L 2 MI Info232 – Mini Projet

FeatureSelectionChallenge

HABINSHUTI THIERRY

lien du challenge : https://competitions.codalab.org/competitions/3931?secret_key=d6c218a3-3b83-4eed-8e39-5b895c5a5e35#learn_the_details

Lien GitHub : https://github.com/2chenj/FeatureSelectionChallenge

Numéro de soumission : 601404

Lien de vidéo : https://youtu.be/-Dux2qiME4s

Lien des diapositives :
https://github.com/2chenj/FeatureSelectionChallenge/blob/master/miniProjetRattrapage.pptx

Table des matières

INTRODUCTION

Feature selection (sélection de caractéristiques/attributs) est l'une des premières étapes et l'une des plus importants en machine learning (apprentissage automatique). Etant donnée une donnée, il arrive que certaines caractéristiques n'aient pas ou aie peu d'impact sur la sortie. Et même souvent garder ces caractéristiques inutiles gâtent le model. De ce fait il est important d'effectuer la sélection de caractéristiques.
Dans un premier temps on va choisir le model approprié à chaque donné
Puis dans un second temps on essaiera plusieurs méthodes de sélection de caractéristiques présentes dans la bibliothèque de scikit learn pour ne garder que la meilleure d'entre elles.

STATISTIQUES SUR NOS DONNEES

Nous avons en notre possession 5 donnés : ARCENE DEXTERE, DOROTHEA, GISETTE, MADELON.
Dans chaque donné nous avons dataNames_valid.labels, dataNames.param, dataNames_train.data, dataNames_train.labels, dataNames_test.data, dataNames_valid.data

Tableau 1 : statistiques sur les données : ARCENE

DATASET	Nombre d'exemples (Instances) = nombre de lignes.	Nombre de Variables (Attribues/Features) = nombre de colonnes.	Sparsité (fraction de zéros)	Y-a-t-il variables catégorielles (ou <<nominales>>) ?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemples dans chaque Classe
Train (données d'apprentissage avec label)	100	10000	Non-sparse Matrix	Integer	NON	Class 1 : 44 Class -1 : 56
Valid (données phase 1 de test, labels à prédire)	100	10000	Non-sparse Matrix	Integer	NON	Class 1 : 44 Class -1 : 56
Test (données phase 2 de test, labels à prédire)	700	10000	Non-sparse Matrix	Integer	NON	Class 1 = 310 Clase -1 = 390

Tableau 1 : statistiques sur les données : GISETTE

DATASET	Nombre d'exemples (Instances) = nombre de lignes.	Nombre de Variables (Attribues/Features) = nombre de colones.	Sparsité (fraction de zéros)	Y-a-t-il variables catégorielles (ou <<nominales>>) ?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemples dans chaque Classe
Train (données d'apprentissage avec label)	6000	5000	rather-sparse	Integer	NON	Class 1 : 3000 Class -1 : 3000
Valid (données phase 1 de test, labels à prédire)	1000	5000	rather-sparse	Integer	NON	Class 1 : 500 Class -1 : 500
Test (données phase 2 de test, labels à predire)	13500	5000	rather-sparse	Integer	NON	Class 1 : 6750 Class -1 : 6750

Tableau 1 : statistiques sur les données:DEXTER

DATASET	Nombre d'exemples (Instances) = nombre de lignes.	Nombre de Variables (Attributes/Features) = nombre de colones.	Sparsité (fraction de zeros)	Y-a-t-il variables catégorielles (ou <<nominales>>) ?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemples dans chaque Classe
Train (données d'apprentissage avec label)	300	20000	Very-sparse	Sparse-integer	NON	Class 1 : 150 Class -1 : 150
Valid (données phase 1 de test, labels à prédire)	300	20000	very-sparse	Sparse-integer	NON	Class 1 : 150 Class -1 : 150
Test (données phase 2 de test, labels à prédire)	2000	20000	Very-sparse	Sparse-integer	NON	Class 1 : 1000 Class -1 : 1000

