Dr. Christian Tsoungui Obama

Fiche mémo ML #2

Votre feuille de route pour construire un modèle de classification déployable

Un guide fondamental pour les aspirants Data Scientists

Basé sur
"Machine Learning avec Scikit-learn, Keras & TensorFlow"
par

Aurélien Géron

Étape 1 : Bien cadrer le problème

- Imaginez une banque examinant des demandes de prêt et essayant de répondre à la question : « Étant donné le revenu et l'historique de crédit d'une personne, pouvons-nous prédire si son prêt doit être approuvé ? »
- Pensez maintenant à une usine qui surveille ses machines. Quelqu'un demande : « Sur la base des relevés de vibrations et de température, pouvons-nous prédire si une machine fonctionne normalement ou est sur le point de tomber en panne ? »

Répondre à de telles questions en utilisant l'apprentissage automatique (ML) nécessite de suivre une feuille de route comme illustrée dans la « Fiche mémo ML #1 » de cette série ML (voir Figure 1).

Les étapes pratiques pour construire des modèles de classification sont décrites ci-après.

Étape 2 : Acquérir les données

Sources de données possibles (voir Fiche mémo ML #1).

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Charger un DataFrame avec une version specifique d'un CSV
loan = kagglehub.dataset_load(
                                KaggleDatasetAdapter.PANDAS,
                                "taweilo/loan-approval-classification-data/versions/1",
                                "loan_data.csv",
                              )
```

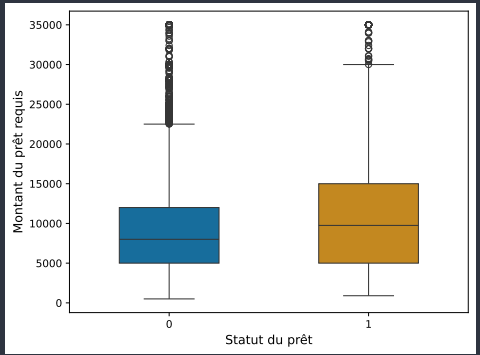
Exemple de code pour importer l'ensemble de données "Approbation de prêt" depuis Kaggle[1]

[1] www.kaggle.com/datasets/taweilo/loan-approval-classification-data

Étape 3 : Explorer les données

Analyse exploratoire des données (AED)

1. diviser les données en ensembles d'entraînement et de test
2. examiner la corrélation linéaire et le déséquilibre des classes cibles (attributs numériques)
3. transformer les données à l'aide de pipelines (ex: imputation, one-hot-encoding)



Déséquilibre de la classe cible : attribut potentiellement non discriminatif (chevauchement important)

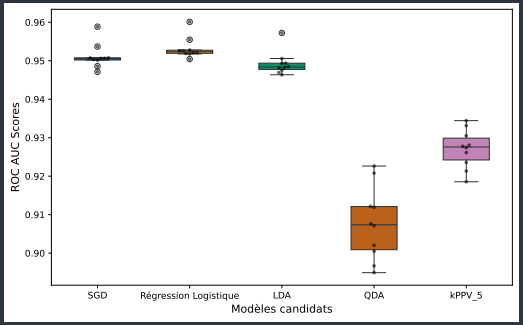
Matrice de corrélation : les valeurs quantifient la force et la direction de la corrélation linéaire

Étape 4 : Entraîner et évaluer les modèles candidats

Quelques modèles candidats pour la classification :

Modèles	Avantages	Inconvénients
Régression Logistique	simple, interprétable, rapide à entraîner	suppose une relation linéaire entre attributs et log-odds & sensible à la multicolinéarité
Analyse Discriminante Linéaire (ADL)	efficace en calcul & fonctionne bien avec des attributs normalement distribués	suppose la normalité et des matrices de covariance égales, sensible aux valeurs aberrantes et à la multicolinéarité
K-Plus Proches Voisins (KPPV)	simple, non paramétrique & modélise des frontières de décision complexes	coûteux en calcul pour les grands ensembles de données, sensible aux attributs non pertinents et à la mise à l'échelle

```python
# Entrainement du classifieur de regression logistique

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

score = "roc_auc"

