Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach
Tools: Python 2.7.3, Python 3, Binder notebook si Excel

Name: Salamon Adrian
Group: 30241
Email: salamon.adrianpaul@gmail.com

Contents

Chapter 1

Obiectivul lucrarii

Obiectivul lucrarii este utilizarea algoritmului KNN fara a specifica parametrul k in mod empiric. Metoda propusa in acest articol a fost asamblarea clasificatoarelor KNN cu k=1, 3, 5, 7 ... n (unde n reprezinta radacina patrata a dimensiunii setului de date) intr-un singur clasificator care va clasifica in urma deciziei majoritare.
Rezultatele experimentale arată că clasificatorul propus depășește clasificatorul tradițional KNN care folosește un număr diferit de vecini, este competitiv cu alți clasificatori și este un clasificator promițător, cu potențial puternic pentru o gamă largă de aplicații.

Chapter 2

Probleme intampinate

2.1 Asamblarea in mod dinamic a unui numar variabil de clasificatori

Dezvoltarea clasificatorului asamblat KNN a reprezentat cea mai dificila parte a acestei lucrari. Cea mai mare parte a timpului de dezvoltare a constat in descoperirea unei metode de a genera dinamic clasificatorii 1-NN, 3-NN, 5-NN ... $\sqrt{n}$-NN.
Solutia propusa de mine este urmatoarea:

```
# get a voting ensemble of models
def get_voting(n):
        k=-1; count=0; models = list(); label="-NN"; labelList=[];
        while k<n:
                k=k+2;
                count=count+1;
                labelList.append(str(k)+label)
                # define the base models
                models.append((str(k)+label, KNeighborsClassifier(n_neighbors=k)))
        # define the voting ensemble
        ensemble = VotingClassifier(estimators=models, voting='hard')
        return ensemble
```

Figure 2.1: Implementarea clasificatorului asamblat

Parametrul n contine valoarea ultimului k din clasificatorul asamblat ($\sqrt{n}$-NN).
La prima iteratie din interiorul instructiunii while vom instantia primul clasificator din cadrul ansamblului (1-NN) si il vom adauga la lista de modele. La urmatoarea iteratie vom adauga clasificatorul 3-NN la lista de modele , apoi 5-NN pana cand k va deveni $\sqrt{n}$, moment in care vom avea in lista de modele toti clasificatorii (1-NN, 3-NN, 5-NN ... $\sqrt{n}$- NN) si vom iesi din instructiunea while.
Clasificatorii au fost instantiati de libraria sklearn si corespund cu specificatiile algoritmului prezentat in articol.
Mai ramane doar sa instantiem clasificatorul KNN asamblat prin intermediul

metodei VotingClassifier definita in cadrul librariei sklearn si sa returnam acest clasificator ca rezultat.

2.2 Descoperirea si formatarea seturilor de date folosite in lucrare

2.2.1 Descoperirea seturilor de date

In articol este doar mentionat linkul: `http://archive.ics.uci.edu/ml` ca fiind sursa seturilor de date, insa dupa o cautare indelungata am descoperit doar fisierele prezentate in sectiunile urmatoare pe acest site.

2.2.2 Convertirea seturilor de date

Majoritatea seturilor de date erau de tip .txt .dat si .data. Pentru a putea refolosi codul python implementat am decis sa convertesc toate seturile de date in format .csv.
Durata acestui proces a fost marita de faza de verificare a fisierelor in urma conversiei.

2.3 Bug de ordonare a clasificatorilor la afisare

Am observat in urma verificarii primului set de date ca la afisare clasificatorii apar intr-o ordine aleatoare in urma unui bug al librariei sklearn. Clasificatorul avea in dreptul sau acuratetea corespunzatoare, insa perechile apareau intr-o ordine neasteptata.

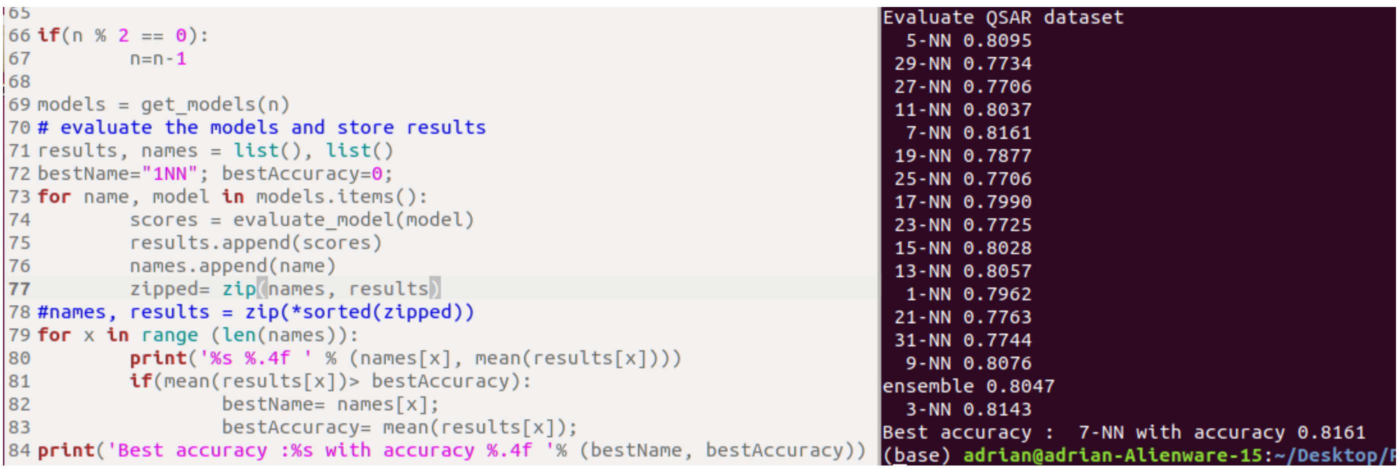


Figure 2.2: Ordine aleatoare a rezultatelor in urma rularii codului python 2.7.3

Pentru a rezolva aceasta problema am decis sa sortez clasificatorii la afisare pastrand legatura dintre numele clasificatorului si performanta acestuia. Am realizat acest lucru legat lista de nume cu lista de scoruri prin intermediul instructiunii zip, astfel in urma ordonarii listei numelor si lista acuratetilor va fi sortata in asa fel incat clasificatorii sa fie legati de scorurile lor prin acelasi index.

A aparut insa problema ordonarii alfabetice, adica, din punct de vedere alfabetic 11-NN este inaintea lui 3-NN. Pentru a rezolva aceasta problema am adaugat doua spatii in fata clasificatoriilor cu o cifra(1-NN, 3-NN, 5-NN, 7-NN, 9-NN) si un spatiu in fata clasificatorilor cu doua cifre(11-NN .... 99-NN) pentru a fi ordonati in ordinea asteptata.

```python
# get a list of models to evaluate
def get_models(n):
        models = dict()
        k=-1; count=0; label="-NN"; labelList=[];
        while k<n:
                k=k+2;
                count=count+1;
                labelList.append(str(k)+label)
                # define the base models
                if(k<10):
                        models['  '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
                elif(k>10 and k<100):
                        models[' '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
                else:
                        models[str(k)+label] = KNeighborsClassifier(n_neighbors=k)

        models['ensemble'] = get_voting(n)
        return models
```

Figure 2.3: Metoda de creare a clasificatorilor

In urma modificarilor de mai sus afisarea functioneaza normal

```python

if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