Tableau 1 : statistiques sur les données:DOROTHEA

DATASET	Nombre d'exemples (Instances) = nombre de lignes.	Nombre de Variables (Attributes/Features) = nombre de colones.	Sparsité (fraction de zeros)	Y-a-t-il variables catégorielles (ou <<nominales>>) ?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemples dans chaque Classe
Train (données d'apprentissage avec label)	800	100000	very-sparse	Binary	NON	Class 1 : 78 Class -1 : 722
Valid (données phase 1 de test, labels à prédire)	350	100000	very-sparse	Binary	NON	Class 1 : 34 Class -1 : 316
Test (données phase 2 de test, labels à prédire)	800	100000	Veryt-sparse	Binary	NON	Class 1 : 190 Class -1 : 1760

Tableau 1 : statistiques sur les données:MADELON

DATASET	Nombre d'exemples (Instances) = nombre de lignes.	Nombre de Variables (Attribues/Features) = nombre de colones.	Sparsité (fraction de zeros)	Y-a-t-il variables catégorielles (ou <<nominales>>)?	Y-a-t-il des valeurs manquantes ?	Nombre d'exemples dans chaque Classe
Train (données d'apprentissage avec label)	2000	500	Non-sparse	Integer	NON	Class 1 : 1000 Class -1 : 1000
Valid (données phase 1 de test, labels à prédire)	600	500	Non-sparse	Integer	NON	Class 1 : 300 Class -1 : 300
Test (données phase 2 de test, labels à prédire)	1800	500	Non-sparse	Integer	NON	Class 1 : 900 Class -1 : 900

SELECTION DE MODEL (classifier)

Pour choisir le meilleurs model on utilisera la fonction cross_val_score (CV) du module model_selection de scikitlearn avec pour paramètre dataName_train.data et dataName_train.labels, Pour calculer l'accuracy score
Ce même model sera réutilisé pour faire des prédictions sur dataName_test.data qui seront publié sur Codalab

Table2 : Résultats : ARCENE

Performances	OneR	Naive Bayes	Decision Tree	Ridge regression	Nearest Neighbor	Random Forest
Train. Sur les données d'apprentissage		0,54	0,77		0,88	0,7
CV. Obtenues par cross-validation.		0,55	0,77		0,81	0,73
Valid. Obtenues sur le leaderboard en soumettant sur les résultats sur le site						

Table2 : Résultats : GISETTE

Performances	OneR	Naive Bayes	Decision Tree	Ridge regression	Nearest Neighbor	Random Forest
Train. Sur les données d'apprentissage		0,754	0,928		0,962	0,887
CV. Obtenues par cross-validation.		0,732	0,93		0,959	0,892
Valid. Obtenues sur le leaderboard en soumettant sur les résultats sur le site						

Table2 : Résultats : DEXTER

Performances	OneR	Naive Bayes	Decision Tree	Ridge regression	Nearest Neighbor	Random Forest
Train. Sur les données d'apprentissage						
CV. Obtenues par cross-validation.						
Valid. Obtenues sur le leaderboard en soumettant sur les résultats sur le site						

Table2 : Résultats : DOROTHEA

Performances	OneR	Naive Bayes	Decision Tree	Ridge regression	Nearest Neighbor	Random Forest
Train. Sur les données d'apprentissage						
CV. Obtenues par cross-validation.						
Valid. Obtenues sur le leaderboard en soumettant sur les résultats sur le site						

Table2 : Résultats : MADELON

Performances	OneR	Naive Bayes	Decision Tree		Nearest Neighbor	Random Forest
Train. Sur les données d'apprentissage		0,6	0,76		0,65	0,62
CV. Obtenues par cross-validation.		0,59	0,75		0,64	0,62
Valid. Obtenues sur le leaderboard en soumettant sur les résultats sur le site						

