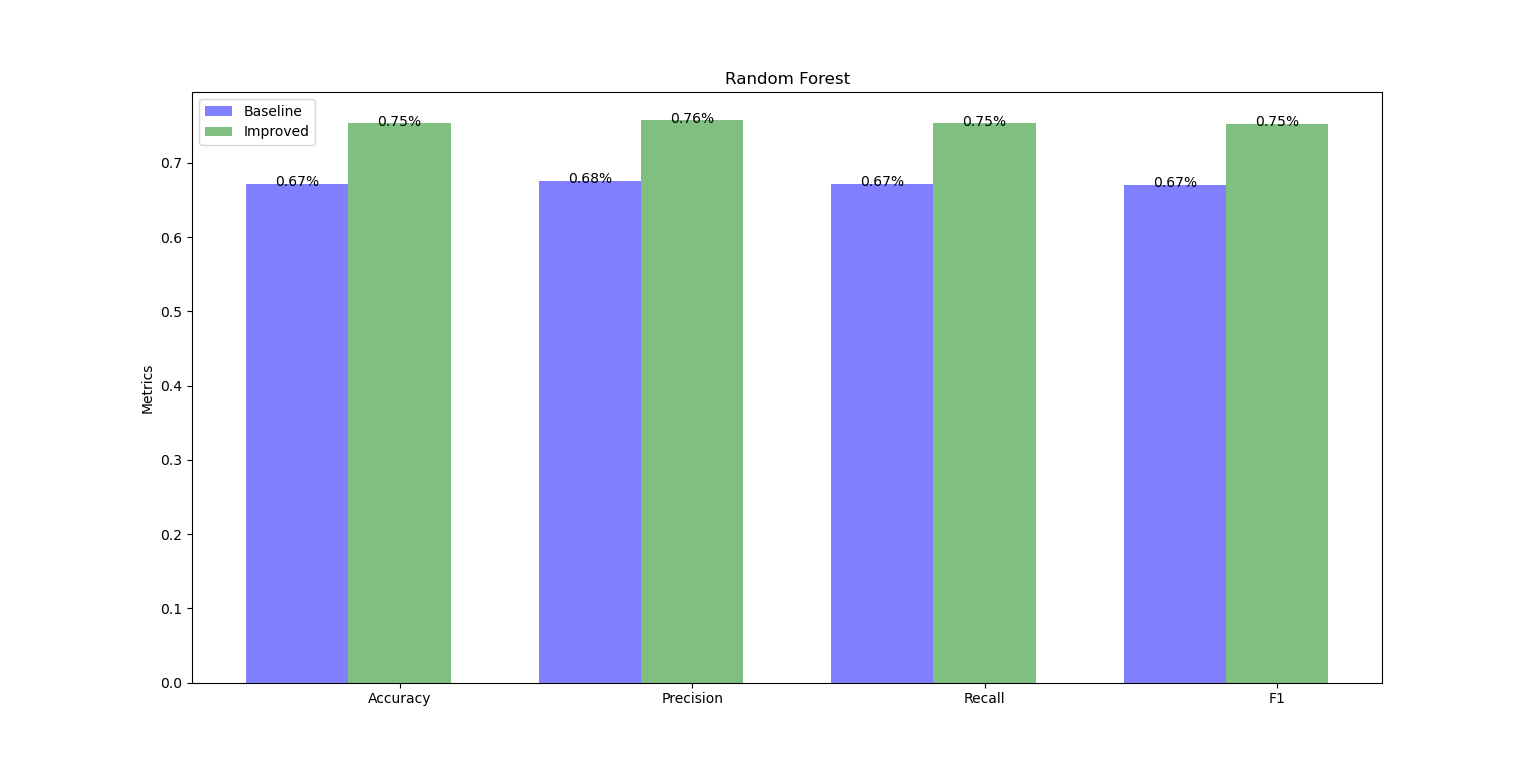
#### Atribute

S-a incercat combinatia dintre audio\_features cu fiecare din celelalte din categoria Echonest si cateva din Features. Singura combinatie care a adus un plus performantei a fost:

‘audio\_features’ cu ‘temporal\_features’

### Grafice intra-algoritmi

* Acuratete: 0.67% baseline, 0.75% varianta imbunatatita
* Precizie: 0.68% baseline, 0.76% varianta imbunatatita
* Recall: 0.67% baseline, 0.75% varianta imbunatatita
* F1: 0.67% baseline, 0.75% varianta imbunatatita



## XGBoost Classifier

### Baseline

XGBClassifier(random\_state=0)

### Imbunatatiri

#### Hiperparametri

XGBClassifier(n\_jobs=-1, random\_state=0, subsample=0.9, objective='multi:softmax', num\_class=4, learning\_rate=0.1)

S-au incercat urmatorii hiper-parametri:

**learning\_rate**:

* = 0.1: performanta imbunatatita cu 7%
* > 0.1: performanta imbunatatita cu 5%
* < 0.1: performanta intre baseline si baseline + 5%

S-a ales 0.1.

