TP N° 6
Apprentissage unsupervisé : réduction de la dimension
Détection d'anomalie: étude de cas de détection de fraude bancaire

Contexte

Nous proposons dans ce TP une **approche non supervisée** pour détecter des fraudes dans des transaction bancaires. Nous précisions que les labels seraient utilisés que pour évaluer l'algorithme. L'un des plus grands défis de ce problème est que la cible est fortement déséquilibrée. Mais l'avantage de l'approche d'**apprentissage par représentation** est qu'elle est toujours capable de gérer la nature déséquilibrée des problèmes. En utilisant **tsne**, nous pouvons essayer de voir comment les transactions sont similaires.

L'idée principale de cette approche est de *compresser* les données en faisant une **représentation latente**, puis de les reconstruire, i.e., si les données reconstruites ne sont pas similaires aux données originales, nous avons une fraude.

1 Dataset

L'ensemble de données que nous utiliserons contient des transactions bancaires effectuées en septembre 2013 par des titulaires de cartes européennes. L'ensemble de données a été collecté et analysé lors d'une collaboration de recherche entre Worldline et le Groupe de Machine Learning de l'ULB (Université Libre de Bruxelles) sur le big data mining et la détection des fraudes.

Ces transactions sont un sous échantillon de toutes les transactions en ligne qui ont eu lieu en deux jours, où nous avons 492 fraudes sur 284,807 transactions. L'ensemble de données est fortement déséquilibré, la classe positive (fraude) représentant 0,172% de toutes les transactions. Il ne contient que des variables d'entrée numériques qui sont le résultat d'une transformation ACP. Malheureusement, pour des raisons de confidentialité, nous ne pouvons pas fournir les caractéristiques originales et plus d'informations sur les données. Les variables $V_1, V_2, ..., V_{28}$ sont les composantes principales obtenues par une ACP, les seules variables qui n'ont pas été transformées sont `Time` (temps entre les transactions en secondes) et `Amount`. La variable `Time` contient les secondes écoulées entre chaque transaction et la première transaction de l'ensemble de données. La variable `Amount` est le montant de la transaction (combien d'argent a été transféré dans cette transaction), cette variable peut être utilisée pour l'apprentissage sensible aux coûts dépendant de l'exemple. La variable `Class` est la variable de réponse et prend la valeur 1 en cas de fraude et 0 sinon.

2 Partie I : Analyse exploratoire des données

2.1 Importation des librairies et chargement des données

1. Importer toutes les librairies Python nécessaires pour le développement de vos modèles de prédiction.
2. Charger les données et les enregistrer dans une structure de données dataframe de Pandas.
3. Décrire succinctement les variables (on pourra lire et afficher le texte de la description à l'aide de la méthode `open`.
4. Quelles sont les dimensions de votre dataframe ?
5. Indiquer le type de chacune des variables.

2.2 Analyse univariée

1. Représenter la variable `Class` à l'aide de l'histogramme de comptage. Interpréter.
2. Tracer les distributions de variables. Interpréter.
3. La variable `Time` est donnée en secondes. Changer cette variable de telle sorte qu'elle nous indique à quelle heure la transaction était effectuée.

4. Tracer la distribution de la variable `Amount` sachant que les transactions soient fraudes ou normales. Interpréter.
5. Tracer la distribution de la variable `Time` sachant que les transactions soient fraudes ou normales. Interpréter.
6. Afficher la matrice de corrélation pour les variables numériques. Interpréter.
7. Visualiser les distributions de certaines variables V_i pour un certain $i \in \{1, \ldots, 28\}$, pour les transactions fraudes et normales.

3 Partie II : Pré-traitement de données

1. Écrire une fonction `preprocessing` qui permet de standariser les variables : `Time` et `Amount` et retourner un une matrice de design X et un vecteur de labels y.
2. Séparer en données en `df_train` et `df_test`.
3. Appliquer la fonction `preprocessing` à `df_train` et `df_test`.

4 Partie III : Modélisation + Évaluation

4.1 Score d'anomalie : fonction calculant l'erreur de reconstruction

Les algorithmes de réduction de la dimension réduisent la dimension des données tout en essayant de minimiser l'erreur de reconstruction. En d'autres termes, ces algorithmes essaient de capturer les informations les plus importantes des variables originales de manière à pouvoir reconstruire l'ensemble des ces variables d'origine à partir de l'ensemble réduit de variables possibles.