En comparant les scores des model sur chaque donnée on voit qu'il faut utiliser Nearest Neighbour pour ARCENNE et GISETTE et plutôt decisionTree pour MADELON

DESCRIPTION DES ALGORITHMES ETUDIE

Les Etapes à faire sont les même pour les 5 différentes données qu'on a c'est pour cela qu'on va regarder en détail qu'une seule donnée pour éviter des répétition inutile

Etude de cas : ARCENE

Étape 1 : on importe les datas comme étant des dataframe ici on utilise pandas

#importation du module pandas
#création du dataframe df ;
#suppression d'éventuel valeurs null/NaN

```
Import pandas as pd
df = pd.read_csv(filename, sep=" ", header=None)
df = df.dropna(axis=1, how="any")
```

```
In [ ]:
data_dir = 'ARCENE'
data_name = 'arcene'
#!wc $data_dir/*

In [ ]:

In [3]:
# pandas module importation
import pandas as pd
#arcene_train.data importation as a dataframe
#removal of NaN values
trainData = pd.read_csv(data_dir  + '/' + data_name+"_train.data", sep=" ", header=None)  # The data are loaded as a Pand
trainData = trainData.dropna(axis=1, how="any")
#trainData # the standard output dataframe

In [4]:
# trainData.loc[:10,:10].plot() #selection of some column to plot

In [4]:
#arcene_train.labels importation as a dataframe
#removal of NaN values
trainLabels = pd.read_csv(data_dir  + '/' + data_name+"_train.labels", sep=" ",header=None, encoding='utf8')
trainLabels = trainLabels.dropna(axis=1, how="any")
#trainLabels # the standard output dataframe

In [ ]:

In [5]:
#arcene_valid.data importation as a dataframe
#removal of NaN values
validData = pd.read_csv(data_dir  + '/' + data_name+"_valid.data", sep=" ",header=None, encoding='utf8')
validData=validData.dropna(axis=1,how="any")
#validData  # the standard output dataframe

In [11]:
#arcene_valid.data importation as a dataframe
#removal of NaN values
validLabels = pd.read_csv(data_name+"_valid.labels", sep=" ", header=None)  # The data are loaded as a Pandas Data Frame
validLabels=validLabels.dropna(axis=1,how="any")
#validLabels  # the standard output dataframe

In [15]:
#arcene_test.data importation as a dataframe
#removal of NaN values
testData = pd.read_csv(data_dir  + '/' + data_name+"_test.data", sep=" ",header=None, encoding='utf8')  # The data are lo
testData=testData.dropna(axis=1,how="any")
#vtestData  # the standard output dataframe
```

Étape 2 : on fait ces lignes de code ce qui nous ont pour de remplir la premier ligne **Train** de ***table 2***

#Importation du model
#création du model
#entrainement du model
#prédiction avec le model
#affichage du score d'« accuracy »sur la sortie standard

```
from sklearn.module_containing_your_model_of_choice import your_model_of_choice
clf = your_model_of_choice()
clf.fit(trainData, trainLabels)
labelsPred = clf.predict(validData);
acc = accuracy_score(validLabels,labelsPred)
acc
```

Nearest Neighbor

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
clf = KNeighborsClassifier(n_neighbors=1,algorithm='brute')
clf.fit(trainData, trainLabels)
labelsPred = clf.predict(validData);
acc = accuracy_score(validLabels,labelsPred)
acc
```

```
/home/tochange/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  """
```

Out[12]: 0.88

Etape 3 : on utilisation le cross validation pour remplir la ligne **cv** de la ***table 2***

Cv : il s'agit de diviser les donné en k parts égaux et utiliser chaque donné tour à tour comme trainningData puis comme test Date puis faire une moyenne.

#importation du model
#importation du cross_val_score du sklearn.module model_selection
#appelle du 10-fols cross-validation sur your_model_of_choice (cv=10)
from sklearn import your_model_of_choice
from sklearn.model_selection import cross_val_score
clf = tree.DecisionTreeClassifier()
print (cross_val_score(clf,trainData,trainLabels,cv=10, scoring="accuracy").mean())

Nearest Neighbor

In [13]:
```
#10-fols cross-validation with DecisionTree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
clf = KNeighborsClassifier(n_neighbors=1,algorithm='brute')
print (cross_val_score(clf,trainData,trainLabels,cv=10, scoring="accuracy").mean())
```