```
Evaluate QSAR dataset
  1-NN 0.7962
  3-NN 0.8143
  5-NN 0.8095
  7-NN 0.8161
  9-NN 0.8076
 11-NN 0.8037
 13-NN 0.8057
 15-NN 0.8028
 17-NN 0.7990
 19-NN 0.7877
 21-NN 0.7763
 23-NN 0.7725
 25-NN 0.7706
 27-NN 0.7706
 29-NN 0.7734
 31-NN 0.7744
ensemble 0.8047
Best accuracy :   7-NN with accuracy 0.8161
(base) adrian@adrian-Alienware-15:~/Desktop/
```

Figure 2.4: Ordinea corecta a clasificatorilor

Dovada a nealterarii rezultatelor:

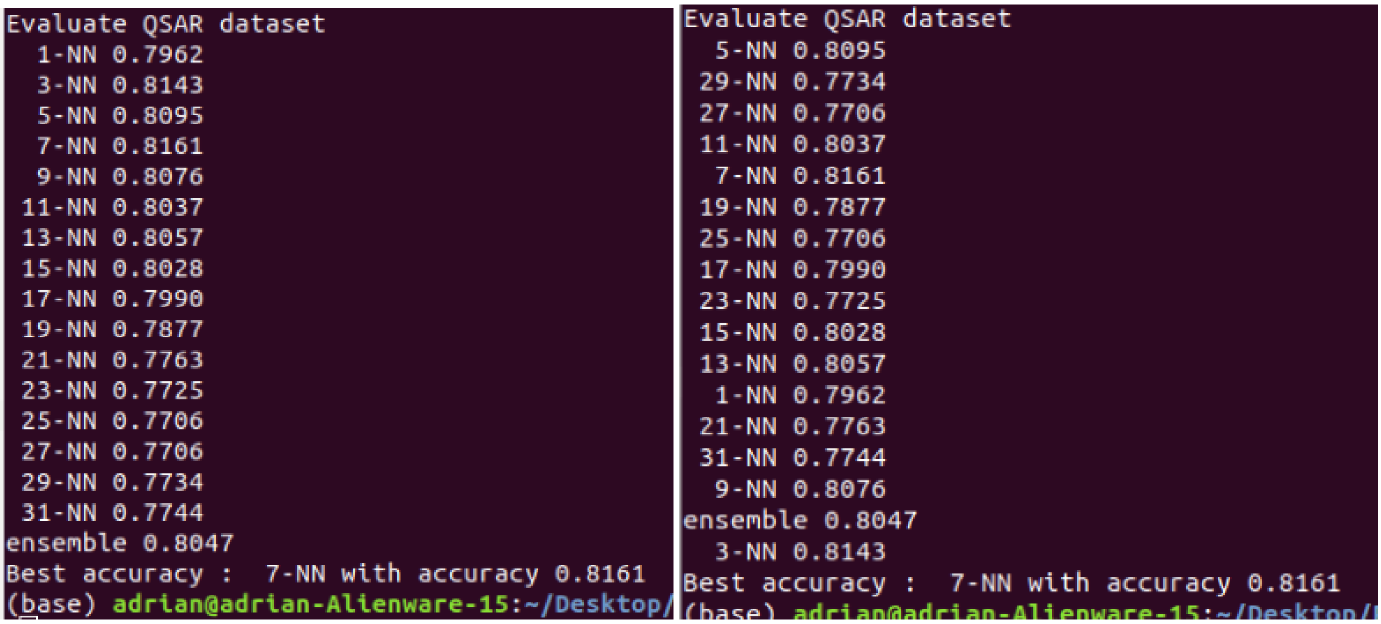


Figure 2.5: Stanga: Ordinea corecta a clasificatorilor
Dreapta: Ordinea aleatoare a clasificatorilor

Chapter 3

KNN ensemble classifier

3.1 Componente

Generarea clasificatorului asamblat

```
# get a voting ensemble of models
def get_voting(n):
	k=-1; count=0; models = list(); label="-NN"; labelList=[];
	while k<n:
		k=k+2;
		count=count+1;
		labelList.append(str(k)+label)
		# define the base models
		models.append((str(k)+label, KNeighborsClassifier(n_neighbors=k)))
	# define the voting ensemble
	ensemble = VotingClassifier(estimators=models, voting='hard')
	return ensemble
```

Figure 3.1: Aceasta metoda returneaza clasificatorul asamblat KNN Ensemble

Generarea unei liste care contine clasificatorii 1-NN, 3-NN ... $\sqrt{n}$- NN si clasificatorul KNN Esemble

```
# get a list of models to evaluate
def get_models(n):
	models = dict()
	k=-1; count=0; label="-NN"; labelList=[];
	while k<n:
		k=k+2;
		count=count+1;
		labelList.append(str(k)+label)
		# define the base models
		if(k<10):
			models['  '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
		elif(k>10 and k<100):
			models[' '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
		else:
			models[str(k)+label] = KNeighborsClassifier(n_neighbors=k)

	models['ensemble'] = get_voting(n)
	return models
```

Figure 3.2: Aceasta metoda returneaza o lista a clasificatorilor

Evaluarea acuratetii

```
# evaluate a give model using cross-validation
def evaluate_model(model):
        cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=1)
        scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
        return scores
```

Figure 3.3: Aceasta metoda care va evalua fiecare model individual, metrica de interes fiind acuratetea. Pentru testare am impartit setul de date in 70% date de antrenare si 30% date de testare cum a specificat autorul documentuluir

Citirea datelor si instantierea listei modelelor(Exemplu QSAR)

```
input_file = "QSAR .csv"

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

X, y = data[data.columns.drop('F43')], data['F43']

n=int(math.sqrt(1055))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
```

Figure 3.4: Pentru fiecare set de date valoarea input_file va corespunde cu numele setului de date
etu QSAR contine 1055 de randuri de date si 43 de feature-uri, feature-ul 43 fiind cel pe care dorim sa-l clasificam.
se calculeaza $\sqrt{dimeniuneasetuluidedate}$ si se scade 1 daca acesta este par, apoi se instantiaza lista de modele care va contine clasificatorii 1-NN, 3-NN... $\sqrt{dimeniuneasetuluidedate}$-NN, KNN Ensemble(care contine fiecare dintre clasificatorii mentionati anterior)

Afisarea rezultatelor

```
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.5: Afisarea rezultatelor

3.2 Australian data set

Australian data set contine 690 randuri de date, 15 de feature-uri, feature-ul pe care il vom clasifica este F15 care are 2 posibile clase

```
print('Evaluate Australian dataset')
input_file = "australian.csv"

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

X, y = data[data.columns.drop('F15')], data['F15']

n=int(math.sqrt(690))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.6: Australian data set

Rezultate

```
Evaluate Australian dataset
  1-NN 0.6347
  3-NN 0.6668
  5-NN 0.6812
  7-NN 0.6827
  9-NN 0.6928
 11-NN 0.6929
 13-NN 0.6682
 15-NN 0.6609
 17-NN 0.6653
 19-NN 0.6813
 21-NN 0.6755
 23-NN 0.6828
 25-NN 0.6929
ensemble 0.6755
Best accuracy : 25-NN with accuracy 0.6929
```

Figure 3.7: Australian data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

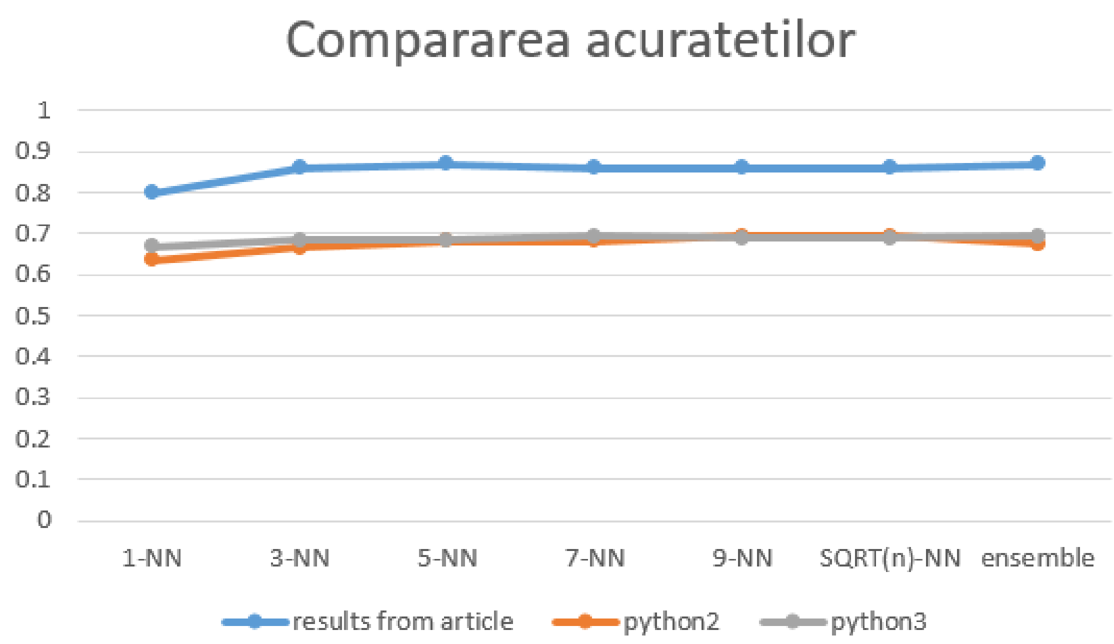


Figure 3.8: Australian data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.3 Balance data set

Balance data set contine 625 randuri de date, 4 feature-uri, feature-ul pe care il vom clasifica este F1 care are 3 posibile clase

```
print('Evaluate Balance dataset')
input_file = "balance.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(625))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.9: Balance data set

Rezultate

```
Evaluate Balance dataset
  1-NN 0.7776
  3-NN 0.7935
  5-NN 0.8352
  7-NN 0.8688
  9-NN 0.8833
 11-NN 0.8800
 13-NN 0.8800
 15-NN 0.8944
 17-NN 0.8928
 19-NN 0.8944
 21-NN 0.8929
 23-NN 0.8929
 25-NN 0.8928
ensemble 0.8881
Best accuracy : 15-NN with accuracy 0.8944
```