# Regression logistique
log_reg = LogisticRegression()
log_roc_auc_scores = cross_val_score(log_reg, loan_prepared, loan_labels, scoring=score, cv=10)
```



Performance des modèles : scores de l'aire sous la courbe ROC (AUC – le plus élevé est le meilleur)

Étape 5.1 : Affiner le meilleur modèle candidat (Régression logistique)

Les hyperparamètres du modèle sont affinés pour améliorer ses performances :

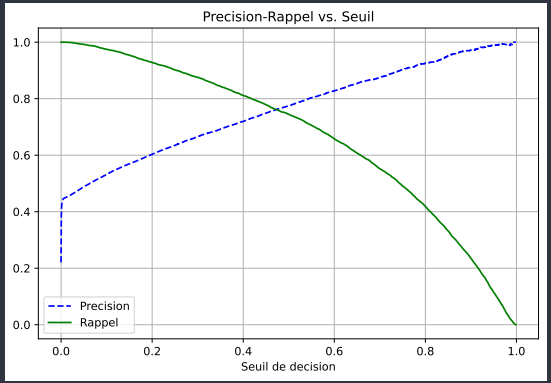
1. explorer un large espace avec la recherche aléatoire
2. affiner la recherche avec la recherche par grille
3. utiliser la validation croisée dans les deux cas

```
# Affinage des parametres du modele avec la recherche par grille
# Pipeline pour definir les iterations a l'interieur de la validation croisee
pipe_crossval = Pipeline([('log_reg', LogisticRegression(max_iter=2000))])
param_grid = {
    'log_reg__C': [0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19],
    'log_reg__penalty': ['l1'],
    'log_reg__solver': ['liblinear']
            }
grid_search = GridSearchCV(estimator=pipe_crossval,
                           param_grid = param_grid,
                           cv=5,
                           scoring='roc_auc',
                           n_jobs=-1,
                           verbose=2)

# Entrainer le modele pour toutes les combinaisons de parametres
grid_search.fit(loan_prepared, loan_labels)

# Selection du meilleur modele de la recherche par grille
best_model = grid_search.best_estimator_
```

Étape 5.2 : Être conscient du compromis Précision-Rappel

Selon l'application, on peut vouloir soit :

- une haute précision, c'est-à-dire que si le modèle prédit la classe positive, il est très probable qu'il ait raison,
- un haut rappel, c'est-à-dire que si la vraie classe est positive, le modèle est très susceptible de la prédire comme telle.



Compromis Précision-Rappel : pour l'approbation de prêt, la précision est préférable au rappel (seuil de décision : 60 %)

Étape 6 : Évaluer sur les données de test la généralisation du modèle affiné

Préparer les données de test avec le même pipeline que celles d'entraînement

```
# Definition de l'ensemble de donnees de test
X_test = strat_test_set.drop("loan_status", axis=1)
y_test = strat_test_set["loan_status"].copy()

X_test_prepared = full_pipeline.fit_transform(X_test)

```

Évaluer la performance du modèle

```
# Execution des predictions sur l'ensemble de test
# Probabilites predites pour la classe positive (1 = pret approuve)
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

y_proba_test = best_model.predict_proba(X_test_prepared)[:, 1]
threshold = 0.6
y_pred_test = (y_proba_test >= threshold).astype(int)

print("Score ROC AUC:", roc_auc_score(y_test, y_pred_test))

```

Étape 7 : Sauvegarder le modèle pour un déploiement éventuel

```python
# Sauvegarde du modele en tant que fichier joblib
def save_model(model, name="model", model_path="./"):
    """
    Sauvegarde le modele avec un nom donne a un emplacement specifique.

    Parametres
    ----------
    model :
        Le modele entraine.
    name:
        Le nom de fichier sous lequel sauvegarder le modele
    model_path:
        Le chemin dans lequel le modele est sauvegarde
        Exemple:
            "./models/"

    """
    os.makedirs(model_path, exist_ok=True)
    model_name = name + ".pkl"
    pkl_path = os.path.join(model_path, model_name)
    joblib.dump(model, pkl_path)

MODEL_PATH = os.path.join("models", "loan")
save_model(best_model, name="Regression_Logistique_V1.1", model_path=MODEL_PATH)
```

Avez-vous trouvé cela utile ?

Obtenez le notebook complet : » Dépôt GitHub «

Commentez ci-dessous : Pouvez-vous penser à un exemple où le rappel est plus important que la précision ?

Suivez-moi : pour la prochaine fiche mémo de cette série.

Aimez ce post : pour l'aider à atteindre plus de monde.

Restons en contact !

/in/christiantsoungui @chrisTs