Etude Pratique (5) – Master Informatique – IAA – 2021
Université d'Aix-Marseille

Cécile Capponi – QARMA, LIS – AMU
Cecile.Capponi@lis-lab.fr

25 mars 2021

Objectifs de la séance :
- comprendre l'impact des jeux de données déséquilibrés
- pratiquer une expérimentation rigoureuse de sélection de modèles : chaîne de traitement

Déséquilibre en classification et Challenge

1 Classification à partir de jeux de données avec déséquilibres

Les jeux de données réelles sont souvent déséquilibrés : certaines classes sont plus représentées que d'autres dans l'échantillon disponible. Nous allons étudier l'impact de ce déséquilibre sur l'échantillon. Pour cela, nous allons utiliser dans un premier temps un jeu de données artificiellement généré pour comprendre l'impact du déséquilibre sur les modèles appris. Puis dans un second temps, nous étudierons le jeu de données `breast_cancer` pour développer les deux modes basiques de traitement de ce genre de problème.

1.1 Jeu de données artificiellement généré

Le programme suivant permet de créer un jeu de données déséquilibré et de lancer des apprentissages de classifieurs à l'aide de quatre méthodes (`basic_imbalance.py`).

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.dummy import DummyClassifier

X,Y = make_classification(n_samples=150, n_features=20, n_informative=10,
                                n_redundant=5, n_repeated=5, n_classes=2,
                                n_clusters_per_class=2, weights=[0.9, 0.1],
                                flip_y=0.01, class_sep=1.5, hypercube=True)
cnames=["M","m"]

X_app,X_test,Y_app,Y_test=train_test_split(X,Y,test_size=0.30,random_state=12)

print('********** Rapport avec déséquilibre **********')
clf_nb = DummyClassifier(strategy="stratified")
clf_nb.fit(X_app,Y_app)
```

```
y_pred_nb = clf_nb.predict(X_test)
print('Maj:',classification_report(Y_test, y_pred_nb, target_names=cnames))

clf_nb = GaussianNB()
clf_nb.fit(X_app,Y_app)
y_pred_nb = clf_nb.predict(X_test)
print('NB: ',classification_report(Y_test, y_pred_nb, target_names=cnames))

clf_dt = DecisionTreeClassifier()
clf_dt.fit(X_app,Y_app)
y_pred_nb = clf_dt.predict(X_test)
print('DT: ',classification_report(Y_test, y_pred_nb, target_names=cnames))

clf_kppv = KNeighborsClassifier()
clf_kppv.fit(X_app,Y_app)
y_pred_nb = clf_kppv.predict(X_test)
print('KP: ',classification_report(Y_test, y_pred_nb, target_names=cnames))
```

1. Comprendre, puis exécuter ce programme, et observer le résultat : les différentes informations données par le rapport de classification doivent notamment être expliquées.
2. Comment ces résultats évoluent-ils quand le déséquilibre diminue (faire varier le paramètre `weights`) ?

1.2 Sur de vraies données

Le jeu de données `breast_cancer` comporte 2 classes, avec 212 exemples de cancers malins et 357 exemples de cancers bénins : ce n'est pas un fort déséquilibre mais il suffira à illustrer les techniques basiques de traitement du déséquilibre. Les exemples sont décrits à l'aide de 30 attributs.

1. Modifier le programme précédent pour que les rapports de classification soient effectués sur le jeu de données `breast_cancer` disponible sous `sklearn` dans le package `datasets`.
2. Modifier le programme pour ajouter des rapports concernant les performances de classification de ces quatre méthodes, mais avec un échantillon de base qui résulte du sous-échantillonnage de l'initial : l'idée est de créer un nouvel échantillon qui contient le même nombre d'exemples pour chaque classe, en enlevant aléatoirement des exemples de la classe majoritaire (technique dite de *sub-sampling*). Expliquez la différence de comportement des apprentissages à travers les rapports de classification.
3. Même question que la précédente, mais en appliquant la technique de l'*over-sampling* : nous allons artificiellement ajouter des exemples de la classe minoritaire dans le jeu de données, selon deux techniques :
   — dupliquer autant d'exemples de la classe minoritaire (pour cela, un random avec remise est de mise)
   — ajouter des exemples dans le jeu de données qui résultent du bruitage très léger d'exemples de la classe minoritaire. Soit z le nombre d'exemples de la classe minoritaire à rajouter dans le jeu de données pour atteindre l'équilibre des classes. Nous allons appliquer un *bruit gaussien* comme vu en cours : sur chaque composante j de l'espace de description, on applique le bruit sur z exemples de la classe minoritaire, tirés aléatoirement avec remise ; pour chacun de ces z exemples, notons le x_i, on le copie pour créer l'exemple $x_{i'}$, avec $y_{i'} = y_i$ et $x_{i',j} = x_{i,j} + \iota_j$ avec ι_j tiré aléatoirement selon la distribution $\mathcal{N}(0, \sigma_j/\mu_j)$ où μ_j est la moyenne des valeurs des $x_{\_,j}$ et σ_j son écart-type.

   Pour chacune de ces deux techniques, évaluer son impact sur les rapports de classification, notamment en termes de rappel, de précision, et de F1-mesure. L'observation des matrices de confusion peut aider à visualiser le phénomène.

2 Challenge sur données réelles (Kaggle), pour pratiquer

Le jeu de données Titanic contient, pour plusieurs centaines de passagers, un certain nombre d'informations (âge, genre, adresse, port d'embarquement, nombre d'accompagnants, etc.). Une colonne contient la cible : est-ce que le passager en question a survécu ou non. La particularité de ce jeu de données est que les colonnes sont de types tous différents, pas forcément numériques, et qu'il faut donc au préalable pré-traiter ces données avant de pouvoir apprendre à prédire la survie ou la mort d'un passager du Titanic.
La bibliothèque `pandas` est très utile pour cela, en lien avec `sklearn`, car permet une manipulation souple des lignes (exemples) et colonnes (attributs). Nous n'illustrons ici qu'une petite partie de la notion de pre-traitement des données, car les possibilités sont extrêmement nombreuses.
Le code ci-après permet de charger le jeu de données sous un `DataFrame` (gestionnaire avancé de données structurées) de pandas :