Nous allons définir une fonction qui calcule le score d'anomalie de chaque transaction. Plus la transaction est anormale, plus il est probable qu'elle soit frauduleuse, en supposant que la fraude est rare et a une apparence quelque peu différente de la la majorité des transactions, qui sont normales.

[ ]:
```python
# Calculer l'erreur de reconstruction
def anomaly_scores(x_original, x_reduced):
    loss = np.sum((np.array(x_original) - np.array(x_reduced))**2, axis=1)
    loss = pd.Series(data=loss, index=x_original.index)
    loss = (loss - np.min(loss)) / (np.max(loss) - np.min(loss)) # loss entre 0. et 1.
    return loss
```

La fonction `anomaly_scores` définit le score d'anomalie comme l'erreur de reconstruction pour chaque transaction $\vec{x}_i \in \mathbb{R}^{30}$ comme la somme des différences au carré entre la transaction d'origine $\vec{x}_i$ et la transaction reconstruite $\widehat{\vec{x}}_i \in \mathbb{R}^{30}$ à l'aide de l'algorithme de réduction de la dimension. Nous allons mettre à l'échelle la somme des différences au carré par la quantité maximale-minimale de la somme des différences au carré pour l'ensemble de données, de sorte que toutes les erreurs de reconstruction soient comprises entre 0 et 1.

Les transactions qui présentent la plus grande somme de différences au carré auront une erreur proche de 1 (le plus susceptible d'être frauduleux), tandis que celles qui présentent la plus petite somme de différences au carré auront une erreur proche de 0.

4.2 Métriques d'évaluation

4.2.1 Cas des données fortement déséquilibrées

Les données des transactions sont fortement déséquilibrées dans lesquels le nombre de négatifs (transactions normales) est largement supérieur au nombre de positifs (transactions frauduleuses). Bien que les courbes ROC soient visuellement attrayantes et fournissent une vue d'ensemble des performances d'un classifieur sur une large gamme de spécificités, on peut se demander si les courbes ROC ne sont pas faussées lorsqu'elles sont appliquées à des scénarios de classification déséquilibrée.

Courbe et aire Précision/Rappel (Precision Recall Curve (PRC)) et AU-PRC. Pour notre jeu de données déséquilibré sur les transactions, une meilleure façon d'évaluer les résultats est d'utiliser la précision et le rappel (. En effet :

- Une précision élevée signifie que parmi toutes nos prédictions positives, beaucoup sont de vrais positifs (autrement dit, le taux de faux positifs est faible).
- Un rappel élevé signifie que le modèle a capturé la plupart des vrais positifs (en d'autres termes, il a un faible taux de faux négatifs).
- Une solution avec un rappel élevé mais une précision faible renvoie de nombreux résultats capturant de nombreux positifs, mais avec de nombreuses fausses alarmes.
- Une solution avec une précision élevée mais un rappel faible est exactement l'inverse ; elle renvoie peu de résultats - elle capture une fraction de tous les positifs de l'ensemble de données, mais la plupart de ses prédictions sont correctes.

Résumé. Si la solution avait une grande précision mais un faible rappel, un très petit nombre de transactions frauduleuses seraient découvertes, mais la plupart seraient réellement frauduleuses.

En revanche, si la solution avait une faible précision mais un rappel élevé, elle signalerait de nombreuses transactions comme étant frauduleuses, ce qui permettrait de détecter une grande partie de la fraude, mais la plupart des transactions signalées ne seraient pas frauduleuses.

De toute évidence, les deux solutions présentent des problèmes majeurs. Dans le cas d'une précision élevée et d'un rappel faible, la société de cartes de crédit perdrait beaucoup d'argent à cause de la fraude, mais elle ne se mettrait pas les clients à dos en rejetant inutilement des transactions. Dans le cas d'une précision faible et d'un rappel élevé, la société de cartes de crédit détecterait une grande partie de la fraude, mais elle mettrait très certainement les clients en colère en rejetant inutilement un grand nombre de transactions normales et non frauduleuses.

