Ecole d'Ingénieurs Denis Diderot

Module d'Intelligence Artificielle - Mini-Projet

Mise au point d'un modèle d'apprentissage machine - Cas Titanic

Christophe Denis

2024-2025

Organisation du mini-projet et rendu

Ce mini-projet est à réaliser en **deux séances de travaux pratiques**. Il s'agit d'un travail encadré qui vous guidera progressivement dans la mise en œuvre d'une analyse de données réelles, jusqu'à la construction d'un modèle d'apprentissage automatique.À l'issue de ces deux séances, chaque binôme devra rendre un **mini-rapport de 5 pages maximum**, à déposer sur la plateforme Moodle.

Contenu attendu du rapport :

1. **Introduction** Présenter brièvement le contexte du Titanic, l'objectif du projet et la démarche choisie.
2. **Exploration des données** Analyse des caractéristiques des données (variables pertinentes, valeurs manquantes, distributions).
3. **Prétraitement** Explication des choix de nettoyage et de transformation des données (imputations, encodages, normalisation éventuelle).
4. **Modélisation et évaluation** Présentation du ou des modèles testés, des résultats obtenus, et interprétation des performances (accuracy, matrice de confusion...).
5. **Conclusion et pistes d'amélioration** Retour d'expérience sur la démarche, propositions d'amélioration ou tests complémentaires éventuels.

Le rapport peut contenir des figures, tableaux, extraits de code commentés, mais il doit surtout mettre en valeur votre compréhension des étapes de l'analyse et de la modélisation.

1 Le jeu de données Titanic

Le naufrage du Titanic est l'une des catastrophes maritimes les plus célèbres de l'histoire. Le 15 avril 1912, lors de son voyage inaugural, le RMS Titanic, pourtant considéré comme "insubmersible", a sombré dans l'océan Atlantique Nord après avoir heurté un iceberg. Faute de canots de sauvetage en nombre suffisant, 1502 des 2224 passagers et membres d'équipage ont perdu la vie. Bien que le facteur chance ait certainement joué un rôle dans la survie, certains groupes de personnes semblaient avoir plus de chances de survivre que d'autres : les femmes, les enfants, ou les passagers de première classe, par exemple. Dans cette perspective, la plateforme **Kaggle** propose une compétition intitulée *Titanic : Machine Learning from Disaster*. Il s'agit d'une excellente opportunité pour les débutants en science des données de se familiariser avec les concepts fondamentaux de l'apprentissage automatique. **Objectif de la compétition :** Développer un modèle prédictif capable de répondre à la question suivante :

Quels types de passagers étaient les plus susceptibles de survivre ?

Pour cela, vous disposez d'un jeu de données contenant diverses informations sur les passagers : nom, âge, sexe, classe socio-économique, nombre de proches à bord, prix du billet, port d'embarquement, etc.

2 Charger le jeu de données Titanic avec `scikit-learn`

Une autre manière de récupérer le jeu de données Titanic est d'utiliser la fonction `fetch_openml()` fournie par la bibliothèque `scikit-learn`. Cette méthode permet d'accéder directement à des bases de données publiques disponibles sur la plateforme OpenML.

```
from sklearn.datasets import fetch_openml

# T l charger le jeu de donn es Titanic depuis OpenML
titanic = fetch_openml(name="titanic", version=1, as_frame=True
    )

# R cup rer les donn es sous forme de DataFrame
df = titanic.frame

# Aper u des premi res lignes
print(df.head())
```

— Le paramètre `as_frame=True` permet de récupérer les données directement sous forme de `pandas.DataFrame`.

— Le DataFrame obtenu contient à la fois les variables explicatives et la variable cible `survived`.

— Il est recommandé de commencer par afficher les colonnes et les types de données pour mieux comprendre la structure :

```
print(df.columns)
df.info()
```

3 Exploration des données

Question 1 : Charger les fichiers CSV d'entraînement et de test.

```
# Utiliser pd.read_csv pour charger train.csv et test.csv
# Afficher les premi res lignes avec .head()
```

Question 2 : Examiner la structure et les types de données.

```
# .info() pour voir types et valeurs manquantes
# .describe() pour les statistiques
```

Question 3 : Identifier les colonnes avec des valeurs manquantes.

```
# .isnull().sum() ou .isna().sum()
```

Question 4 : Visualiser la répartition des classes (Pclass, Sex, Survived...).

```
# Utiliser sns.countplot()
# Param tres utiles : hue=..., x=...
```

Question 5 : Visualiser l'impact de la classe sur l'âge.

```
# Utiliser sns.boxplot(x="...", y="...", data=...)
```

4 Prétraitement

Question 6 : Remplacer les valeurs manquantes d'Age par une valeur représentative.

```
# Calculer median ou moyenne
# .fillna(..., inplace=True)
```