**max\_depth:**

* None: acurarete la fel ca la baseline
* > ¾ din numarul de atribute: acurarete la fel ca la baseline
* < ¾ din numarul de atribute: acuratete mai slaba ca la baseline
* = ¾ din numarul de atribute: acurarete la fel ca la baseline

S-a ales default: None.

**booster:**

* ’gbtree’: acurarete la fel ca la baseline
* ’gblinear’: acuratete mai slaba ca la baseline
* ’dart’: acurarete la fel ca la baseline

S-a ales default: None

**num\_parallel\_tree:**

S-au incercat mai multe valori, toate crescand complexitatea temporala si neaducand un plus de performanta. S-a ales default: None

**n\_estimators:**

* >= 100: acurarete la fel ca la baseline

S-a ales default: 100

**importance\_type:**

S-au incercat mai multe valori, fara rezultat. S-a ales default: None

**subsampling:**

* 1: acurarete la fel ca la baseline
* < 0.9: acuratete cu 1-2% mai mare
* = 0.9: acuratete cu 2% mai mare

S-a ales 0.9

**objective:**

S-a ales valoarea 'multi:softmax' deoarece modelul foloseste by default binary:logistic, care se foloseste la clasificator binar.

**num\_class:**

S-a ales valoarea 4 deoarece acesta este numarul de clase a setului nostru de date. Hiper-parametrul se foloseste in combinatie cu ’objective’.

**random\_state:**

S-a ales valoarea 0 pentru a face modelul determinist.

#### Preprocesare

Pentru preprocesarea setului de invatare, s-a incercat:

* Standardizare: fara rezultat
* Normalizare: fara rezultat
* SelectKBest: fara rezultat

Astfel, nu s-a efectuat nicio preprocesare a setului de invatare pentru acest model.

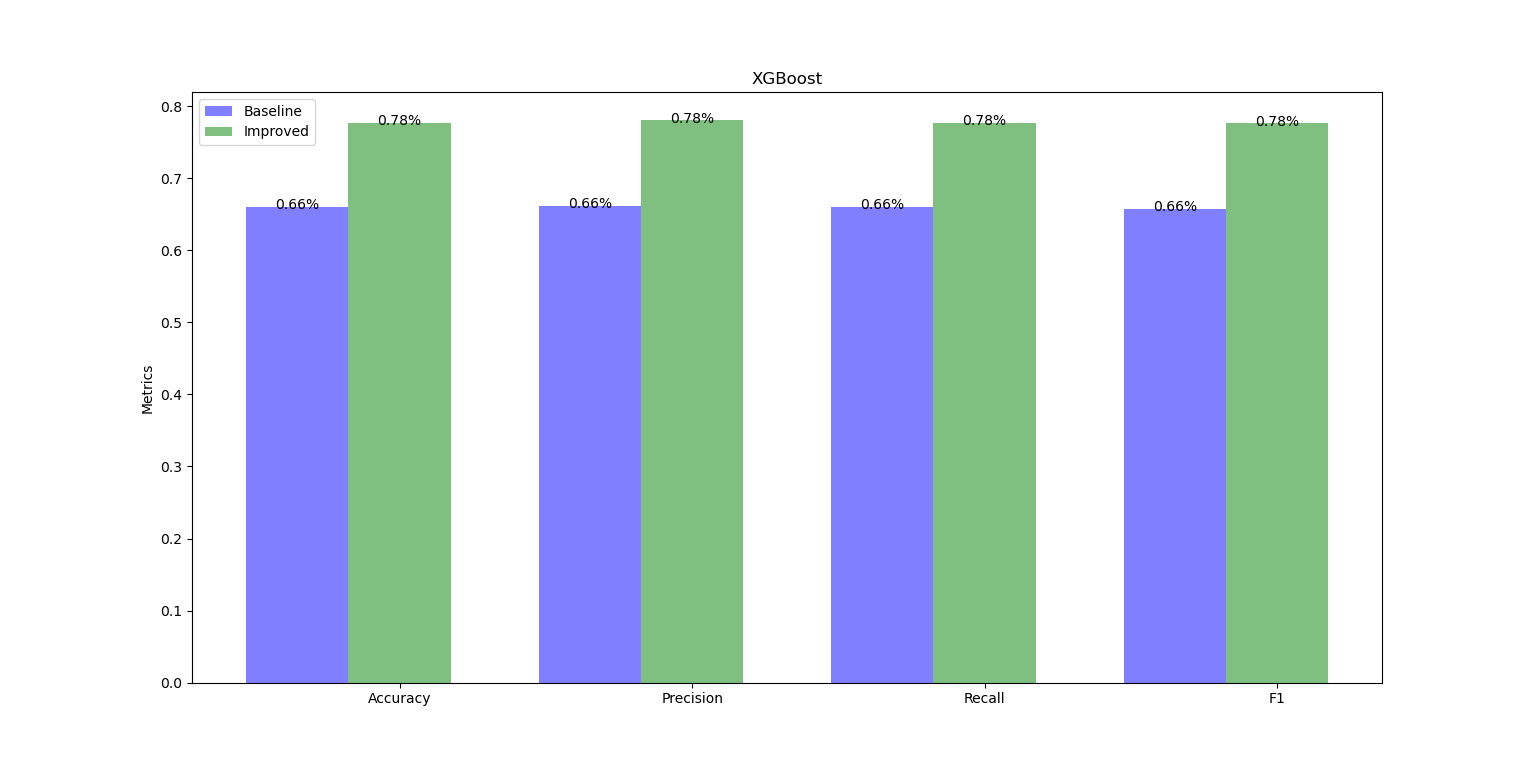
#### Atribute

S-a incercat combinatia dintre audio\_features cu fiecare din celelalte din categoria Echonest si cateva din Features. Singura combinatie care a adus un plus performantei a fost:

