Hes·so Master

MSE | MASTER OF SCIENCE IN ENGINEERING

Multimodal Processing, Recognition, and Interaction (MPRI)

Anomaly detection

–

SVM

Auteur : Spinelli Isaia
Prof : Stefano Carrino

Date : 03.11.2020
Salle : A2 – Lausanne
Classe : MPRI

Table des matières

Introduction

Dans ce TP, nous utiliserons le classifieur mono-classe OneClassSVM de scikit-learn. Nous allons utiliser un vrai set de données afin de faire de la détection de Novelty.

OneClassSVM avec Scikit-learn

1. Comment les échantillons sont générés ? Pourquoi y a-t-il deux pôles de concentration de points ?

Pour générer les échantillons, on génère des échantillons de type float à partir d'une distribution univariée « normale » (gaussienne) de moyenne 0 et de variance 1 en utilisant la fonction « numpy.random.rand » avec un sigma de 0.3. Ensuite, on ajoute ± 2 à toutes les valeurs des échantillons, ce qui double le nombre de donnés. C'est aussi ceci qui va provoquer les deux différents pôles de concentration de points.

Pour générer les échantillons outliers, il génère 40 échantillons de manière uniforme entre [-4 ; 4 [. (4 non compris).

2. Dans le code ci-dessous, à quoi correspondent les variables X_train, X_test et X_outliers. Pourquoi n'a-t-on pas uniquement le training set et le test set ?

```
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
```

La variables X_tain correspond aux données d'entrainement (training obersvations). La variable X_test correspond aux données de test (new regular observations). La variable X_outliers correspond aux données « outliers », autrement dit, les données distante des autres observations (new abnormal observation).

Nous avons maintenant un outliers set afin de vérifier la bonne détection des anomalies.

3. Pourquoi ne donne-t-on pas les étiquettes (ou labels) à la fonction fit() d'apprentissage ?

Car toutes les données dans le set d'entrainement sont de la même classe, des cancers bénins.

4. Tester d'autres noyaux (kernel) pour le classifieur et garder celui qui donne les meilleurs scores. Modifier ensuite les différents paramètres afin d'améliorer les prédictions (gamma, nu, et d'autres en fonction du noyau choisi - voir documentation).

Dans la documentation, voici les différents kernel possibles : linear, poly, rbf, sigmoid, precomputed. Dans mon cas, precomputed ne compile pas. Sinon, les meilleurs scores ont été produit par le kernel par défaut rbf.

Après plusieurs tentatives, je me suis rendu compte que ce n'était pas évident de rechercher les meilleurs paramètres car les données sont générées aléatoirement et cela peut faire varier notre système facilement. Personnellement, je trouvé que ces paramètres apportaient les meilleurs résultats de manière générale : nu=0.001 et gamma=0.2.

Novelty detection

Tâche 1

La tâche 1 consista à implémenter la fonction «extract_features » qui permet de extraire les caractéristiques souhaitaient. Premièrement, deux colonnes peuvent rapidement être supprimer ; « sample code number » et « Class ».

Sample Code number n'aide en rien la prédication et Class contient la « solution ». Ensuite, j'ai analysé les différentes caractéristiques lancé quelques fois l'algorithme avec et sans quelques caractéristique. Je me suis rendu compte que les caractéristiques « Normal Nucleoli » et « Bland chromatin » permettait une bonne prédiction. De ce fait, j'ai décidé de garder uniquement ces deux caractéristiques, ci-dessous le code permettant de faire le filtre :

```
feat = data.iloc[:, [7,8]]
```

Remarque : Il serait préférable de filtrer ces colonnes en utilisant leurs noms au lieu de leurs indexes.

Tâche 2

La tâche 2 a pour but de séparer le dataset en deux parties selon la classe. En effet, nous voulons faire de la "novelty detection", et donc le set d'entraînement ne doit pas contenir d'exception. Voici le code qui m'a permis de faire ceci :

```
benign_set = df.loc[df['Class'] == 2]
outliers = df.loc[df['Class'] == 4]
```

Tâche 3

L'objectif de la tâche 3 est d'optimiser le nombre d'échantillons bénins détectés comme tel, tout en optimisant le nombre d'échantillons malins détectés comme tel. Pour ce faire, j'ai commencé par tester les différents kernels possible. Le kernel rbf offrait de meilleurs résultats que les autres kernels. Ensuite j'ai testé de modifier quelques paramètres afin d'affiner les résultats. Pour finir, les meilleurs résultats observés étaient avec les paramètres par défaut et le paramètre nu à 0.1. Voici donc la création du classificateur :

```
clf = svm.OneClassSVM(nu=0.1, kernel="rbf")
```

Et voici les résultats obtenus :

```
--> ERRORS
Errors on training data : 23 / 306
Errors on regular data  : 11 / 152
Errors on outliers  : 11 / 241
-----------------------------------------
--> RESULTS: TEST SET
Accuracy    : 0.944
Precision   : 0.928
Recall      : 0.928
F1-score    : 0.928
-----------------------------------------
--- THE END ---
```

