CHALLENGE KAGGLE : PREDICTION DE MATCH DE TENNIS

FDMS

Jeremy Lê

Introduction

J'explique dans ce rapport le processus de data mining réalisé lors de ce challenge sur la prédiction de match de tennis professionnel. Ce rapport s'accompagne d'un notebook. On y retrouve le code permettant la prédiction de notre fichier test.

Vous pourrez trouver des notes directement dans le notebook décrivant les choix faits. Cependant, nous reviendrons en détails pourquoi ces choix ont été fait.

De plus, j'analyse les résultats de mes modèles sur des cross validation… Je n'ai pas pu envoyer de soumission ou accéder aux leaderboard… Ainsi, le travail ne sera effectué qu'en locale car une amélioration en locale devrait se traduire par une amélioration sur le leaderboard.

Description du problème

Il faut au préalable s'interroger sur la problématique posée et transformer ce problème en un problème de Machine Learning. Ici, le challenge consiste à prédire des matchs de tennis sur une certaine période (2012 à 2016). Notre ensemble d'apprentissage et test contiennent les matchs dans cette période. Ainsi, on définira la victoire de l'identifiant 0 par la valeur 1 et la victoire de l'identifiant 2 par 1.

Dans ce cas, on peut traduire notre problème par un problème de classification ou de régression. Nous utiliserons deux familles d'algorithmes pour réaliser notre classification : Decision Tree et SVM.

Description des données

Provenance des données

Les données ont été récupéré sur une base de données SQL fourni par Monsieur Denoyer. Nous n'avons aucune information sur la provenance de ces données. Nous ne pouvons donc établir la véracité de nos données en analysant la source des données.

Games_atp et Games_atp_public

Ces deux tables constitueront l'ensemble d'entrainement et l'ensemble de test. On entrainera nos modèles sur la base d'entrainement (games_atp) et on réalisera nos soumissions qui n'est autres que la prédiction de notre ensemble de test (games_atp_public)

Games_atp contient XXXXX match de 2012 à 2016. Elle est constituée de l'identifiant des deux joueurs, l'identifiant du tournoi, l'identifiant du round (décrit ci-dessous), la date du match et le résultat du match. Ici, le joueur 1 gagne à chaque le match : il faudra donc transformer cette table pour avoir autant de label positif que négatif.

Games_atp_public contient XXXXX matchs pour la fin 2016. Elle est constituée de l'identifiant des deux joueurs, l'identifiant du tournoi, l'identifiant du round (décrit ci-dessous) et la date du match.

Rounds
Cette table contient le mapping entre l'identifiant d'un round et sa valeur sémantique (Finale, demi-finale, quart-finale ,...).

Tours
Cette table contient le mapping entre l'identifiant d'un tournoi et les valeurs associés à un tournoi. En effet, chaque tournoi de chaque année a des identifiants différents. (Ex : Roland Garros apparait autant de fois qu'il y a d'années).

Players_atp
Cette table contient le mapping entre l'identifiant d'un joueur et les caractéristiques de celui (entre autres le nom).

Stat_atp
Cette table contient l'historique des matchs professionnelles. On y retrouve les statistiques moyennes de chaque match. Nous utiliserons principalement cette table afin de créer nos features pour un match. De plus, de nombreuses colonnes ne contiennent que peu de valeur (<20%). Ainsi, nous n'utiliserons les colonnes ayant suffisament de valeur pour faire nos features.

Facts_atp
Cette table contient des informations (nombre total de aces, point gagnées…) sur certains joueurs pour certaines années. Cependant, nous verrons dans le notebook qu'elle n'est pas très grande … Nous ne l'utiliserons donc pas dans la suite du projet.

Feature Extraction
Comme nous l'avons dit plus haut, la table stat_atp nous servira de base pour créer nos features. Nous utiliserons l'historique des matchs d'un joueur à la date du match pour obtenir les features du match.

De plus, les features d'un match doivent être symétriques car lorsque l'on inverse l'identifiant 1 et 2 du match, la prédiction doit être inversé (0 → 1). C'est pourquoi nous utiliserons la différence des features associés aux deux joueurs pour créer les features finales du match.

Voici la liste des features ainsi crées :
- RANK : différence de rank entre joueur 1 et 2
- POINT : différence de points atp
- W1SP : différence entre de points gagnées au premier service
- W2SP : différence de points gagnées au second service
- WSP : différence de points gagnées au service
- WRP : difference de points gagnées sur retour
- ACES : difference de aces
- DF : difference des doubles fautes
- BP : difference des balles de break convertis
- TPW : difference des pourcentages de points gagnés lors d'un match
- Dummy variables liés aux ratings du tournoi, le round et le type de terrain.

Toute ces valeurs seront normalisées dans [-1;1].