‘audio\_features’ cu ‘temporal\_features’

### Grafice intra-algoritmi

* Acuratete: 0.66% baseline, 0.78% varianta imbunatatita
* Precizie: 0.66% baseline, 0.78% varianta imbunatatita
* Recall: 0.66% baseline, 0.78% varianta imbunatatita
* F1: 0.66% baseline, 0.78% varianta imbunatatita



## SVM (SVC) Classifier

### Baseline

SVC(random\_state=0)

### Imbunatatiri

#### Hiperparametri

SVC(random\_state=0, decision\_function\_shape='ovo', kernel='rbf')

**kernel:**

* ‘rbf’: acuratete imbunatatita
* ‘poly’: acuratete imbunatatita mai putin
* ‘linear’: dureaza foarte mult procesul de invatare
* ‘sigmoid’: acuratate mai proasta decat cea din varianta ‘poly’

S-a ales default: ’rbf’.

**gamma & C:**

S-au incercat diferite valori, toate aducand performante foarte scazute. S-a ales default.

**random\_state:**

S-a ales valoarea 0 pentru a face modelul determinist.

**decision\_fucion\_shape:**

S-a ales valoarea ‘ovo’ (one vs one) pentru a imbunatati perfomanta clasificatorului multiclass.

#### Preprocesare

preprocessing.StandardScaler().fit\_transform(x\_train, y\_train)

Pentru preprocesarea setului de invatare, s-a incercat:

* Standardizare: performanta s-a imbunatati cu 20%
* Normalizare: fara rezultat
* SelectKBest: fara rezultat

Astfel, s-a ales standardizarea.