Il existe généralement un compromis entre la précision et le rappel, qui est habituellement déterminé par le seuil fixé par l'algorithme pour séparer les cas positifs des cas négatifs ; dans notre exemple, un cas positif est une fraude et un cas négatif n'est pas une fraude. Si le seuil est trop élevé, très peu de cas sont prédits comme positifs, ce qui entraîne une précision élevée mais un rappel faible. Lorsque le seuil est abaissé, davantage de cas sont prédits comme positifs, ce qui diminue généralement la précision et augmente le rappel.

Choix du seuil. Le choix du seuil est très important et implique généralement la participation des décisionnaires de l'entreprise. Les spécialistes des données peuvent présenter la courbe précision-rappel à ces décideurs pour déterminer le seuil à retenir.

Solution optimale est de maximiser en même temps la précision et le recall.

Courbe Précision/Rappel. Dans notre cas des transactions, nous pouvons trouver le modèle qui présente le meilleur rapport précision/rappel. Pour cela, nous utilisons le graphique du compromis entre la précision et le rappel est connu sous le nom de *courbe précision-rappel*. Pour évaluer la courbe précision-rappel, nous pouvons calculer la précision moyenne, qui est la moyenne pondérée de la précision obtenue à chaque seuil. Plus la précision moyenne est élevée, meilleure est la solution.

Les courbe PRC peuvent fournir à l'observateur une prédiction précise des performances de classification futures, car ils évaluent la fraction de vrais positifs parmi les prédictions positives. L'AUC-PRC : aire normalisée sous la courbe paramétrique définie par la Précision et le Rappel en fonction du seuil de décision.

Ci-dessous quelques liens pour plus de détails sur les courbes PRC :

- https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432
- Lien de scikit-learn pour la documentation de cette méthode : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html

4.2.2 Fonctions de visualition des résultats des courbes PRC et l'aire AU-PRC

[ ]:
```
def plotResults(labels, anomaly_scores, returnPreds=False):
    df_preds = pd.concat([labels, anomaly_scores], axis=1)
    df_preds.columns = ['Labels', 'AnomalyScores']

    ############################
    ## Courbe PRC et aire AU-PRC
    ############################
    precision, recall, thresholds = precision_recall_curve(df_preds['Labels'],␣
 ↪df_preds['AnomalyScores'])
    average_precision = average_precision_score(df_preds['Labels'],␣
 ↪df_preds['AnomalyScores'])

    plt.step(recall, precision, color='k', alpha=0.7, where='post')
    plt.fill_between(recall, precision, step='post', alpha=0.3, color='k')

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])

    plt.title('Courbe Precision-Recall: Moyenne de précision = \
    {0:0.2f}'.format(average_precision))

    #############
    ## Courbe ROC
    #############
    fpr, tpr, thresholds = roc_curve(df_preds['Labels'], df_preds['AnomalyScores'])
    areaUnderROC = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='r', lw=2, label='Courbe ROC')
    plt.plot([0, 1], [0, 1], color='k', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Taux de faux positifs (FPR)')
    plt.ylabel('Taux de vrais positifs (TPR)')
    plt.title('Courbe ROC: Aire sous la courbe ROC = {0:0.2f}'.format(areaUnderROC))
    plt.legend(loc="lower right")
    plt.show()

    if returnPreds==True:
        return df_preds
```

La fonction `scatterPlot` affiche la séparation des points que l'algorithme ACP réalise dans les deux premières composantes seulement.

[ ]:
```
# Visualiser les premières compsantes principales de l'ACP
def scatterPlot(X, y):
    temp = pd.DataFrame(data=X.loc[:,0:1], index=X.index)
    temp = pd.concat((temp,y), axis=1, join="inner")
    temp.columns = ["PC1", "PC2", "Label"]
    sns.lmplot(x="PC1", y="PC2", hue="Label", data=temp, fit_reg=False)
    ax = plt.gca()
```

5 ACP : Analyse en composante principale

5.1 Composantes PCA égale au nombre original de la dimension

1. Modéliser une ACP avec nombre de composantes égale au nombre original de la dimension : `n_components=30`.
2. Calculer les scores de détection.
3. Calculer les courbe de précision-recall et afficher la précision moyenne.
4. Interpréter.
5. Changer le nombre de composantes de l'ACP et reprendre les questions 2, 3, 4.