Figure 3.10: Balance data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

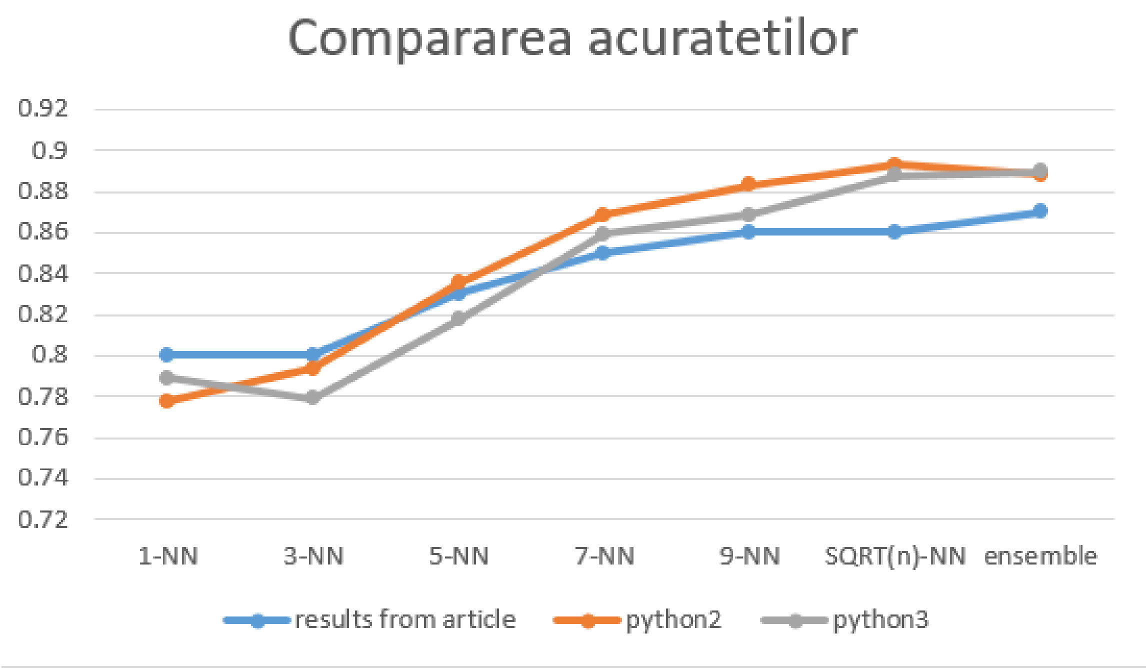


Figure 3.11: Balance data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.4 Banknote data set

Banknote data set contine 1372 randuri de date, 5 feature-uri, feature-ul pe care il vom clasifica este F5 care are 2 posibile clase

```
print('Evaluate Banknote dataset')
input_file = "banknote.csv"

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

X, y = data[data.columns.drop('F5')], data['F5']

n=int(math.sqrt(1372))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.12: Banknote data set

```
Evaluate Banknote dataset
  1-NN 0.9993
  3-NN 1.0000
  5-NN 1.0000
  7-NN 1.0000
  9-NN 1.0000
 11-NN 1.0000
 13-NN 0.9971
 15-NN 0.9964
 17-NN 0.9964
 19-NN 0.9949
 21-NN 0.9942
 23-NN 0.9927
 25-NN 0.9927
 27-NN 0.9927
 29-NN 0.9920
 31-NN 0.9920
 33-NN 0.9905
 35-NN 0.9891
 37-NN 0.9891
ensemble 0.9949
Best accuracy :  3-NN with accuracy 1.0000
```

Figure 3.13: Banknote data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

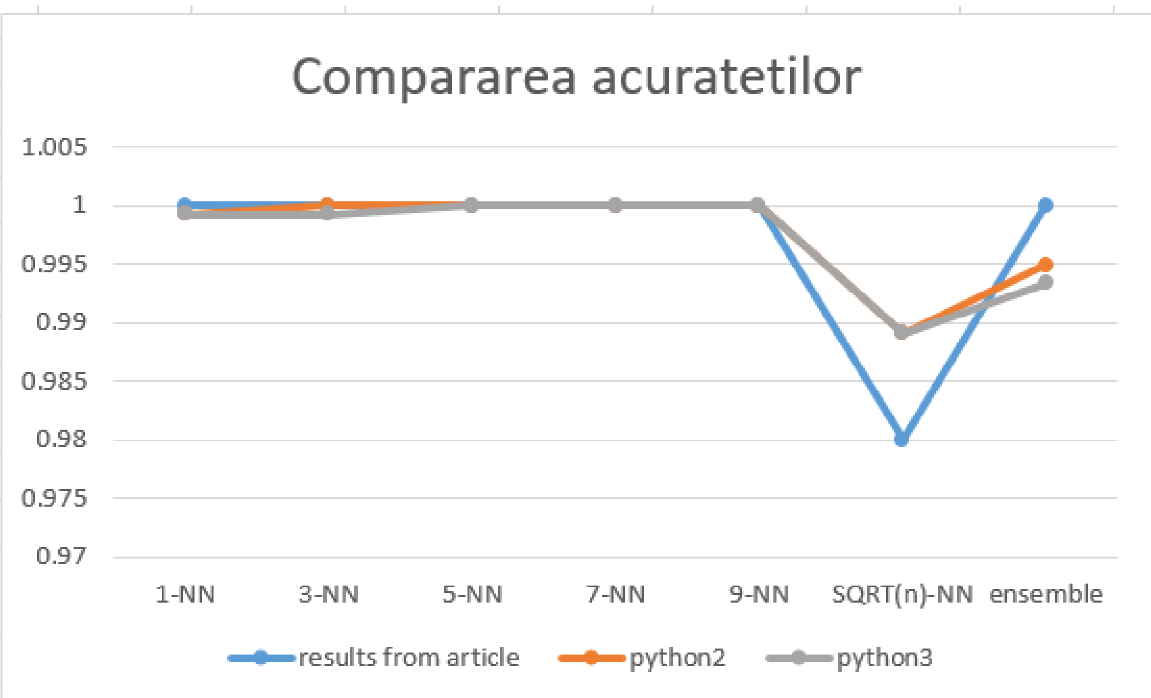


Figure 3.14: Banknote data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.5 EEG data set

EEG data set contine 14980 randuri de date, 15 feature-uri, feature-ul pe care il vom clasifica este F15 care are 2 posibile clase

```
print('Evaluate EEG dataset')
input_file = "EEG.csv"

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

X, y = data[data.columns.drop('F15')], data['F15']

n=int(math.sqrt(14980))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.15: EEG data set

```
  1-NN 0.9801
  3-NN 0.9761
  5-NN 0.9701
  7-NN 0.9656
  9-NN 0.9599
 11-NN 0.9566
 13-NN 0.9547
 15-NN 0.9509
 17-NN 0.9485
 19-NN 0.9459
 21-NN 0.9438
 23-NN 0.9403
 25-NN 0.9386
 27-NN 0.9352
 29-NN 0.9326
 31-NN 0.9307
 33-NN 0.9298
 35-NN 0.9272
 37-NN 0.9253
 39-NN 0.9236
 41-NN 0.9212
 43-NN 0.9189
 45-NN 0.9174
 47-NN 0.9169
 49-NN 0.9152
 51-NN 0.9128
 53-NN 0.9115
 55-NN 0.9097
 57-NN 0.9085
 59-NN 0.9060
 61-NN 0.9045
 63-NN 0.9037
 65-NN 0.9023
 67-NN 0.9016
 69-NN 0.9001
 71-NN 0.8991
 73-NN 0.8981
 75-NN 0.8967
 77-NN 0.8956
 79-NN 0.8944
 81-NN 0.8941
 83-NN 0.8923
 85-NN 0.8919
 87-NN 0.8902
 89-NN 0.8893
 91-NN 0.8892
 93-NN 0.8875
 95-NN 0.8870
 97-NN 0.8864
 99-NN 0.8850
101-NN 0.8839
103-NN 0.8836
105-NN 0.8822
107-NN 0.8816
109-NN 0.8808
111-NN 0.8797
113-NN 0.8784
115-NN 0.8782
117-NN 0.8774
119-NN 0.8770
121-NN 0.8758
ensemble 0.9077
Best accuracy :  1-NN with accuracy 0.9801
```

Figure 3.16: EEG data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

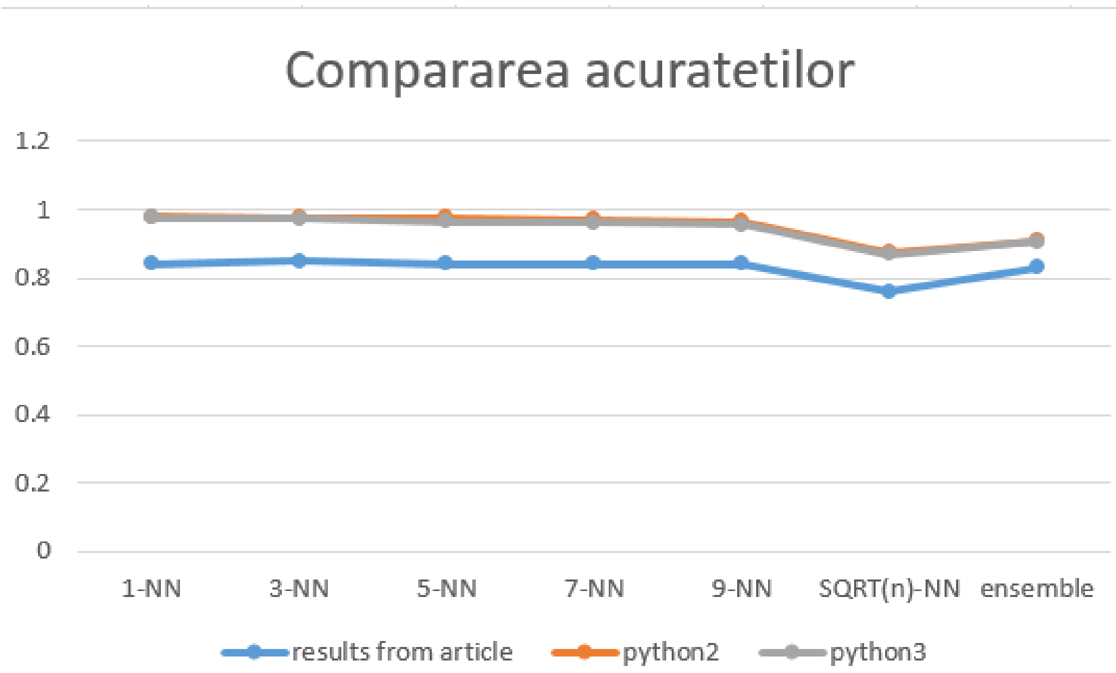


Figure 3.17: EEG data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.6 Haberman data set