Feature creation

En plus des features extraites directement de notre base, nous pouvons créer des features pour augmenter l'expressivité de nos features.

Voici les features que j'ai implémenté :

- Complete : produit entre le nombre de points gagnés aux services et le nombre de points gagnés au retour. Lorsqu'un joueur est complet, il est fort dans ces deux secteurs de jeu.

- Advantage service : difference de produit entre l'identifiant 1 et 2 sur le service et le retour.

Data cleansing

Valeur manquante

Nous avons considéré de prendre deux techniques d'imputation différentes pour nos données : imputer les valeurs manquantes par la moyenne ou retirer les matchs où il manque des valeurs.

On obtient de meilleurs résultats dans le cas où l'on impute par la moyenne. Ainsi, nous considérons tous les résultats pour cette technique.

Nous avons donc imputé stat_atp avec la moyenne pour calculer les features. De plus, après la création de nos features, nous avons rempli certaines colonnes par des valeurs spécifiques à la variable :
RANK : Lorsque l'on ne connait pas le rank d'un joueur, on lui attribue le ranking le plus grand (900)
POINT : Lorsque l'on ne connait pas le rank d'un joueur, on lui attribue le ranking le plus bas(0)

Valeur aberrante

En observant les boxplots, on s'aperçoit qu'il peut y avoir des valeurs aberrantes. Cependant ces valeurs correspondent à des situations exceptionnelles. (Ex : Maximum de Aces en un match est de XXX, or un joueur du nom de karlovic est très bon au service (car il est grand), il fait donc en moyenne beaucoup plus d'ace)

Models

Nous avons défini trois modèles de features. Dans ces trois modèles, on utilise l'historique des matchs pour créer les features. Puis dans le troisième modèle, on split les historiques par le type de surface du match.

Dans chaque modèle, on obtient les résultats en moyennes sur 5 fold pour la cross-validation.

Point & Rank difference

Afin d'avoir un benchmark de notre problème, nous avons fait un simple modèle utilisant seulement les features RANK et POINT.

Random Forest (default parameters)	0,591
Random Forest (Random search)	0,651
SVM	0,470

Voici les paramètres optimaux pour le Random Forest :
{'min_samples_split': 9, 'bootstrap': True, 'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 3}

Historique

Ici, nous prenons les features extrait et construit présenté dans les sections précédentes.

Dans un premier temps, nous avions obtenu un score de 0.89 avec un simple Random Forest… Ce qui semblait énorme… La raison était que lors de la création des features DIRECT et MATCH_WON, nous utilisions le match en question pour prédire ces features (date de match était supérieur à la date du tournoi…). Ainsi, nous avons ré-implémenté ces deux features.

Voici les résultats sans ces deux features biaisés :

Random Forest (default parameters)	0,686
Random Forest (Random search)	0.714
SVM	0,470

Voici les paramètres optimales pour le random forest:
{'bootstrap': True, 'min_samples_leaf': 8, 'min_samples_split': 10, 'criterion': 'entropy', 'max_features': 17, 'max_depth': 4}

Historique selon type de surface

Dans ce modèle, nous avons décidé de prendre en compte que les matchs historiques de même types de terrains pour la création des features. Ainsi, nous avons mis un poids de 1 pour les matchs de même type et 0 sinon.

Random Forest (defaut parameters) : 0,499

Le résultat est faible… En effet, en affectant ces poids à l'historique des matchs, le nombre de matchs considérés est considérablement réduit… biaisant notre modèle…

Pour aller plus loin

Nous nous sommes contentés du random forest avec les hyperparamètres optimaux dans le modèle historique pour faire notre soumission.

Cependant voici quelques pistes pour augmenter notre score de prédiction :

- Chercher des données supplémentaires : enrichir nos données peuvent augmenter son expressivité.
- Créer de nouveaux features : par exemple la fatigue.
- Apprendre des poids sur les surfaces : le modèle historique selon type de surface ne fournissait pas de bon résultat mais si l'on apprend les poids des surfaces, on peut surement augmenter notre score. (En effet, un joueur peut être bon sur une surface mais pas sur une autre : par exemple Nadal gagne tous les Roland Garros mais n'est pas aussi bon sur les autres surfaces)

- Mettre des poids selon la date du match : Afin d'amener de la temporalité à notre modèle, on peut affecter un poids à un match selon la date à laquelle il a été joué.
- Combinaison de model (Boosting) : En général, la combinaison de modèle augmente le score de prédiction de notre modèle.
- SVM avec RFE : Le RFE n'augmentera pas notre score mais réduira le nombre de feature à utiliser sans pour autant descendre considérablement notre score.
- Réseau de neurones : le Deep Learning permet de répondre à de nombreuses problématiques.