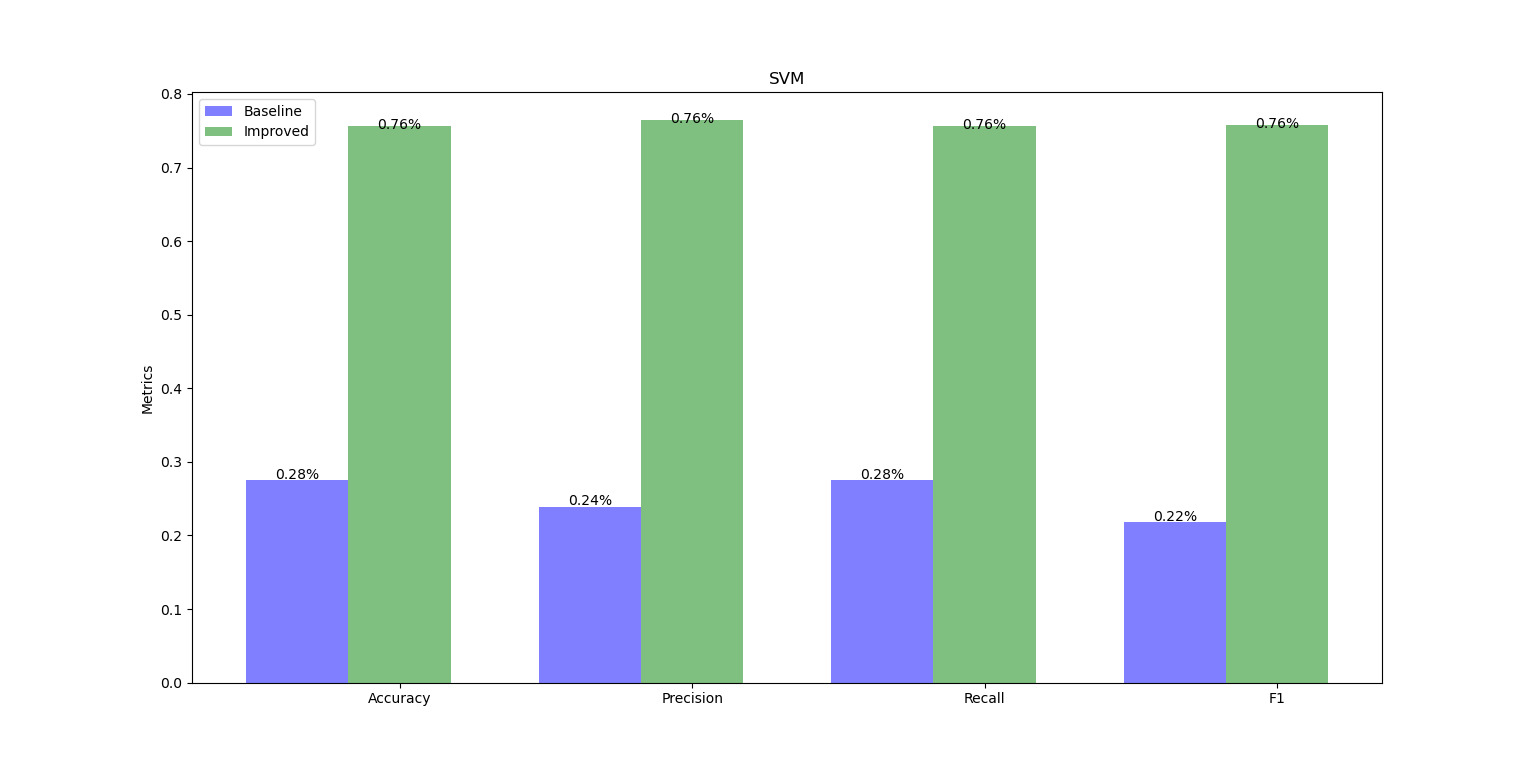
#### Atribute

S-a incercat combinatia dintre audio\_features cu fiecare din celelalte din categoria Echonest si cateva din Features. Singura combinatie care a adus un plus performantei a fost:

‘audio\_features’ cu ‘temporal\_features’

### Grafice intra-algoritmi

1. Acuratete: 0.28% baseline, 0.76% varianta imbunatatita
2. Precizie: 0.24% baseline, 0.76% varianta imbunatatita
3. Recall: 0.28% baseline, 0.76% varianta imbunatatita
4. F1: 0.22% baseline, 0.76% varianta imbunatatita

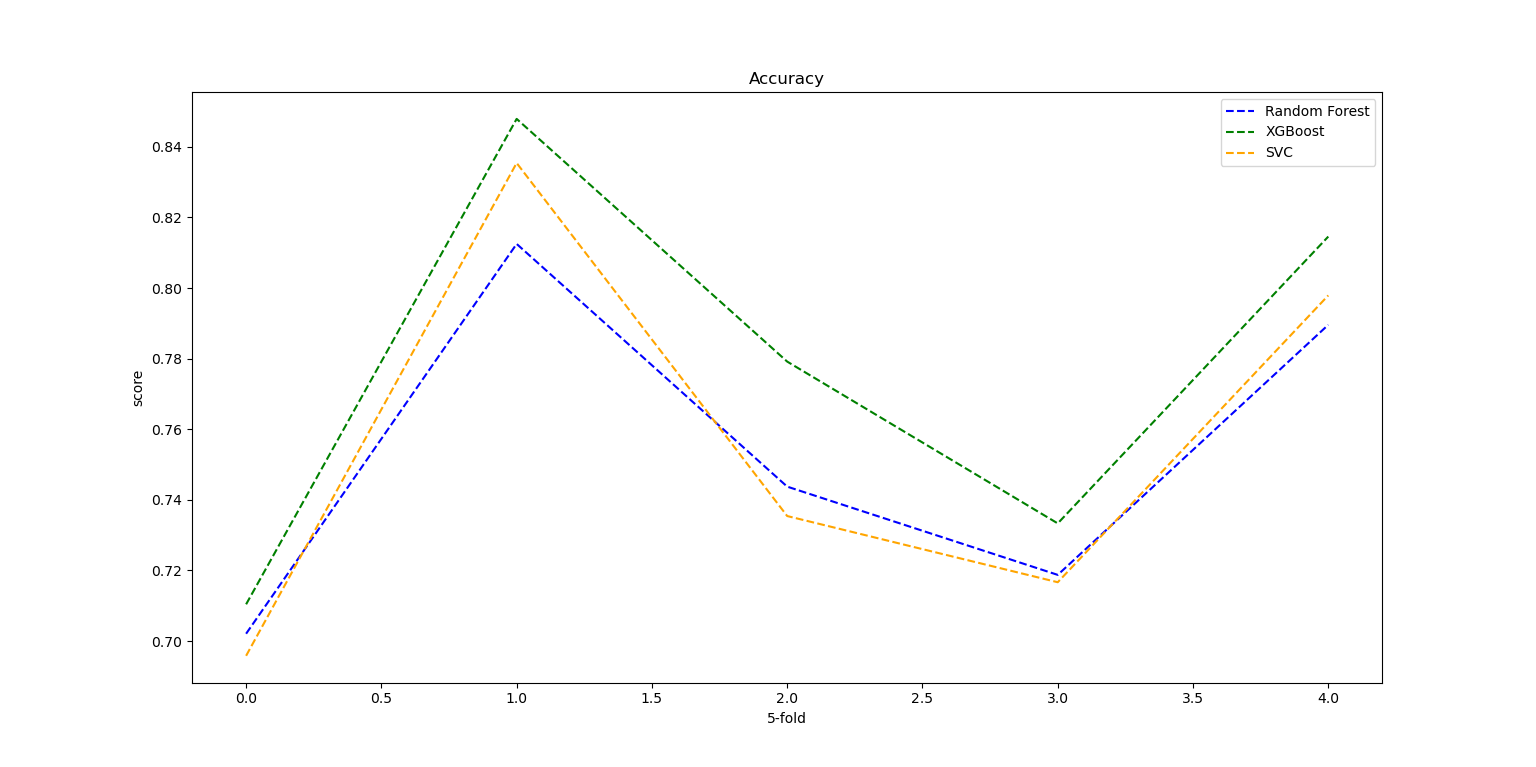


# Grafice inter-algoritmi

Pentru evaluare, s-a folosit 5-Fold Cross Validation si functia cross\_validate cu metricile de mai jos pe fiecare model in parte. Asupra vectorilor rezultati s-a aplicat average pentru a calcula procentul.