Haberman data set contine 306 randuri de date, 4 feature-uri, feature-ul pe care il vom clasifica este F4 care are 2 posibile clase

```
print('Evaluate Haberman dataset')
input_file = "haberman.csv"

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

X, y = data[data.columns.drop('F4')], data['F4']

n=int(math.sqrt(306))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.18: Haberman data set

Rezultate

```
Evaluate Haberman dataset
  1-NN 0.6989
  3-NN 0.6927
  5-NN 0.6992
  7-NN 0.7254
  9-NN 0.7351
 11-NN 0.7351
 13-NN 0.7450
 15-NN 0.7417
 17-NN 0.7385
ensemble 0.7351
Best accuracy : 13-NN with accuracy 0.7450
```

Figure 3.19: Haberman data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

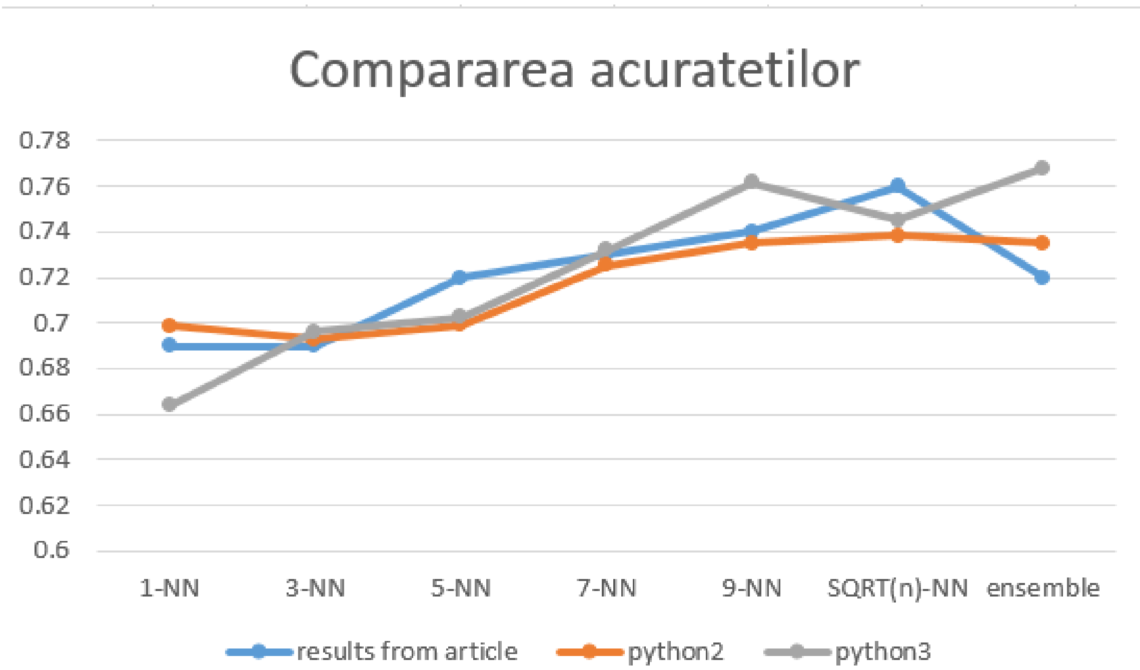


Figure 3.20: Haberman data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.7 Heart data set

Heart data set contine 271 randuri de date, 14 feature-uri, feature-ul pe care il vom clasifica este F14 care are 2 posibile clase

```
print('Evaluate Heart dataset')
input_file = "heart.csv"

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

X, y = data[data.columns.drop('F14')], data['F14']

n=int(math.sqrt(271))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.21: Heart data set

Rezultate

```
Evaluate Heart dataset
  1-NN 0.5926
  3-NN 0.6778
  5-NN 0.6593
  7-NN 0.6667
  9-NN 0.6481
 11-NN 0.6519
 13-NN 0.6593
 15-NN 0.6778
ensemble 0.6593
Best accuracy :  3-NN with accuracy 0.6778
```

Figure 3.22: Heart data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

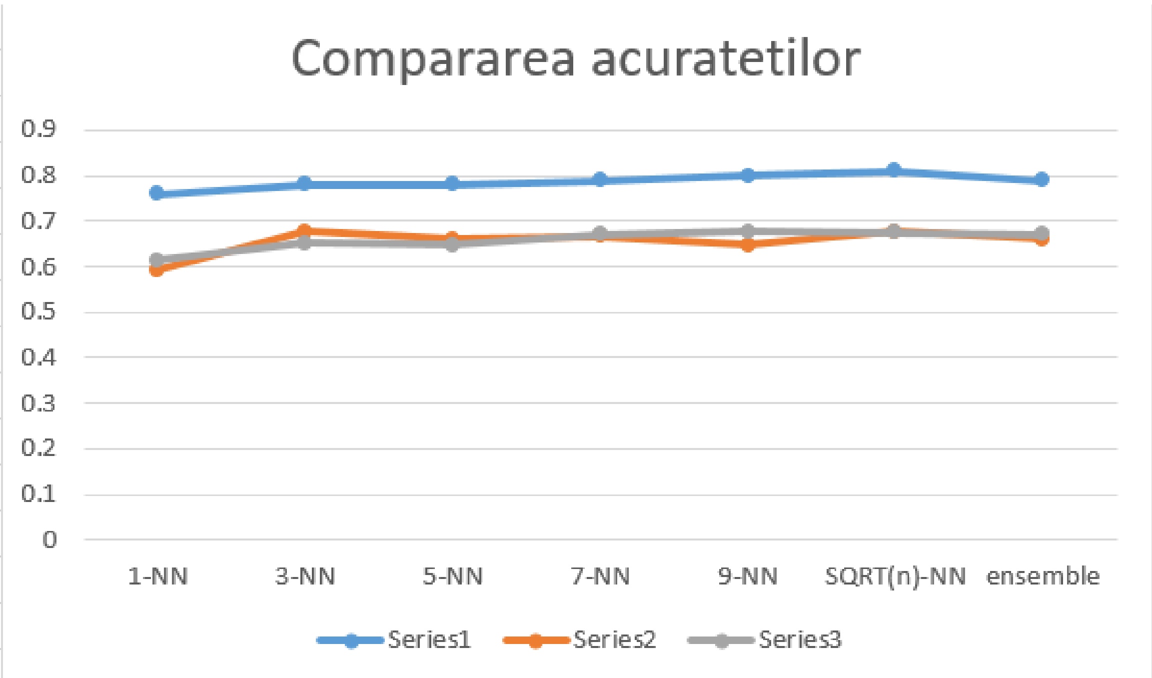


Figure 3.23: Heart data set, rezultatele propuse de autor sunt subliniate cu galben in articol

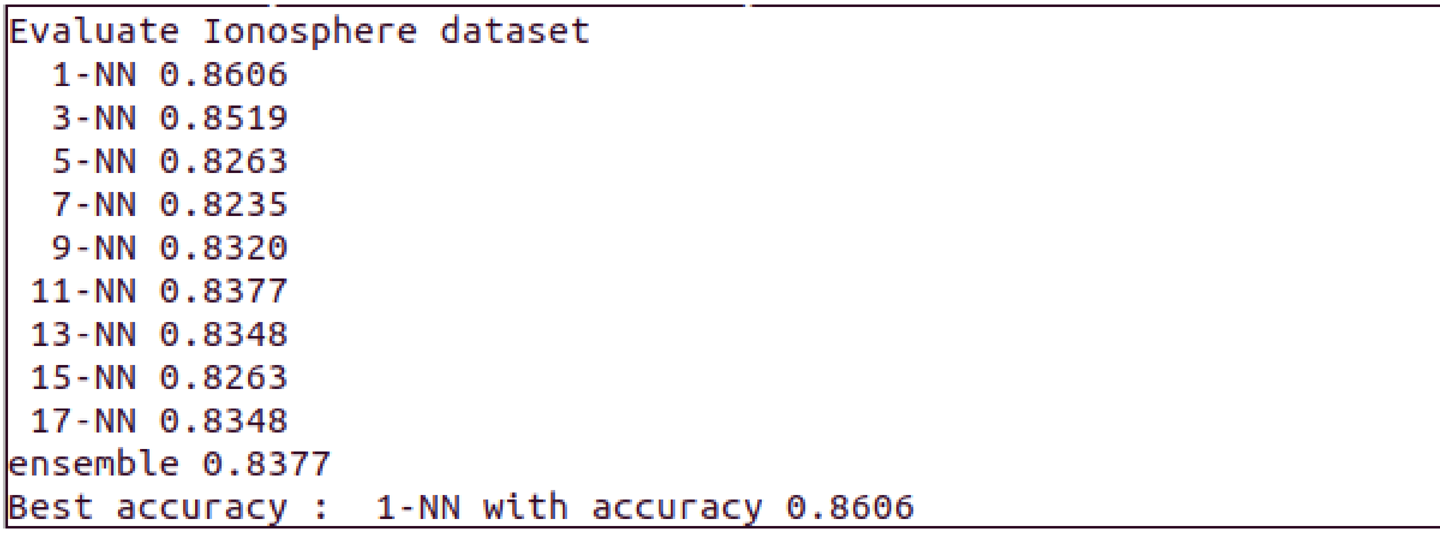
3.8 Ionosphere data set

Ionosphere data set contine 351 randuri de date, 35 feature-uri, feature-ul pe care il vom clasifica este F35 care are 2 posibile clase

```
print('Evaluate Ionosphere dataset')
input_file = "ionosphere.csv"

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

X, y = data[data.columns.drop('F35')], data['F35']

n=int(math.sqrt(351))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.24: Ionosphere data set

Rezultate