```
0.8055555555555556
```

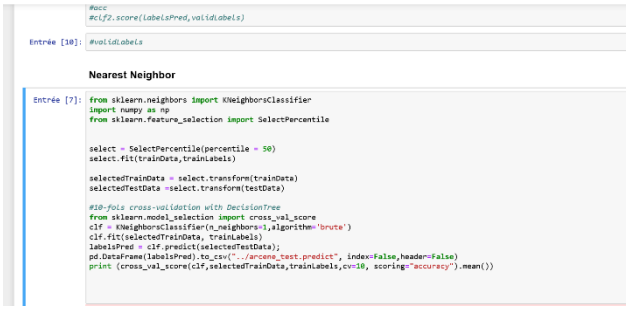
FEATURES SELECTION

Il existe de nombreuses méthodes de sélection de caracteristiques et la plupart sont disponible dans la bibliothèque de scikit learn : VarianceThreshold, SelectKBest, select percentile…

Dans ce challenge il s'agit d'en tester beaucoup et d'en choisir le plus adapté pour chaque donnée.

DESCRIPTION DES ALGORITHMES ETUDIE

#importation sélection de caracteristiques
#création de la sélection de caracteristiques (Percentile=50ne conserver que 50% des caractéristiques)
#entrainement de la sélection de caracteristiques
#création de dataframe sans les caractéristiques inutiles



```
#acc
#clf2.score(labelsPred,validLabels)

Entrée [10]: #validLabels

Nearest Neighbor

Entrée [7]: from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.feature_selection import SelectPercentile


select = SelectPercentile(percentile = 50)
select.fit(trainData,trainLabels)

selectedTrainData = select.transform(trainData)
selectedTestData =select.transform(testData)

#10-fois cross-validation with DecisionTree
from sklearn.model_selection import cross_val_score
clf = KNeighborsClassifier(n_neighbors=1,algorithm='brute')
clf.fit(selectedTrainData, trainLabels)
labelsPred = clf.predict(selectedTestData);
pd.DataFrame(labelsPred).to_csv("../arcene_test.predict", index=False,header=False)
print (cross_val_score(clf,selectedTrainData,trainLabels,cv=10, scoring="accuracy").mean())
```

```
from sklearn.feature_selection import SelectPercentile
select = SelectPercentile(percentile = 50)
select.fit(trainData,trainLabels)
selectedTrainData = select.transform(trainData)
```

DIFFICULTES RENCONTREES

Le challenge en elle-même n'était pas trop compliqué maintenant que j'ai dû recule. Ceci dit je trouve tout de même qu'il y a pas mal de subtilité. Premièrement les informations et les consigné ils étaient éparpillés il fallait bien allez les chercher par tout sans quoi on pouvait faire un hors sujet.

L'autre difficulté c'est l'importation des données en tant que dataFrame j'ai passé plus de la moitié du temps que j'ai passé sur ce projet à essayer d'importer les données sur DEXTER et DOROTHEA et enfin de compte je n'ai pas réussi c'est pour cela que la troisième ligne des tables 2 résultats n'est pas rempli puisque je n'ai pas pu voir le score mes soumission car je n'avais pas tous les fichiers nécessaires:(

DISCUSION ET CONCLUSION

Malheureusement notre soumission n'a pas pu être considéré comme valide puisque je n'ai pas réussi à traiter tous les fichiers du fait de leurs types un peut être un peu plus complexe à traiter. C'est l'occasion de considérer d'aller voir d'autre étapes du proprecessing tel que la normalisation des données ...