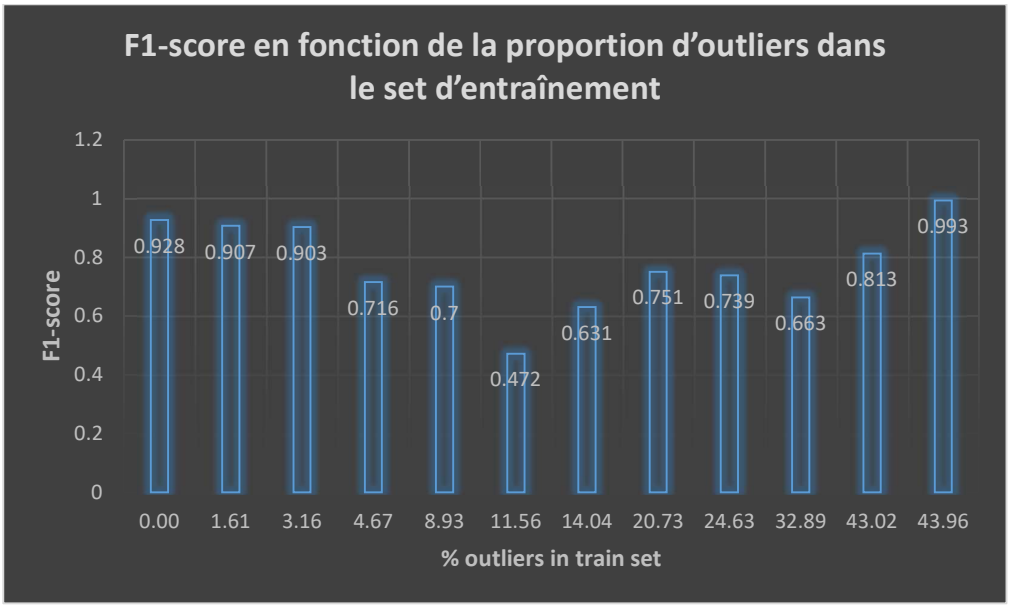
Nous pouvons voir que les résultats offerts ne sont pas parfaits mais assez bons avec une précision de ~93%.

Outliers detection

Jusqu'à présent, nous avons entrainé le modèle uniquement avec des échantillons bénins. Dans un cas pratique réel, il se peut que des échantillons malins se glissent dans le set d'entrainement. C'est ce que nous allons simuler dans cet exercice.

Notre but est d'ajouter de plus en plus d'outliers dans le set d'entraînement et de voir l'évolution du score de classification

Voici un graphique qui permet de visualiser l'évolution du F1-score en fonction de la proportion d'outliers dans le set d'entraînement :



Le nombre d'outliers varie entre 0 (0%) et 240 (44%) dans le set d'entrainement. On peut voir que le F1-score commence par diminuer lorsqu'on augmente la proportion d'outliers dans le set

d'entrainement. Cependant, à partir de ~12%, le F1-score raugmente. Finalement, en mettant tous les outliers sauf 1 dans le set d'entrainement, on obtient un très bon score. Je ne suis pas sûr mais peut-être que cela s'explique car le set d'entrainement est bien plus complet et l'algorithme arrive bien à faire la différence entre un cancer bénins et malins. Les outliers sont devenus une classe à part entière.

Conclusion

Difficultés rencontrées

- Bonne compréhension des principes et conséquences

Compétences acquises

- Familiarisation avec le classifieur « OneClassSVM »
- Amélioration de la maîtrise du langage python
- Les différents paramètre de l'algorithme SVM

Résultats obtenus

Toutes les étapes demandées du laboratoire ont été réalisées et les questions répondues. J'ai trouvé ce laboratoire particulièrement instructif car il était bien dirigé, ce qui m'a aidé à le réaliser pas à pas. De plus, le fait de travailler avec des données réelles est particulièrement intéressant.

Annexe

- Tableau et graphique du chapitre « Outliers detection ».
- Code Pyhton de l'exercice 2 (ex2-breast-cancer.py)
- Code Pyhton de l'exercice 3 (ex3-breast-cancer_outliers.py)

Date : 04.11.20

Nom de l'étudiant : Spinelli Isaia

f1-score	% outliers	Nb outliers in train	train data	outliers
0.928	0.00	0	306	241
0.907	1.61	5	311	236
0.903	3.16	10	316	231
0.716	4.67	15	321	226
0.7	8.93	30	336	211
0.472	11.56	40	346	201
0.631	14.04	50	356	191
0.751	20.73	80	386	161
0.739	24.63	100	406	141
0.663	32.89	150	456	91
0.813	43.02	231	537	10
0.993	43.96	240	546	1