```
Evaluate Ionosphere dataset
  1-NN 0.8606
  3-NN 0.8519
  5-NN 0.8263
  7-NN 0.8235
  9-NN 0.8320
 11-NN 0.8377
 13-NN 0.8348
 15-NN 0.8263
 17-NN 0.8348
ensemble 0.8377
Best accuracy :  1-NN with accuracy 0.8606
```

Figure 3.25: Ionosphere data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

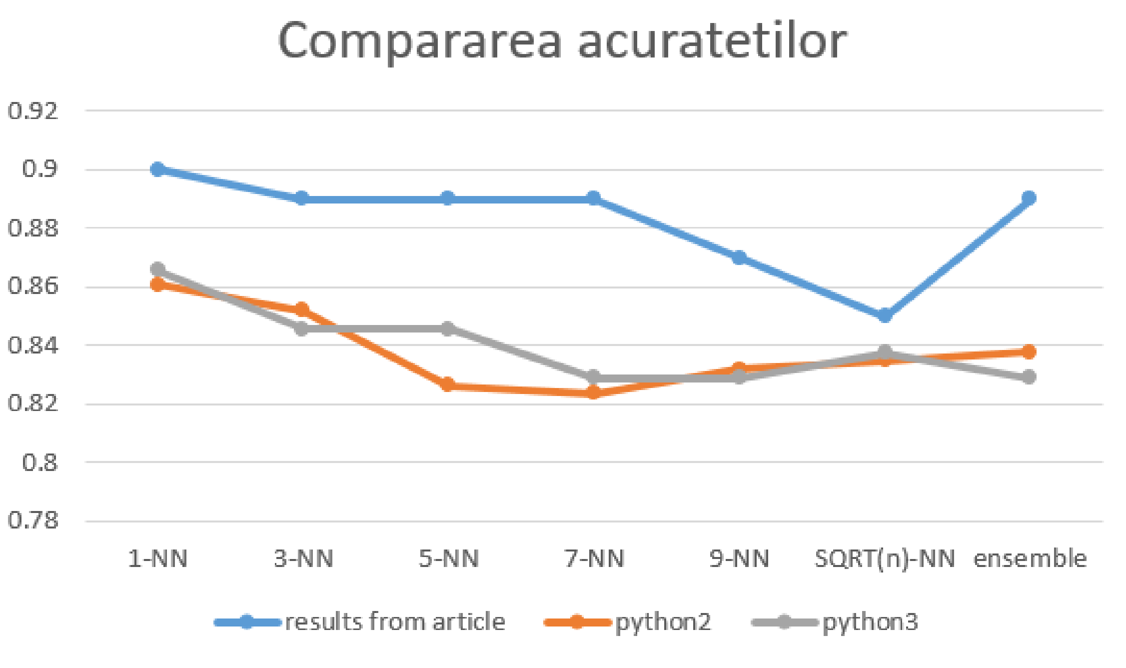


Figure 3.26: Ionosphere data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.9 Iris data set

Iris data set contine 151 randuri de date, 5 feature-uri, feature-ul pe care il vom clasifica este F5 care are 3 posibile clase

```
print('Evaluate Iris dataset')
input_file = "iris.csv"

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

X, y = data[data.columns.drop('F5')], data['F5']

n=int(math.sqrt(151))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.27: Iris data set

Rezultate

```
Evaluate Iris dataset
  1-NN 0.9667
  3-NN 0.9667
  5-NN 0.9600
  7-NN 0.9733
  9-NN 0.9600
 11-NN 0.9600
ensemble 0.9667
Best accuracy :  7-NN with accuracy 0.9733
```

Figure 3.28: Iris data set rezultate python 2.7.3Rezultatele python3 sunt afisate in notebook

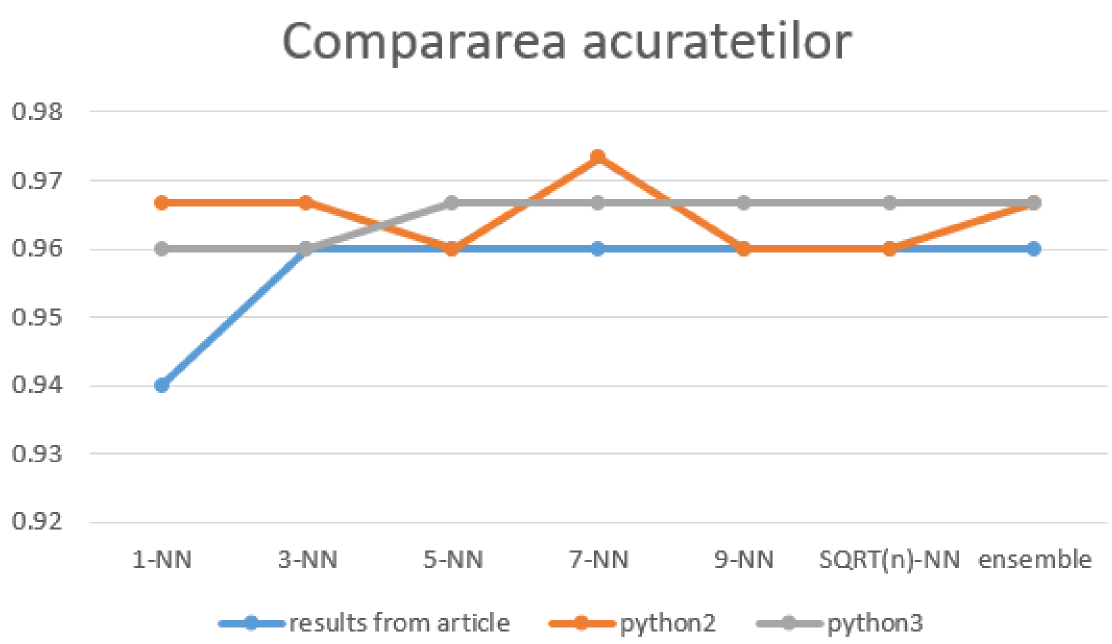


Figure 3.29: Iris data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.10 Letter Recognition data set

Letter recognition data set contine 20000 randuri de date, 16 feature-uri, feature-ul pe care il vom clasifica este F1 care are 26 posibile clase

```
print('Evaluate Letter-Recognition dataset')
input_file = "letter-recognition.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(20000))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.30: Letter recognition data set

Rezultate