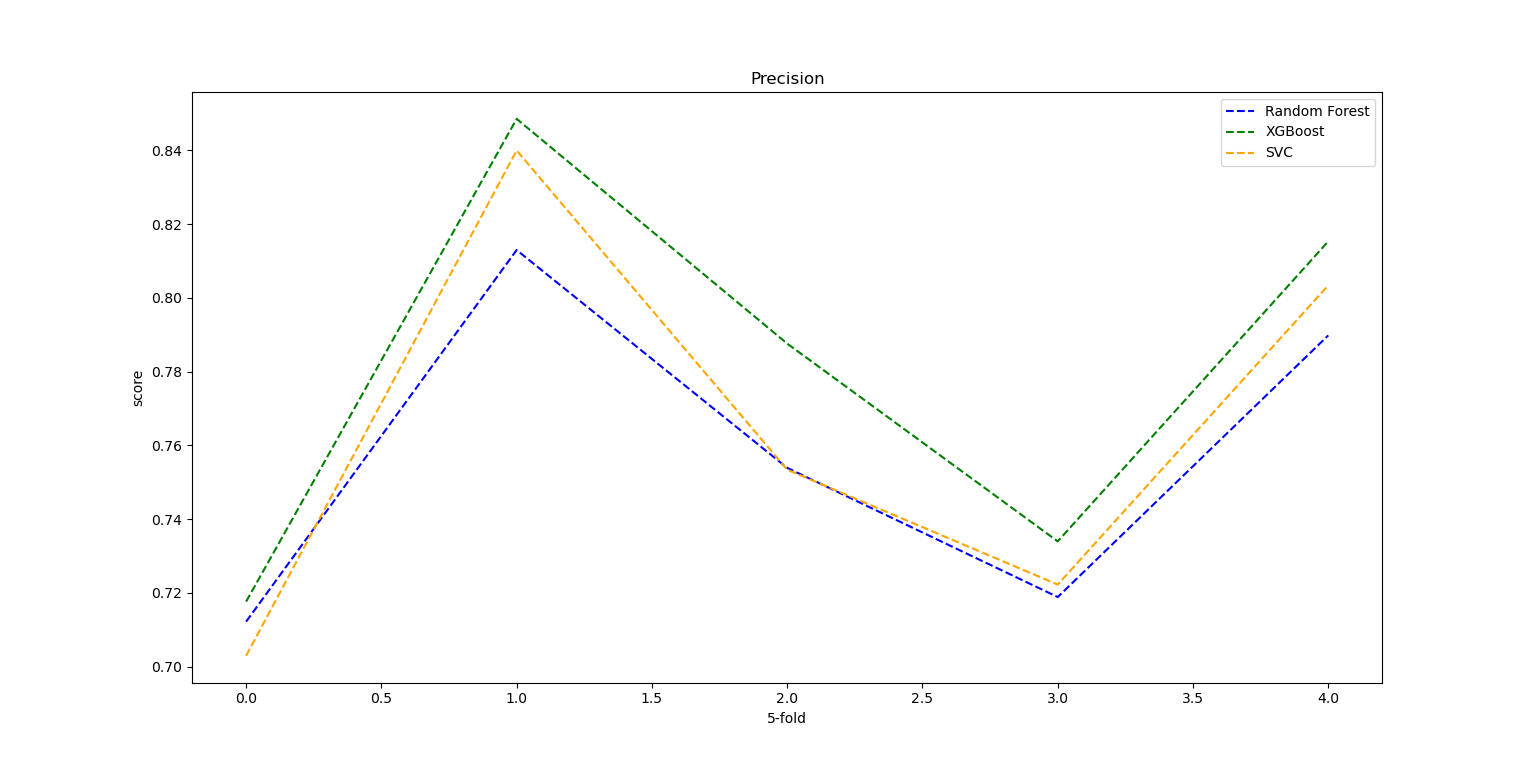
### Acuratete

* Random Forest: 0.75%
* XGBoost: 0.78%
* SVM: 0.76%



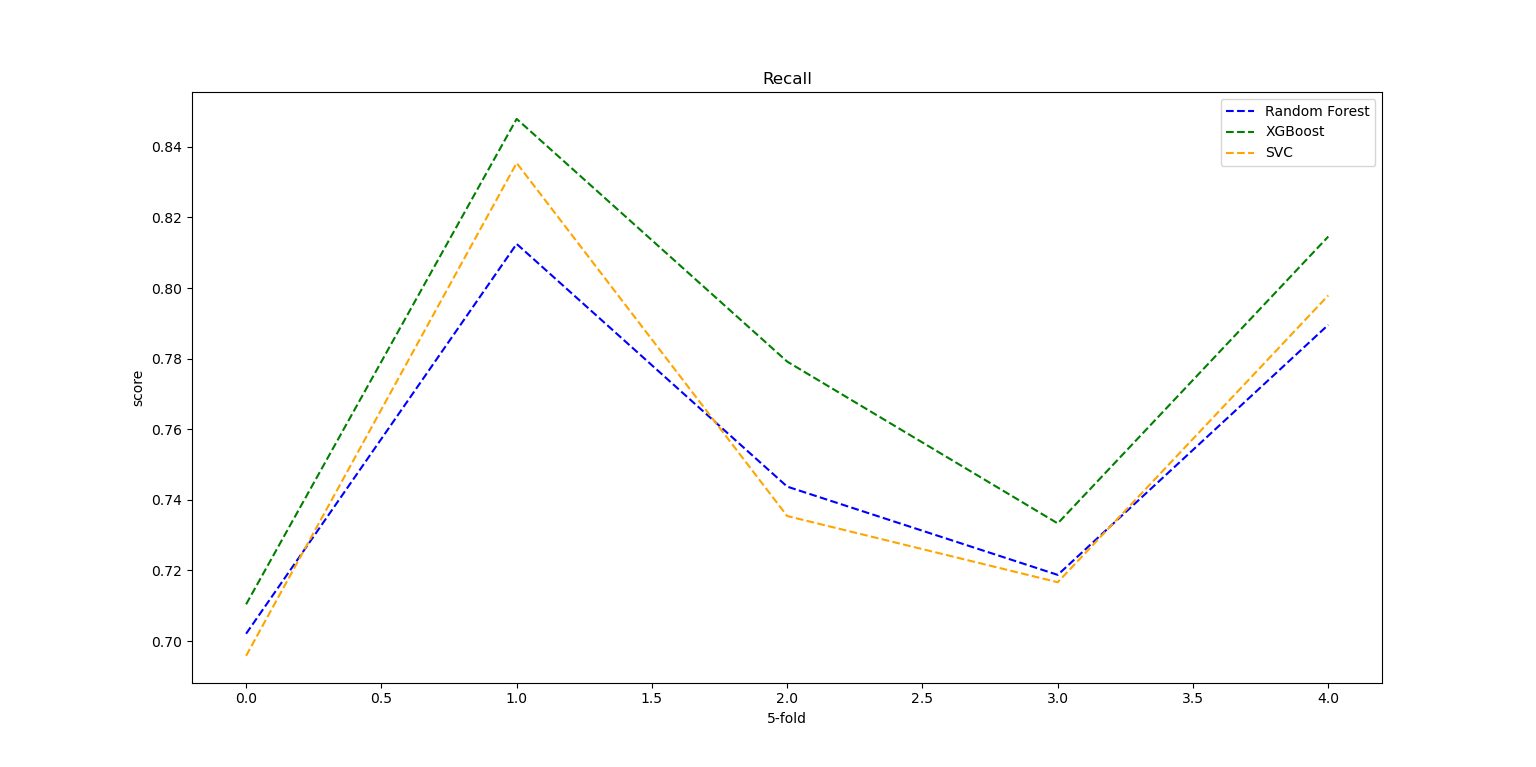
### Precizie

* Random Forest: 0.76%
* XGBoost: 0.78%
* SVM: 0.76%



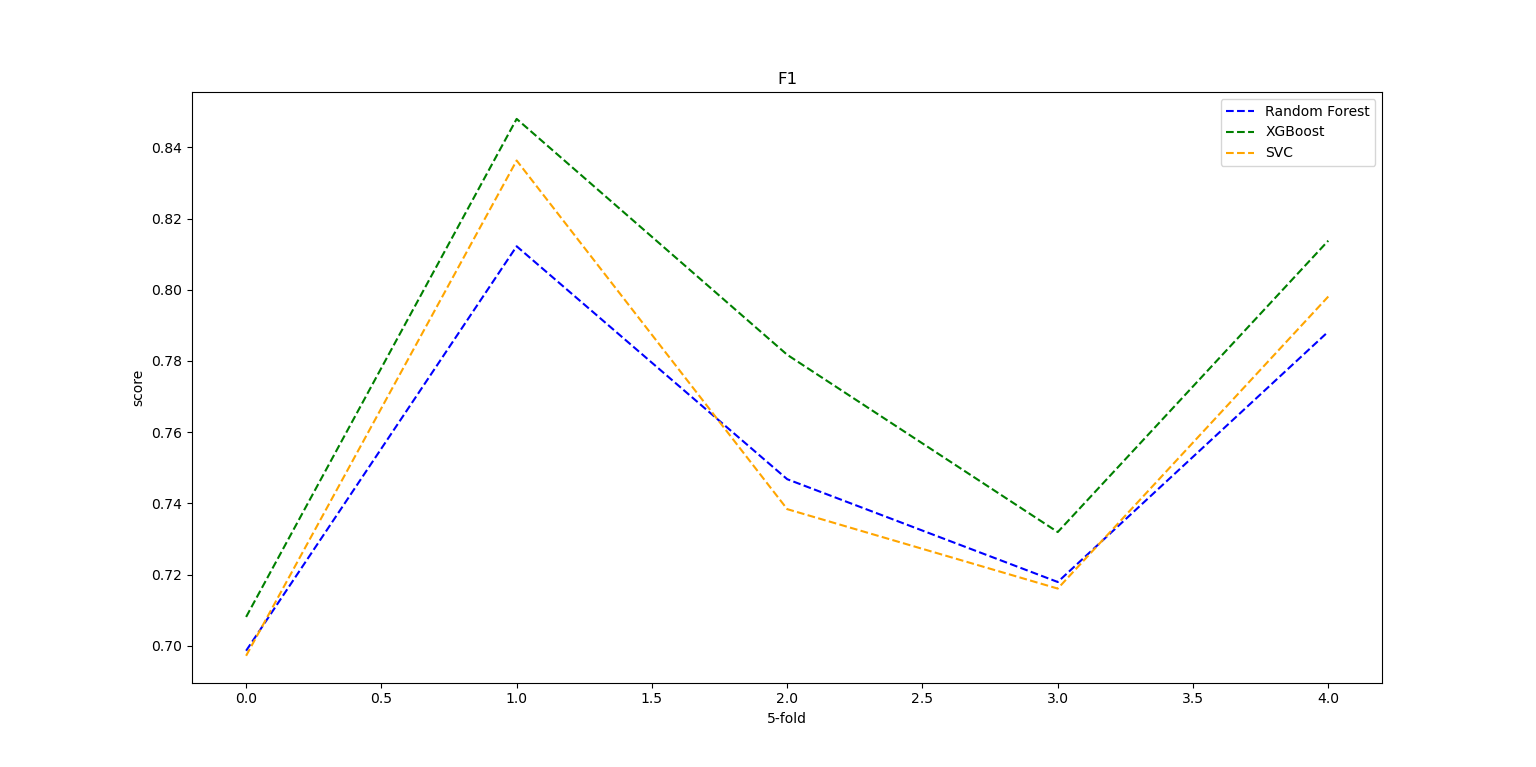
### Recall

* Random Forest: 0.75%