Question 7 : Remplacer les valeurs manquantes d'Embarked.

```
# .mode()[0] pour trouver la valeur la plus fr quente
```

Question 8 : Convertir la variable Sex en variable numérique.

```
# Utiliser .map({'male': ..., 'female': ...})
```

Question 9 : Créer des colonnes indicatrices (dummies) pour Embarked.

```
# pd.get_dummies(..., prefix=..., drop_first=True)
```

Question 10 : Choisir un sous-ensemble de colonnes pertinentes pour X.

```
# Ex : Pclass, Sex, Age, SibSp, Parch, Fare, Embarked_*
```

5 Modélisation

Question 11 : Séparer X et y.

```
# y = data["Survived"]
# X = data[["...", "...", ...]]
```

Question 12 : Diviser les données en jeu d'entraînement et de test.

```
# Utiliser train_test_split(..., test_size=0.2, random_state
    =42)
```

Question 13 : Créer un modèle de régression logistique.

```
# from sklearn.linear_model import LogisticRegression
# model = LogisticRegression(...)
# model.fit(X_train, y_train)
```

6 Évaluation du modèle

Question 14 : Prédire les valeurs et afficher l'accuracy.

```
# model.predict(...)
# accuracy_score(y_test, y_pred)
```

Question 15 : Afficher une matrice de confusion.

```
# confusion_matrix(...), ConfusionMatrixDisplay(...)
```

7 Pour aller plus loin

Exercice 1 : Essayez un modèle d'arbre de décision.

```
# DecisionTreeClassifier, .fit(), .predict(), .score()
```

Exercice 2 : Essayez une forêt aléatoire (Random Forest).

```
# RandomForestClassifier(...), puis m me pipeline
```

Exercice 3 : Faites une évaluation croisée (cross-validation).

```
# cross_val_score(model, X, y, cv=5)
```

Exercice 4 : Ajoutez des variables ou supprimez-en pour voir l'impact.

```
# Essayer X avec ou sans certaines colonnes
```

[fragile]Exercice 1 : Lire un CSV **Objectif :** Charger un fichier CSV et afficher les 5 premières lignes.

```
df = pd.read_csv("data.csv")
print(df.head())
```

Explication : Lecture simple d'un fichier.

[fragile]Exercice 2 : Afficher les colonnes

```
print(df.columns)
```

Explication : Liste les noms des colonnes.

[fragile]Exercice 3 : Décrire les données

```
df.describe()
```

Explication : Statistiques de base.

[fragile]Exercice 4 : Repérer les NaN

```
df.isnull().sum()
```

Explication : Compte les valeurs manquantes.

[fragile]Exercice 5 : Remplir les NaN

```
df["Age"] = df["Age"].fillna(df["Age"].mean())
```

Explication : Remplissage par la moyenne.

[fragile]Exercice 6 : Filtrer les femmes

```
df[df["Sex"] == "female"]
```

[fragile]Exercice 7 : Créer une colonne booléenne

```
df["Jeune"] = df["Age"] < 18
```

Explication : True/False selon l'âge.

[fragile]Exercice 8 : Grouper par sexe

```
df.groupby("Sex")["Fare"].mean()
```

Explication : Moyenne par groupe.

[fragile]Exercice 9 : Trier par âge

```
df.sort_values("Age")
```

[fragile]Exercice 10 : Fusionner deux tables

```
pd.merge(df1, df2, on="id")
```

[fragile]Exercice 11 : Créer une variable Total

```
df["Total"] = df["SibSp"] + df["Parch"]
```

[fragile]Exercice 12 : Catégoriser les âges

```
bins = [0, 12, 18, 65, 100]
labels = ["enfant", "ado", "adulte", "senior"]
df["Groupe"] = pd.cut(df["Age"], bins=bins, labels=labels)
```

[fragile]Exercice 13 : Extraire initiale du nom

```
df["Initiale"] = df["Name"].apply(lambda x: x[0])
```

[fragile]Exercice 14 : Convertir une date

```
df["Date"] = pd.to_datetime(df["Date"])
df["Mois"] = df["Date"].dt.month
```

[fragile]Exercice 15 : Double condition

```
df[(df["Sex"]=="female") & (df["Age"]<30)]
```

[fragile]Exercice 16 : Valeurs uniques

```
df["Embarked"].unique()
```

[fragile]Exercice 17 : Créer des dummies

```
pd.get_dummies(df["Embarked"], prefix="Emb")
```

[fragile]Exercice 18 : Compter les catégories

```
df["Pclass"].value_counts()
```

[fragile]Exercice 19 : Appliquer une fonction

```
df["Nom_maj"] = df["Name"].apply(str.upper)
```

[fragile]Exercice 20 : Sauvegarder en CSV

```
df.to_csv("resultat.csv", index=False)
```