```
  1-NN 0.9616
  3-NN 0.9560
  5-NN 0.9563
  7-NN 0.9538
  9-NN 0.9514
 11-NN 0.9477
 13-NN 0.9464
 15-NN 0.9446
 17-NN 0.9407
 19-NN 0.9391
 21-NN 0.9357
 23-NN 0.9330
 25-NN 0.9298
 27-NN 0.9278
 29-NN 0.9247
 31-NN 0.9227
 33-NN 0.9204
 35-NN 0.9181
 37-NN 0.9151
 39-NN 0.9133
 41-NN 0.9114
 43-NN 0.9084
 45-NN 0.9071
 47-NN 0.9043
 49-NN 0.9040
 51-NN 0.8995
 53-NN 0.8992
 55-NN 0.8958
 57-NN 0.8940
 59-NN 0.8896
 61-NN 0.8874
 63-NN 0.8857
 65-NN 0.8839
 67-NN 0.8814
 69-NN 0.8798
 71-NN 0.8755
 73-NN 0.8737
 75-NN 0.8701
 77-NN 0.8681
 79-NN 0.8662
 81-NN 0.8633
 83-NN 0.8613
 85-NN 0.8587
 87-NN 0.8576
 89-NN 0.8555
 91-NN 0.8535
 93-NN 0.8513
 95-NN 0.8488
 97-NN 0.8470
 99-NN 0.8463
101-NN 0.8440
103-NN 0.8419
105-NN 0.8407
107-NN 0.8387
109-NN 0.8368
111-NN 0.8353
113-NN 0.8339
115-NN 0.8319
117-NN 0.8304
119-NN 0.8285
121-NN 0.8270
123-NN 0.8255
125-NN 0.8239
127-NN 0.8206
129-NN 0.8200
131-NN 0.8184
133-NN 0.8177
135-NN 0.8164
137-NN 0.8144
139-NN 0.8127
141-NN 0.8128
ensemble 0.8836
Best accuracy :  1-NN with accuracy 0.9616
```

Figure 3.31: Letter recognition data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

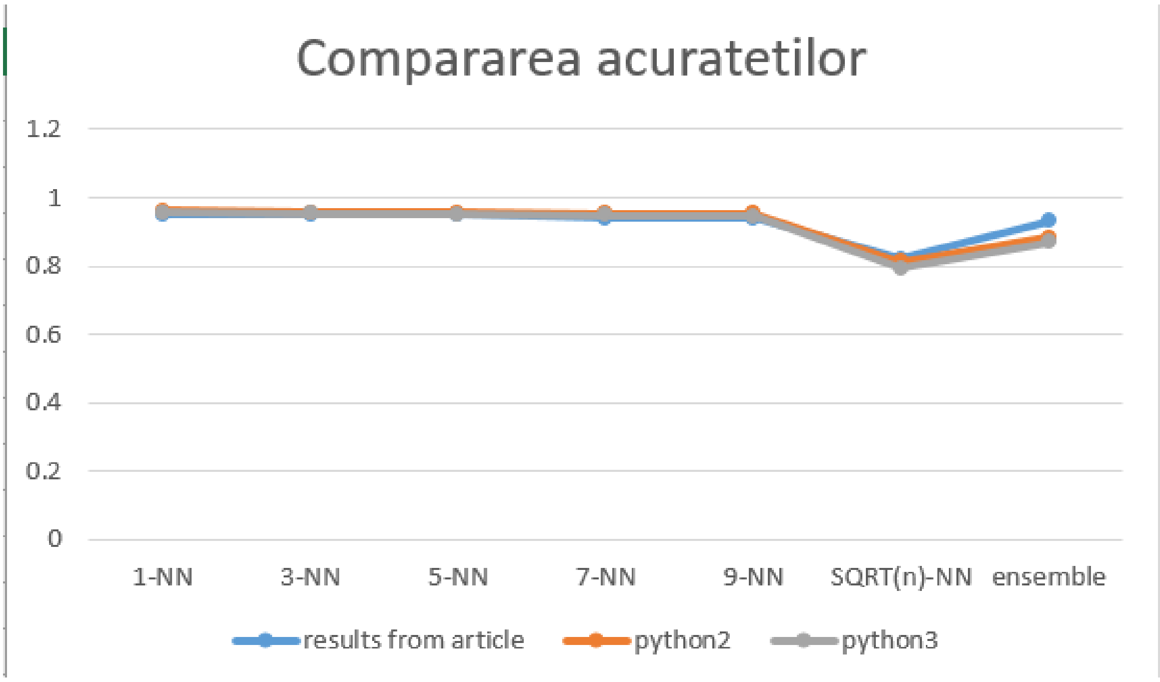


Figure 3.32: Letter recognition data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.11 Liver data set

Liver data set contine 345 randuri de date, 7 feature-uri, feature-ul pe care il vom clasifica este F7 care are 2 posibile clase

```
print('Evaluate Liver dataset')
input_file = "liver.csv"

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

X, y = data[data.columns.drop('F7')], data['F7']

n=int(math.sqrt(345))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.33: Liver data set

Rezultate

```
Evaluate Liver dataset
  1-NN 0.6319
  3-NN 0.6638
  5-NN 0.6754
  7-NN 0.6899
  9-NN 0.6783
 11-NN 0.6754
 13-NN 0.6841
 15-NN 0.6986
 17-NN 0.6754
ensemble 0.6812
Best accuracy : 15-NN with accuracy 0.6986
```

Figure 3.34: Liver data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

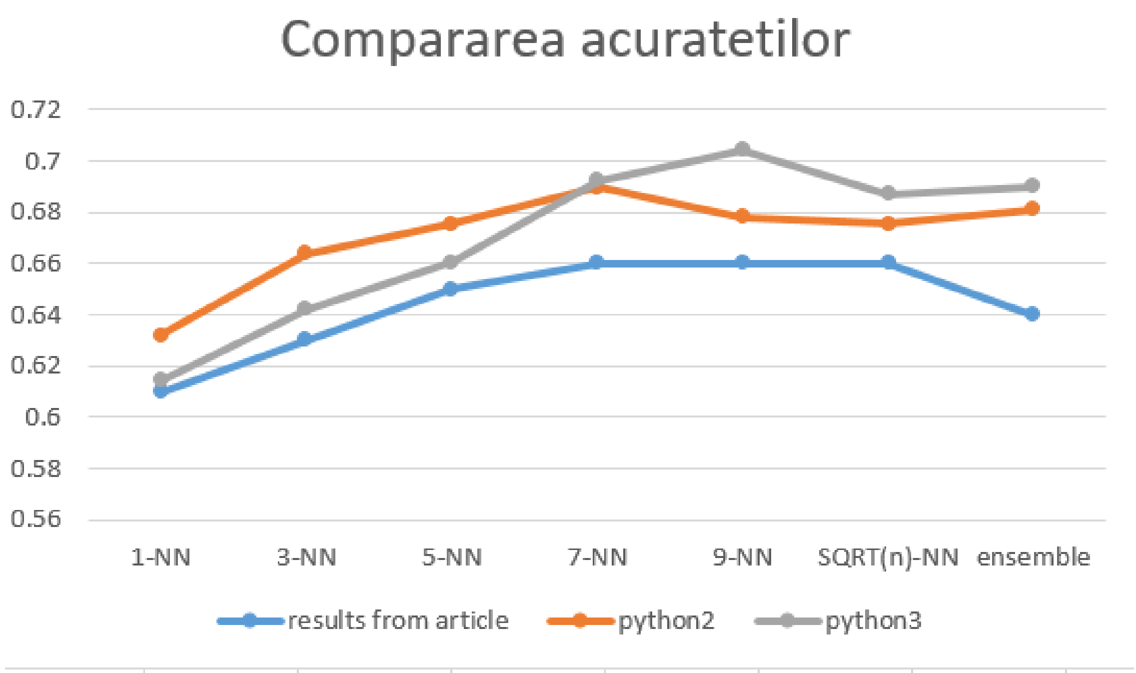


Figure 3.35: Liver data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.12 Parkinson data set

Parkinson data set contine 1040 randuri de date, 27 feature-uri, feature-ul pe care il vom clasifica este F1 care are 2 posibile clase

```
print('Evaluate Parkinson dataset')
input_file = "parkinson.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(168))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.36: Parkinson data set

Rezultate

```
Evaluate Parkinson dataset
  1-NN 0.6179
  3-NN 0.5643
  5-NN 0.5357
  7-NN 0.4929
  9-NN 0.4393
 11-NN 0.4071
ensemble 0.5071
Best accuracy :  1-NN with accuracy 0.6179
```

Figure 3.37: Parkinson data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

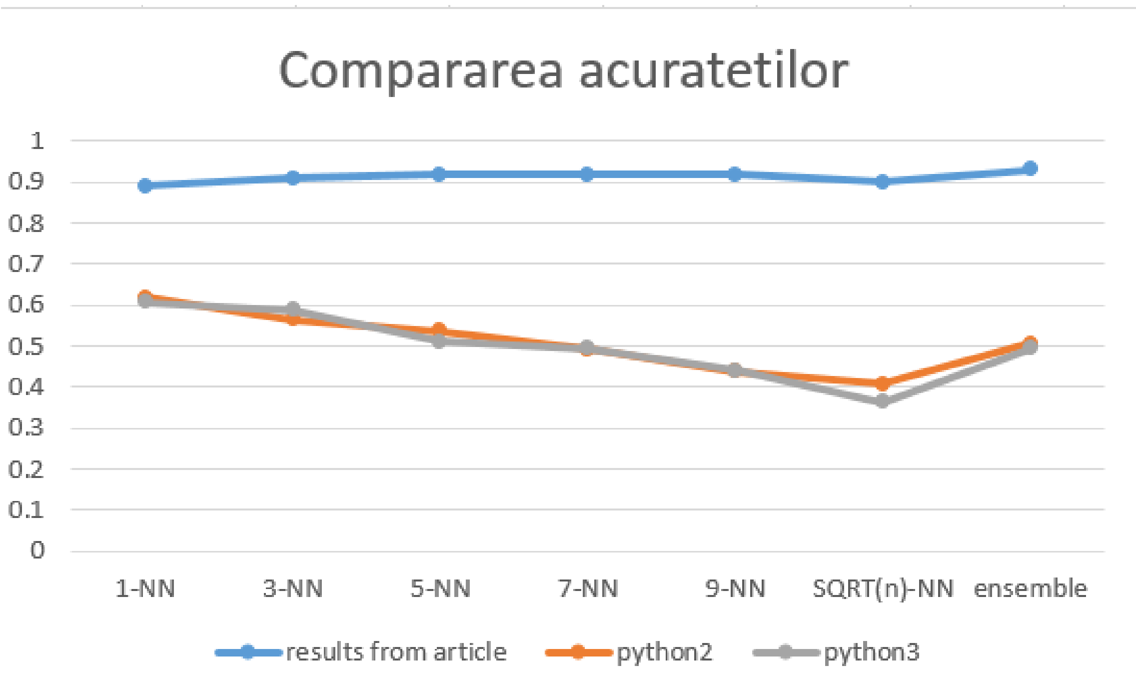


Figure 3.38: Parkinson data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.13 QSAR data set

QSAR data set contine 1055 randuri de date, 43 feature-uri, feature-ul pe care il vom clasifica este F43 care are 2 posibile clase