```
import pandas as pd
df = pd.read_csv('titanic.csv')
```

Ensuite, l'accès aux colonnes se fait via le nom de l'attribut, par exemple `Y = df['Survived']` permet de récupérer dans un tableau numpy `Y` le vecteur des classes des exemples. Nous pouvons éliminer des colonnes (axe 1 du data frame), ou des lignes (axe 0) avec la fonction `df.drop()`. Ceci permet notamment d'éliminer purement et simplement des attributs que l'on estime non pertinents pour le modèle à apprendre :

```
df.drop(['Survived', 'Cabin'], axis=1)
```

Il est conseillé de vider le data frame des colonnes inutiles, notamment de la colonne de classe : il est alors utilisable comme échantillon de description à fournir à un apprenant (le `X` paramètre de la fonction `fit`).

- Consultez avec l'outil de votre choix la nature du jeu de données Titanic. Repérez les informations qui vous sembleraient inutiles. Vous remarquerez la mention NaN (not a number) qui signifie en réalité que la donnée est manquante. Il existe aussi des attributs de type chaîne qui ne prennent que deux ou trois valeurs différentes : dans ce cas, les encoder comme entier est plus pertinent.
- Ecrivez un programme qui lit le fichier Titanic dans un data frame (pandas), et qui utilise la classe `sklearn.preprocessing.LabelEncoder()` pour que le genre devienne 0 ou 1.
- Il faudrait faire de même avec la colonne `'Embarked'`, mais vous pouvez observer qu'elle contient des valeurs manquantes non numériques : il faut préalablement compléter cette colonne. Le code ci-après vous indique comment remplacer chaque valeur manquante par la valeur la plus présente. C'est une stratégie parmi d'autres (remplacement aléatoire, ou remplacement par la valeur la moins présente, remplacement condtionnellement à une ou plusieurs autres colonnes, etc.).
  ```
  nb_manquantes = len(df['Embarked'][df.Embarked.isnull()])
  val_remplacement = df['Embarked'].dropna().mode().values
  df['Embarked'][df['Embarked'].isnull()] = val_remplacement
  ```
  Ajoutez ce traitement dans votre programme, et observez son effet sur les données.
- La colonne `'Embarked'`, tout comme celle du genre, ne comprend finalement que 3 valeurs possibles : des chaînes. Ajoutez à votre programme le même traitement pour coder en entier ces valeurs.
- Complétez votre programme pour que les targets du classifieur soient celles de la colonne `'Survived'`.
- Eliminez dans votre programme toutes les colonnes qui vous semblent inutiles pour l'objectif de prédiction [1].
- S'il reste des colonnes avec des informations manquantes, vous pouvez utilisez dans votre programme la fonction de complétion des données manquantes de `sklearn`, disponible via la classe `Imputer` [2] ; elle s'adapte à toutes sortes de types de données basiques :
  ```
  from sklearn.impute import SimpleImputer
  remplir = SimpleImputer() # adapter le parametrage,  cf la doc sklearn
  df = remplir.fit_transform(df)
  ```

1. En réalité, il existe des approches de sélection automatique d'attributs – que vous pratiquerez en M2
2. https://scikit-learn.org/stable/auto_examples/impute/plot_missing_values.html#sphx-glr-auto-examples-impute-plot-missing-values-py

Vous pouvez, à la place, choisir d'autres méthodes de `sklearn.impute`, par exemple le magnifique `KNNImputer`.

— Votre jeu de données est maintenant prêt. Choisissez deux algorithmes d'apprentissage de votre choix, et mener les expérimentations nécessaires pour prédire le meilleur modèle avec intervalles de confiance : la validation croisée pour estimer le risque réel de chaque classifieur sera préférée.

— *Test de McNemar.* On définit 4 variables :
  — $n00$: le nombre d'exemples de l'échantillon test que les deux classifieurs classent incorrectement,
  — $n11$: le nombre d'exemples que les deux classifieurs classent correctement,
  — $n10$ (resp. $n01$) : le nombre d'exemples que le premier classifieur classe correctement et le second incorrectement (resp. : l'inverse).

  Si
$$\frac{(|n01 - n10| - 1)^2}{n01 + n10} > 3.841459$$
  on peut en déduire avec une confiance supérieure à 95% que l'un des deux classifieurs est meilleur que l'autre. Estimez ces quantités en moyennant sur 10 train-test-split ? Est-ce qu'un des deux classifieurs est meilleur que l'autre selon le test de McNemar, avec 95% de confiance ?

— Vous pouvez vous inscrire au challenge `Kaggle` de ce jeu de données et y évaluer votre rang parmi des milliers de scientifiques des données !