Fiche de révision

#IA #CIPA4

1. Généralités sur l'IA et l'Apprentissage Automatique

1.1 Définitions

- IA : Systèmes capables d'imiter des fonctions cognitives humaines (raisonnement, apprentissage, prise de décision).
- Machine Learning (ML) : Sous-domaine de l'IA où les modèles apprennent à partir de données sans programmation explicite.
- Deep Learning (DL) : ML utilisant des réseaux de neurones profonds pour traiter des données complexes (images, texte, audio).

1.2 Types d'Apprentissage

Type	Description	Exemple
Supervisé	Données étiquetées (input + output connu).	Classification de spam, régression.
Non Supervisé	Données non étiquetées, découverte de structures cachées.	Clustering, réduction de dimension.
Semi-Supervisé	Mélange de données étiquetées/non étiquetées.	Reconnaissance d'images avec peu de labels.
Par Renforcement	Agent apprend par essais/erreurs via des récompenses.	Jeux vidéo, robotique.

2. Apprentissage Supervisé

2.1 Concepts Clés

- Régression : Prédire une valeur continue (ex: prix d'une maison).
- Classification : Prédire une catégorie (ex: chat/chien).
- Métriques d'évaluation :
  - Accuracy : `(TP + TN) / (TP + TN + FP + FN)`
  - Précision : `TP / (TP + FP)`
  - Rappel (Recall) : `TP / (TP + FN)`
  - F1-Score : `2 * (Précision * Rappel) / (Précision + Rappel)`
  - Courbe ROC/AUC : Évalue la performance du classifieur à différents seuils.

2.2 Algorithmes Courants

Algorithme	Utilisation	Code Scikit-Learn (Exemple)
Régression Linéaire	Prédire une valeur continue.	`from sklearn.linear_model import LinearRegression`
SGDClassifier	Classification binaire/multi-classe.	`from sklearn.linear_model import SGDClassifier`
KNN	Classification par similarité.	`from sklearn.neighbors import KNeighborsClassifier`
Arbres de Décision	Classification/Régression non linéaire.	`from sklearn.tree import DecisionTreeClassifier`
Random Forest	Ensemble d'arbres de décision.	`from sklearn.ensemble import RandomForestClassifier`
SVM	Classification avec marge maximale.	`from sklearn.svm import SVC`

2.3 Exemples de Code (TP1 & TP2)

Régression Linéaire (TP1)

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)  # Apprentissage
y_pred = model.predict(X_test)  # Prédiction
```

Classification Binaire (TP2 - Détection du chiffre 5)

```python
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)  # y_train_5 = (y_train == 5)
y_pred = sgd_clf.predict([some_digit])  # Prédit True/False
```

Matrice de Confusion

```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
print(cm)
```

Validation Croisée (Cross-Validation)

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
print(f"Accuracy moyenne: {scores.mean():.2f}")
```

Précision/Rappel

```python
from sklearn.metrics import precision_score, recall_score
precision = precision_score(y_train_5, y_pred)
recall = recall_score(y_train_5, y_pred)
print(f"Précision: {precision:.2f}, Rappel: {recall:.2f}")
```

Standardisation des Données

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

3. Apprentissage Non Supervisé

3.1 Concepts Clés

- Clustering : Regrouper des données similaires (ex: K-Means, DBSCAN).
- Réduction de Dimension : Simplifier les données (ex: PCA, t-SNE).
- Détection d'Anomalies : Identifier des points aberrants.

3.2 Algorithmes Courants

Algorithme	Utilisation	Code Scikit-Learn
K-Means	Clustering par partitionnement.	`from sklearn.cluster import KMeans`
PCA	Réduction de dimension linéaire.	`from sklearn.decomposition import PCA`
t-SNE	Visualisation en 2D/3D.	`from sklearn.manifold import TSNE`
DBSCAN	Clustering basé sur la densité.	`from sklearn.cluster import DBSCAN`

3.3 Exemples de Code (TP3)

PCA pour Réduction de Dimension (TP3)

```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)
print(f"Variance expliquée: {pca.explained_variance_ratio_}")
```

t-SNE pour Visualisation (TP3)

```python
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_train)
```

4. Préparation des Données

4.1 Nettoyage et Prétraitement

- Gestion des valeurs manquantes :

```python
data.fillna(data.median(), inplace=True)  # Remplace par la médiane
```

- Encodage des variables catégorielles :

```python
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
data_cat_encoded = encoder.fit_transform(data_cat)
```

- Normalisation :

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)
```

4.2 Répartition Train/Test

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5. Évaluation et Optimisation

5.1 Métriques

- RMSE (Régression) :

```python
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
```

- Grid Search (Optimisation des Hyperparamètres) :

```python
from sklearn.model_selection import GridSearchCV
param_grid = {"n_estimators": [3, 10, 30], "max_features": [2, 4, 6]}
grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print(f"Meilleurs paramètres: {grid_search.best_params_}")
```

6. Cas pratiques (TP)

TP1 : Régression sur le prix des maisons

- Objectif : Prédire `median_house_value` à partir de features géographiques.
- Code clé :

```python
# Chargement des données
import pandas as pd
data = pd.read_csv("housing.csv")
# Nettoyage
data["total_bedrooms"].fillna(data["total_bedrooms"].median(), inplace=True)
# Modèle
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=30, max_features=6)
model.fit(X_train, y_train)
```

TP2 : Classement des Chiffres MNIST

- Objectif : Classifier des images de chiffres (0-9).
- Code clé :

```python
# Chargement des données
from sklearn.datasets import fetch_openml
mnist = fetch_openml("mnist_784", version=1)
X, y = mnist["data"], mnist["target"]
# Classification multi-classe
sgd_clf.fit(X_train, y_train)  # y_train = chiffres 0-9
```

TP3 : Réduction de Dimension avec PCA / t-SNE

- Objectif : Visualiser MNIST en 2D.
- Code clé :

```python
# PCA
pca = PCA(n_components=0.95)  # Conserver 95% de la variance
X_pca = pca.fit_transform(X_train)
# t-SNE
```

```
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X_train)
```

7. Résumé des Bonnes Pratiques

1. **Toujours séparer les données** : Train/Validation/Test (dans cette ordre)
2. **Nettoyer les données** : Gérer les valeurs manquantes et encoder les catégories.
3. **Standardiser les features** : Surtout pour les algorithmes sensibles à l'échelle (ex: SVM, KNN).
4. **Utiliser la validation croisée** pour évaluer la généralisation.
5. **Optimiser les hyperparamètres** avec `GridSearchCV`.
6. **Visualiser les résultats** : Matrices de confusion, courbes ROC, projections 2D.