```
input_file = "QSAR .csv"

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

X, y = data[data.columns.drop('F43')], data['F43']

n=int(math.sqrt(1055))


if(n % 2 == 0):
        n=n-1

models = get_models(n)

# evaluate the models and store results (sorted)
results, names = list(), list()

for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
```

Figure 3.39: QSAR data set

Rezultate

```
Evaluate QSAR dataset
  1-NN 0.7962
  3-NN 0.8143
  5-NN 0.8095
  7-NN 0.8161
  9-NN 0.8076
 11-NN 0.8037
 13-NN 0.8057
 15-NN 0.8028
 17-NN 0.7990
 19-NN 0.7877
 21-NN 0.7763
 23-NN 0.7725
 25-NN 0.7706
 27-NN 0.7706
 29-NN 0.7734
 31-NN 0.7744
ensemble 0.8047
Best accuracy :  7-NN with accuracy 0.8161
Evaluate Australian dataset
```

Figure 3.40: QSAR data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

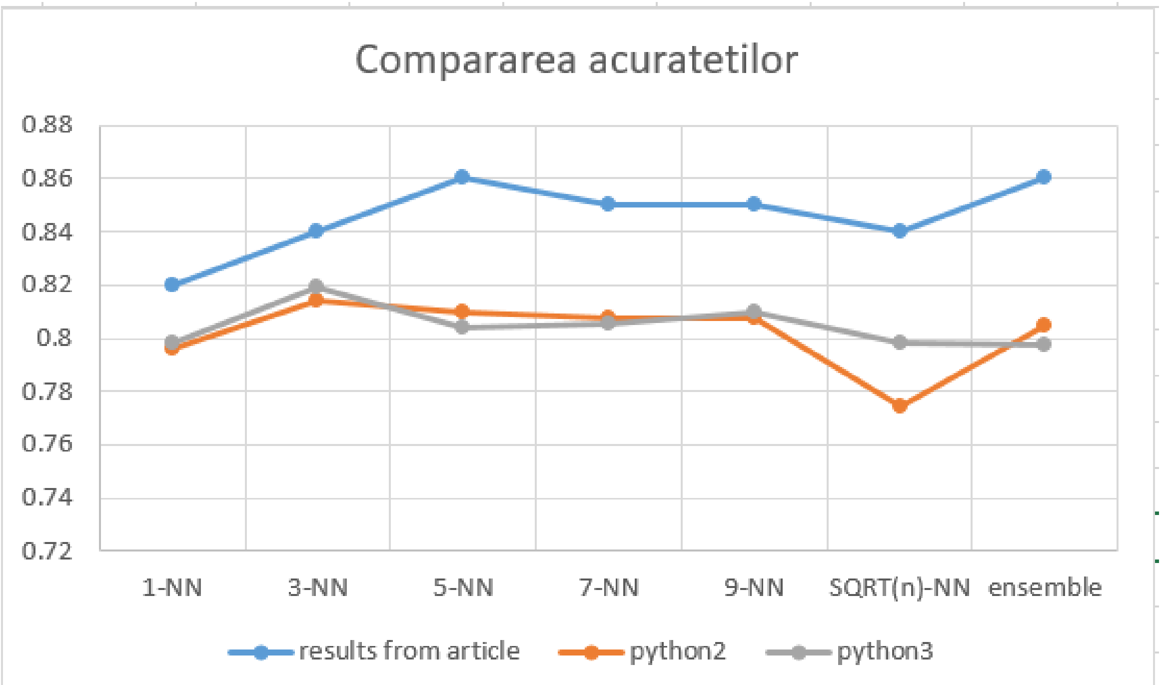


Figure 3.41: QSAR data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.14 Sonar data set

Sonar data set contine 209 randuri de date, 61 feature-uri, feature-ul pe care il vom clasifica este F61 care are 2 posibile clase

```
print('Evaluate Sonar dataset')
input_file = "sonar.csv"

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

X, y = data[data.columns.drop('F61')], data['F61']

n=int(math.sqrt(209))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.42: Sonar data set

Rezultate

```
Evaluate Sonar dataset
  1-NN 0.8141
  3-NN 0.8330
  5-NN 0.8085
  7-NN 0.7264
  9-NN 0.7069
 11-NN 0.7065
 13-NN 0.7015
ensemble 0.7263
Best accuracy :  3-NN with accuracy 0.8330
```

Figure 3.43: Sonar data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

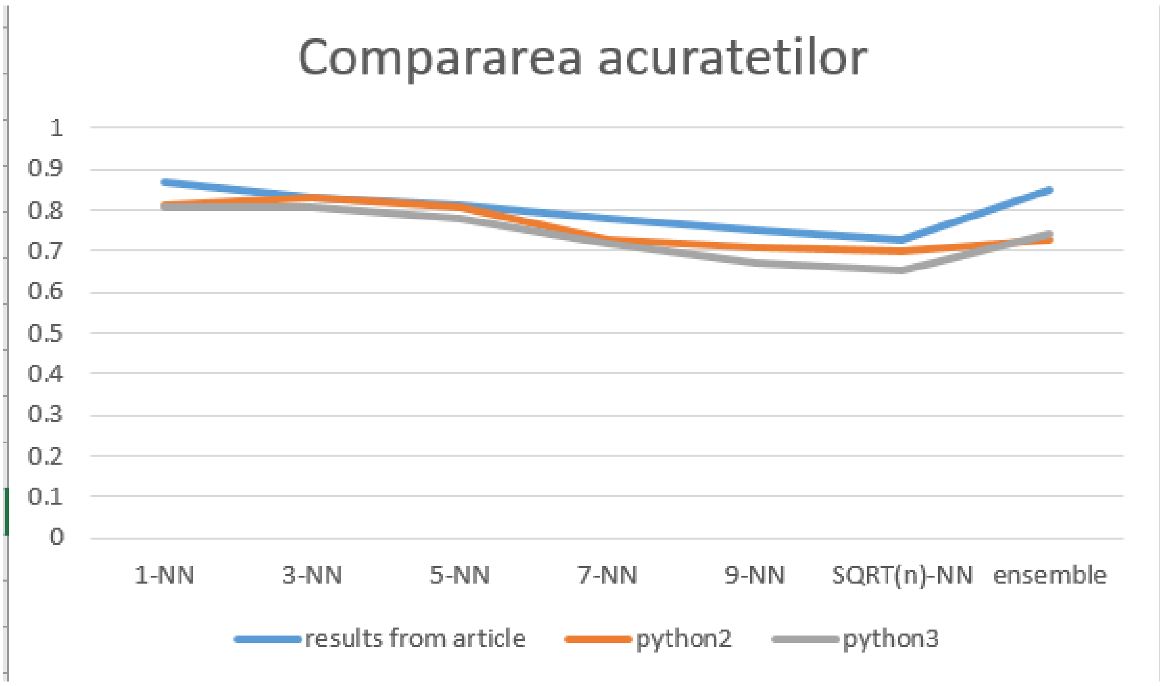


Figure 3.44: Sonar data set, rezultatele propuse de autor sunt subliniate cu galben in articol

3.15 Wine data set

Wine data set contine 179 randuri de date, 13 feature-uri, feature-ul pe care il vom clasifica este F1 care are 3 posibile clase

```
print('Evaluate Wine dataset')
input_file = "wine.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(179))


if(n % 2 == 0):
        n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
        scores = evaluate_model(model)
        results.append(scores)
        names.append(name)
        zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
        print('%s %.4f ' % (names[x], mean(results[x])))
        if(mean(results[x])> bestAccuracy):
                bestName= names[x];
                bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```

Figure 3.45: Wine data set

