

TP · GUIDE

TP League of Legends prédire une victoire à 10 minutes

Objectif et données

- Question : l'équipe bleue gagne-t-elle la partie ? Cible blueWins.
- 9 879 parties ; statistiques numériques des dix premières minutes.

1. Charger et vérifier

```
import pandas as pd
df = pd.read_csv(
    'data/high_diamond_ranked_10min.csv')
print(df.shape)
print(df['blueWins'].value_counts(normalize=True))
```

- Ouvrir lol_jour1_exercices.ipynb : types, classes, valeurs manquantes.
- Librairies : pandas, matplotlib, scikit-learn.

2. Réserver le test

```
from sklearn.model_selection import train_test_split
X = df.drop(columns=['gameId', 'blueWins'])
y = df['blueWins']
X_dev, X_test, y_dev, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

- Garder X_test et y_test fermés jusqu'à l'étape finale.
- Stratifier pour préserver les proportions des deux classes.

3. Baseline et arbre trop profond

```
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
baseline = DummyClassifier(strategy='most_frequent')
arbre = DecisionTreeClassifier(random_state=42)
# Comparer train et validation interne
```

- Métriques : accuracy, matrice de confusion, AUC ROC.

4. Comparer sur les mêmes plis

```
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
cv = StratifiedKFold(5, shuffle=True, random_state=42)
logit = make_pipeline(StandardScaler(),
                      LogisticRegression(max_iter=1000))
```

- Ajouter k-NN et arbre ; calculer moyenne et écart-type.
- Pipeline évite d'ajuster le scaler sur un pli de validation.

5. Régulariser

```
from sklearn.model_selection import GridSearchCV
recherche = GridSearchCV(
    logit, {'logisticregression__C': [0.01, 0.1, 1, 10]},
    cv=cv, scoring='accuracy')
recherche.fit(X_dev, y_dev)
```

- Petit C = pénalisation plus forte
- Aucun hyperparamètre n'est choisi sur X_test.

7. Ajouter la forêt

```
from sklearn.ensemble import RandomForestClassifier
foret = RandomForestClassifier(
    n_estimators=100, min_samples_leaf=5,
    random_state=42)
# Comparer sur les mêmes cinq plis
```

- Comparer coût, stabilité et performance
- Choisir et figer le modèle avant le test final.

8. Test final et erreurs

```
modele_final = recherche.best_estimator_  
predictions = modele_final.predict(X_test)  
probabilites = modele_final.predict_proba(X_test)[:, 1]  
# Accuracy, AUC, matrice de confusion, erreurs
```

- Ouvrir le test une seule fois après le choix.
- Décrire les parties où le modèle s'est trompé.

9. Sauvegarder et recharger

```
from joblib import dump, load
dump(modele_final, 'lol.joblib')
modele_recharge = load('lol.joblib')
# Vérifier la stabilité des prédictions
```

- Sauvegarder le Pipeline entier, normalisation comprise.

Références

[S7] Kaggle · LoL

<https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min>

[S2] scikit-learn · Pièges

https://scikit-learn.org/stable/common_pitfalls.html

[S3] scikit-learn · Validation croisée

https://scikit-learn.org/stable/modules/cross_validation.html