Rapport Intelligence artificielle L3S6 Informatique

HOCHBERGER Dylan

REPONSES

Observations

Types d'erreurs détectables :

- 1. On souhaite prédire l'attribut `target` .
- 2. Il s'agit d'une classification binaire puisque Target prend les valeurs 0 et 1).
- 3. Attributs catégoriels :
  - `sex`
  - `chest_pain_type`
  - `fasting_blood_sugar`
  - `rest_ecg`
  - `exercise_induced_angina`
  - `st_slope`
  - `thalassemia`
  - `target`
  - `num_major_blood_vessels` : Pas sûr car semble calculé mais géré comme un attribut catégoriel
- 4. Les attributs catégoriels ordinaux peuvent être gérés en créant des catégories. Les attributs catégoriels nominaux peuvent être traduits sous forme de valeurs ou matrices.

Normalisation

- 1. Standardisation
- 2. Après séparation des données
- 3. Les attributs non catégoriels donc 0,3,4,7,9
- 4. Non testé car codé en Python

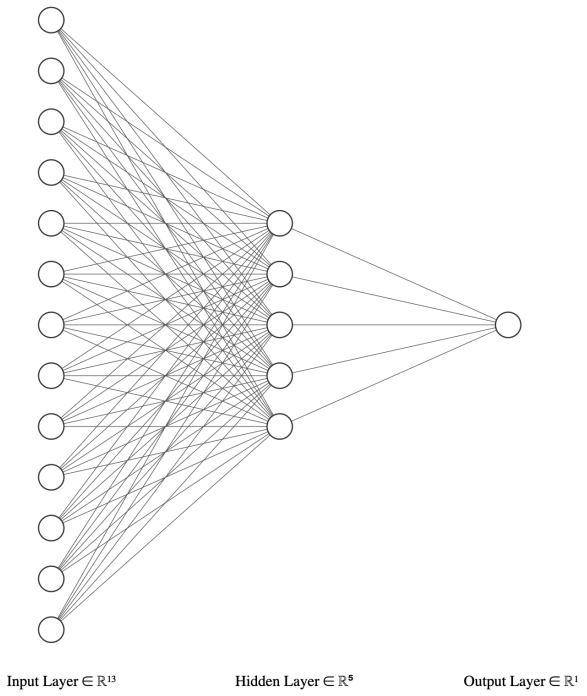
```
 public void normalize(float[][] data, int[] indices){
float[] liste_moyenne = new float[14];
for(int i=0; i<14; i++){
   for (int j=0;j<indices.length;j++){
     if (i == indices[j]){
       moyenne = 0;
       ecart_type = 0;
       for (int k = 0; k<data.length; k++){
         moyenne += data[k][i];
       }
       moyenne = moyenne/data.length;
       for (int k = 0; k<data.length; k++){
         ecart_type += (data[k][i]-moyenne)**2;
       }
       ecart_type = Math.sqrt(ecart_type/data.length);
       for (int k = 0; k<data.length; k++){
         data[k][i] = (data[k][i]-moyenne)/ecart_type;
       }
     }
   }
}
```

En python :

```
def standardize(self, list, indices):
  for i in range(len(list[0])) :
    for k in range(len(indices)):
      if i+1==indices[k] :
        moyenne = 0
        ecart_type = 0
        for j in range(len(list)):
          moyenne += list[j][i]
        moyenne = moyenne/len(list)
        for j in range(len(list)):
          ecart_type += (list[j][i]-moyenne)**2
        ecart_type = math.sqrt(ecart_type/len(list))
        for j in range(len(list)):
          list[j][i] = (list[j][i]-moyenne)/ecart_type
```

Réseau neuronal

- 1.
  - Matrice entrées : [13,1]
  - Matrice paramètres : [5,1]
  - Matrice sorties : [1,1]
- 2.



Conclusion

Le modèle contenant 2 couches cachées ainsi que 5 neurones par couches nous propose les pires statistiques parmi les 3 modèles.
Le pourcentage d'erreur pour 100 et 200 époques atteint plus de 42%, pour 500 époques il baisse à 32%. Cependant la sensibilité reste très proche de 1, montrant que dans la majorité des cas il les estime comme positifs, que ce soit vrai ou faux.
Les performances globales sont proches de 0.5 pour 100 et 200 époques, ce qui reste très faible.
Pour 500 époque elles montent à 0.677, ce qui n'est pas vraiment plus acceptable.

Le modèle contenant 1 couche cachée ainsi que 10 neurones par couche propose des résultats plus acceptables.
Entre 18% d'erreurs pour 500 époques et 22% pour 100 époques, une sensibilité élevée qui indique que certains malades ne sont pas bien décelés.
Cependant les performances globales restent aux alentours de 0.8, qui nous indiquent

que nous pouvons tout de même faire plus confiance à ce modèle là qu'au précédent.

Enfin pour une couche cachée et 5 neurones, pour 100 époques le pourcentage d'erreurs atteint 26%, mais avoisinent les 18% pour 200 et 500 époques. Un modèle légèrement moins acceptable que celui avec 10 neurones dans la couche cachée qui propose des données plus linéaires. Le reste des données avoisinent celle du modèle précédent, les performances globales restent assez similaires. Cependant selon la seed générée par la librairie Numpy, il est possible d'attendre 10% d'erreurs, ce qui reste, à mon avis, acceptable.

Enfin la précision avoisine 80% pour les modèles à couches uniques, le modèle à double couches n'atteint que 68% de précision pour la moyenne la plus élevée. Les modèles à simple couche cachée montent à 85% de précision pour 500 époques.

Ces différents tests m'ont permis de mettre en avant le fait que rajouter des couches cachées dans mon modèle ne font qu'augmenter le nombre d'erreurs, cependant rajouter des neurones dans l'unique couche cachée permet d'obtenir des résultats suffisamment proche les uns des autres malgré le nombre réduit de périodes. D'après quelques recherches internet, mon modèle comportant 2 couches représenterait un sur-apprentissage, tandis que mon modèle comportant 10 neurones dans la couche cachée devrait montrer une régression linéaire.

Il est également possible que la taille du jeu de données ne me permet pas d'obtenir des résultats plus convenables en entraînant le modèle sur plus de données.

Pour ce projet je n'ai utilisé que la librairie Numpy afin de générer aléatoirement les matrices de poids avec la méthode `rand()` , ainsi que pour faciliter les opérations sur les matrices comme la méthode `transpose()` pour transposer et `dot()` pour les multiplier entre elles.
Pour récupérer les données j'ai lancé chaque programme 10 fois pour chaque nombre d'époque et j'ai calculé la moyenne des différentes données. La première page contient les calculs pour 500 époques, la deuxième page pour 200 époques et la troisième pour 100 époques.
Pour normaliser j'ai utilisé la méthode de standardisation, j'ai utilisé la Sigmoid comme fonction d'activation. J'ai fait ses choix après plusieurs heures passées sur internet, les autres personnes conseillaient ces choix pour des petits réseaux neuronaux, j'ai préféré suivre ces conseils.

Le programme requiert l'installation de Numpy : `pip3 install numpy` pour fonctionner.