```
Evaluate Wine dataset
  1-NN 0.7475
  3-NN 0.6910
  5-NN 0.6852
  7-NN 0.6795
  9-NN 0.7080
 11-NN 0.7139
 13-NN 0.7022
ensemble 0.7080
Best accuracy :  1-NN with accuracy 0.7475
```

Figure 3.46: Wine data set rezultate python 2.7.3
Rezultatele python3 sunt afisate in notebook

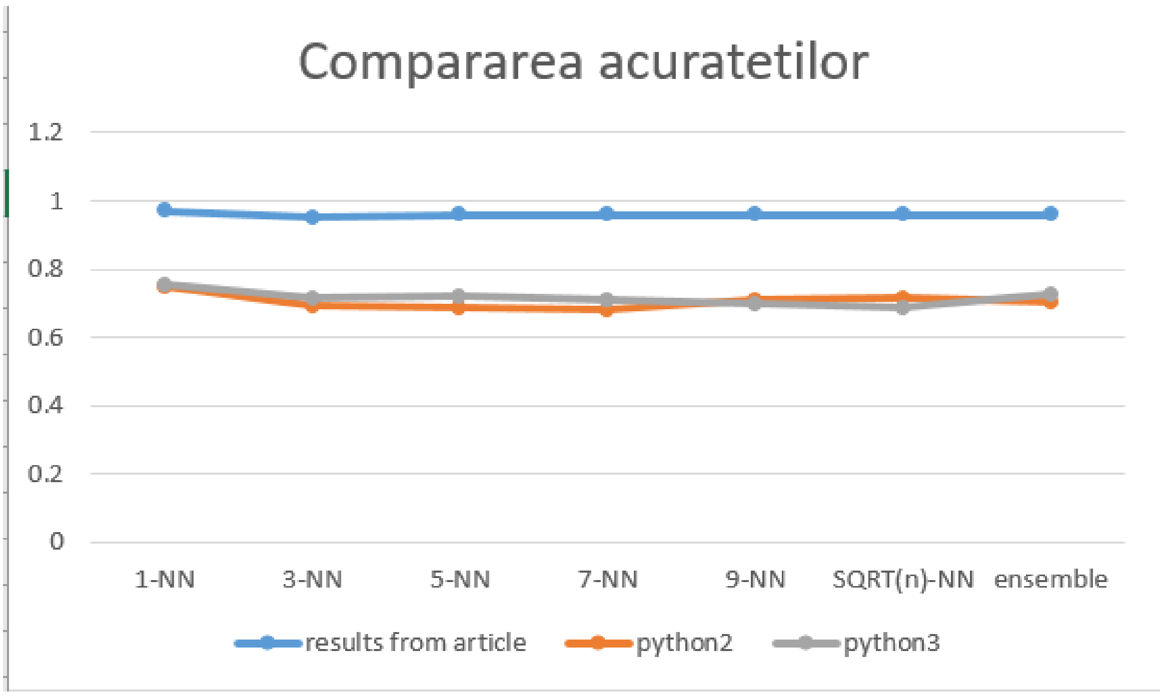


Figure 3.47: Wine data set, rezultatele propuse de autor sunt subliniate cu galben in articol

Chapter 4

Conclizii

In urma experimentelor am remarcat ca desi clasificatorul asamblat descris in articol nu depaseste performanta celui mai bun clasificator KNN din ansamblul sau performanta ansamblului este foarte apropiata de cea mai buna performanta, scutundu-ne de cautarea parametrului k care ar avea cea mai buna performanta.

Acuratete mai mica decat in articol

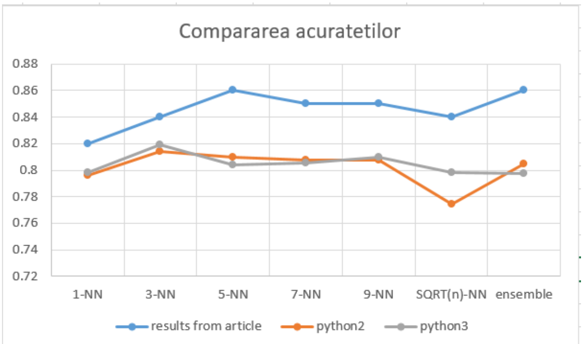


Figure 4.1: Qsar data set

Acuratete identica cu cea din articol

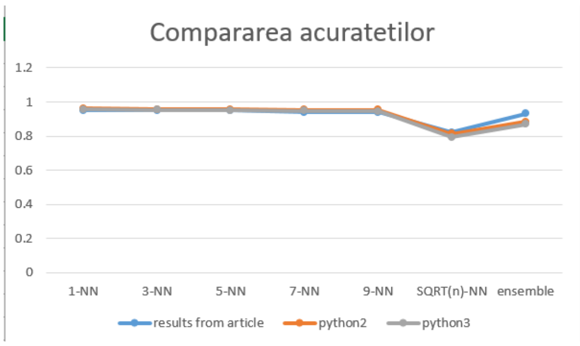


Figure 4.2: Letter recognition data set

Acuratetea din articol <acuratetea oferita de python3 <python 2

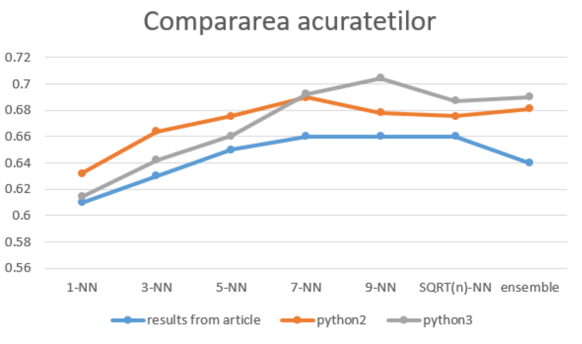


Figure 4.3: Liver data set

Acuratete identica intre python2 si python 3 si mai mare decat cea din articol

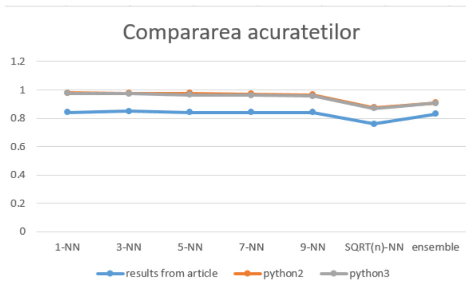


Figure 4.4: EEG data set

Chapter 5

Appendix

Python code

```
#import libraries
import pandas as pd
import numpy as np

from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score
from sklearn.utils import shuffle


from sklearn.ensemble import VotingClassifier
import math

# get a voting ensemble of models
def get_voting(n):
  k=-1; count=0; models = list(); label="-NN"; labelList=[];
  while k<n:
    k=k+2;
    count=count+1;
    labelList.append(str(k)+label)
    # define the base models
    models.append((str(k)+label, KNeighborsClassifier(n_neighbors=k)))
  # define the voting ensemble
  ensemble = VotingClassifier(estimators=models, voting='hard')
  return ensemble

# get a list of models to evaluate
def get_models(n):
  models = dict()
  k=-1; count=0; label="-NN"; labelList=[];
  while k<n:
    k=k+2;
    count=count+1;
    labelList.append(str(k)+label)
    # define the base models
    if(k<10):
      models['  '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
    elif(k>10 and k<100):
      models[' '+str(k)+label] = KNeighborsClassifier(n_neighbors=k)
```

```
    else:
      models[str(k)+label] = KNeighborsClassifier(n_neighbors=k)

  models['ensemble'] = get_voting(n)
  return models

# evaluate a give model using cross-validation
def evaluate_model(model):
  cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=1)
  scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs
   =-1)
  return scores

print('Evaluate QSAR dataset')
input_file = "QSAR .csv"

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

X, y = data[data.columns.drop('F43')], data['F43']

n=int(math.sqrt(1055))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Australian dataset')
input_file = "australian.csv"

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

X, y = data[data.columns.drop('F15')], data['F15']

n=int(math.sqrt(690))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
```

```
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Balance dataset')
input_file = "balance.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(625))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Banknote dataset')
input_file = "banknote.csv"

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

X, y = data[data.columns.drop('F5')], data['F5']

n=int(math.sqrt(1372))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
```

```
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Haberman dataset')
input_file = "haberman.csv"

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

X, y = data[data.columns.drop('F4')], data['F4']

n=int(math.sqrt(306))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Heart dataset')
input_file = "heart.csv"

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

X, y = data[data.columns.drop('F14')], data['F14']

n=int(math.sqrt(271))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
```

```
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Ionosphere dataset')
input_file = "ionosphere.csv"

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

X, y = data[data.columns.drop('F35')], data['F35']

n=int(math.sqrt(351))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Iris dataset')
input_file = "iris.csv"

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

X, y = data[data.columns.drop('F5')], data['F5']

n=int(math.sqrt(151))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
```

```python
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))



print('Evaluate Liver dataset')
input_file = "liver.csv"

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

X, y = data[data.columns.drop('F7')], data['F7']

n=int(math.sqrt(345))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Parkinson dataset')
input_file = "parkinson.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(168))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
```

```
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Sonar dataset')
input_file = "sonar.csv"

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

X, y = data[data.columns.drop('F61')], data['F61']

n=int(math.sqrt(209))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Wine dataset')
input_file = "wine.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(179))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
```

```
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate EEG dataset')
input_file = "EEG.csv"

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

X, y = data[data.columns.drop('F15')], data['F15']

n=int(math.sqrt(14980))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))

print('Evaluate Letter-Recognition dataset')
input_file = "letter-recognition.csv"

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

X, y = data[data.columns.drop('F1')], data['F1']

n=int(math.sqrt(20000))


if(n % 2 == 0):
  n=n-1

models = get_models(n)
# evaluate the models and store results
```

```
results, names = list(), list()
bestName="1NN"; bestAccuracy=0;
for name, model in models.items():
  scores = evaluate_model(model)
  results.append(scores)
  names.append(name)
  zipped= zip(names, results)
names, results = zip(*sorted(zipped))
for x in range (len(names)):
  print('%s %.4f ' % (names[x], mean(results[x])))
  if(mean(results[x])> bestAccuracy):
    bestName= names[x];
    bestAccuracy= mean(results[x]);
print('Best accuracy :%s with accuracy %.4f '% (bestName, bestAccuracy))
```
Listing 5.1: KNN ensemble implementation

Chapter 6

Bibliografie

1. `http://archive.ics.uci.edu/ml` sursa seturilor de date citata de autor
2. http://scikit-learn.org/stable/
3. https://arxiv.org/pdf/1409.0919.pdf
4. Notebook Link 1: `https://view.datalore.io/notebook/Oii225mTnR6rHuFf6cfDO`
5. Notebook Link 2: `https://mybinder.org/v2/gh/Salamonul/ExamenSistemeInte` `004cb12b18dc87f5c56975cb50b0e3cf8e647c5f`

Intelligent Systems Group