* XGBoost: 0.78%
* SVM: 0.76%

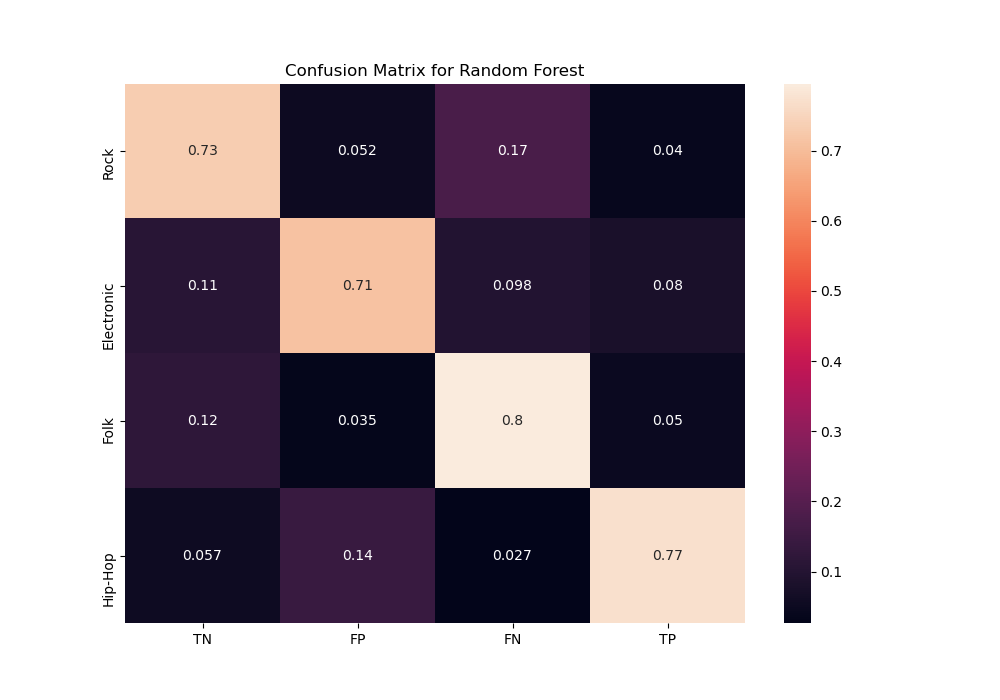


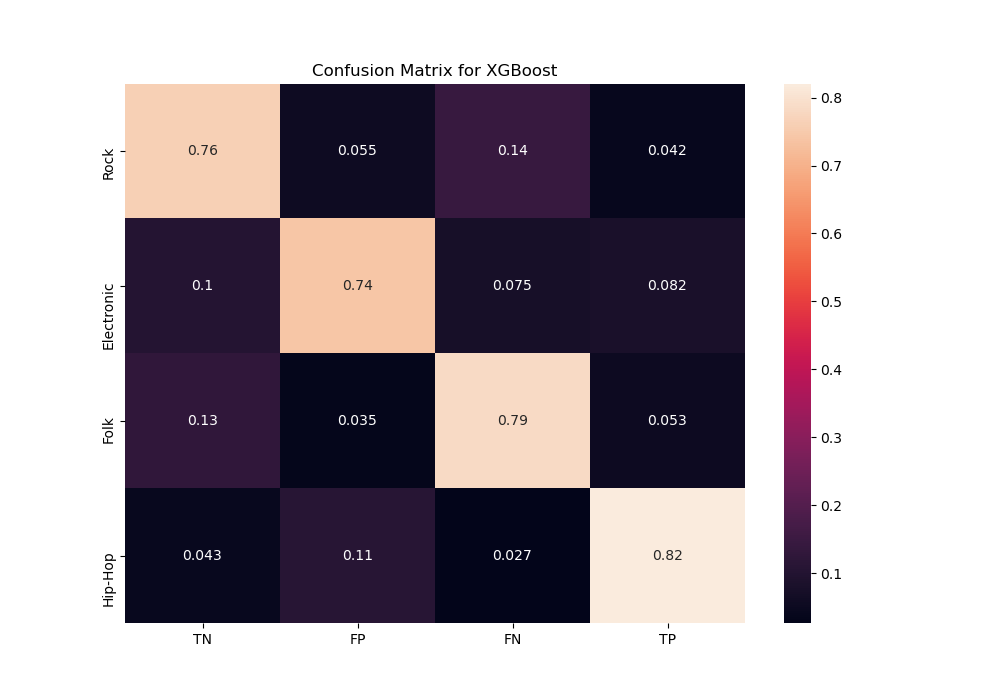
### F1

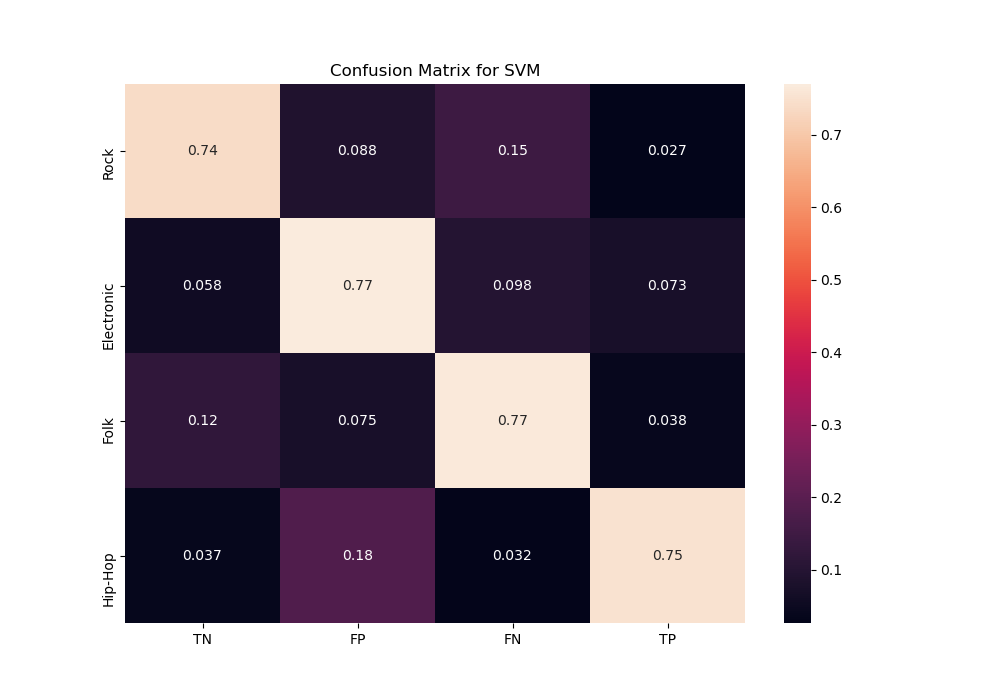
* Random Forest: 0.75%
* XGBoost: 0.78%
* SVM: 0.76%



### Matricea de confuziune







# Concluzie

Cel mai bun model este XGBoost, cu o acuratete de 78%.

Dezavantajul la acest model este faptul ca dureaza mult deoarece se bazeaza pe principiul de Boosting si faptul ca foloseste CPU-ul la maxim. Este, de altfel, si modelul care dureaza cel mai mult.

Pe locul doi ar fi SVM, cu o acuratete de 76%, apoi Random Forest cu aproximativ aceeasi acuratete (57%), iar K-Means nu poate fi comparat din punct de vedere al metricilor cu ceilalti algoritmi deoarece el doar clusterizeaza / grupeaza, singura metrica folosita pentru a prezice corectitudinea fiind Rand Index.