Python for Data Analysis

Devoir maison : prédiction du temps avant résolution de tickets d'incidents

Description du problème

- On cherche à prédire la durée entre la création d'un ticket et sa résolution.

Description du jeu de données

- Le jeu de données provient du UC Irvine Machine Learning Repository
- Il contient les logs d'un système de gestion de tickets
- Il contient 141 712 lignes et 36 colonnes
- Une description est fournie pour chaque colonne

Description des colonnes

- *number*: incident identifier with the same number as total cases;
- *incident state*: attribute with eight levels controlling incident management process transitions from opening until closing the case;
- *active*: boolean attribute indicating if record is active or closed/canceled;
- *reassignment_count*: number of times incident has changed group or support analysts;
- *reopen_count*: number of times incident resolution was rejected by caller;
- *sys_mod_count*: number of incident updates until that moment;
- *made_sla*: boolean attribute to incident exceeded target SLA or not;
- *caller_id*: user identifier affected;
- *opened_by*: user identifier that reported the incident;
- *opened_at*: incident opening date and time;
- *sys_created_by*: user identifier that registered the incident;
- *sys_created_at*: incident creation date and time;
- *sys_updated_by*: user identifier that made update and generated current log record;
- *sys_updated_at*: log update date and time;
- *contact_type*: categorical field with values indicating how incident was reported;
- *location*: location identifier of place being affected;
- *category*: description of the first level of service being affected;
- *subcategory*: description of the second level of service being affected – related to first level;

Description des colonnes

- *u_symptom*: description about user perception of service availability;
- *cmdb_ci*: (confirmation item) identifier (not mandatory) referencing homonyms relation and used to report item being affected;
- *impact*: description of the impact caused by incident. Values are: 1–High; 2–Medium; 3–Low;
- *urgency*: description to the urgency asked by user for incident resolution. Values are same as *impact*;
- *priority*: priority calculated by system based on *Impact* and *urgency*;
- *assignment_group*: identifier referencing the relation *Group* (database relational model in *ServiceNow*^{TM}) describing support group in charge of incident;
- *assigned_to*: user identifier in charge of incident;
- *knowledge*: boolean attribute indicating whether a knowledge base document was used to resolve incident;
- *u_priority_confirmation*: boolean attribute indicating whether *priority* field was double checked;
- *notify*: categorical attribute indicating whether notifications was generated for this incident;
- *problem_id*: identifier referencing homonyms relation describing problem identifier associated with this incident;
- *rfc*: (chance request) identifier referencing homonyms relation describing change request identifier associated with incident;
- *vendor*: identifier referencing homonyms relation describing vendor in charge of incident;
- *caused_by*: relation with RFC code responsible by the incident;
- *close_code*: resolution code of the incident;
- *resolved_by*: user identifier who resolved the incident;
- *resolved_at*: incident resolution date and time;
- *closed_at*: incident close date and time;

Analyse et transformation du jeu de données

- Le jeu de données contient des logs : une ligne correspond donc à une mise à jour d'un ticket. Il existe donc plusieurs lignes pour un même ticket.
- La description des colonnes nous indique que la colonne « priority » est calculée à partir des colonnes « impact » et « urgency ». On décide donc de ne garder que « priority » et de supprimer les deux autres.
- On transforme la colonne « priority » de type *object* en colonne de type *int* en conservant bien l'ordre des différentes valeurs.
- Pour la colonne « notify », il existe deux valeurs possibles qui peuvent correspondre à des valeurs booléennes. On les transforme donc dans ce sens.
- Cinq colonnes contient plus de 98% de valeurs inconnus (« vendor », « cmdb_ci », « problem_id », « rfc », « caused_by »), on décide de les supprimer.

Analyse et transformation du jeu de données

- La valeur que l'on cherche à prédire correspond à la différence entre « resolved_at » et « opened_at », qui représentent des dates. On crée donc une nouvelle colonne, « delta », correspondant à cette différence (en secondes).
- La colonne « contact_type »
  - 5 valeurs possibles mais 98 % des entrées sont égales à « Phone »
  - En calculant la moyenne de delta pour chaque type de contact, on remarque que pour « Self-Service » par exemple, la durée moyenne avant résolution est supérieure à celle de « Phone ».
  - On décide de garder tout de même la colonne bien qu'elle soit déséquilibrée puisque ce déséquilibre représente peut-être une réalité à prendre ne compte ?

Analyse et transformation du jeu de données

- La plupart des colonnes sont de type *object* (ce sont des chaînes de caractères).
- Afin de pouvoir être traitées par nos modèles, il faut préalablement les encoder (comme pour « priority » et « notify »).
- Dans un premier temps, on a tenté de réaliser un one-hot encoding sur l'ensemble de ces colonnes *object* puisque, contrairement à la colonne « priority », il n'existe pas de relation d'ordre entre les différentes valeurs possibles.
- Le jeu de données passe alors de 36 colonnes à plus de 6 000 colonnes. Une fois passé dans un modèle de régression linéaire, on ressort avec un score nul. N'arrivant pas à expliquer ce résultat et le dataset étant devenu très lourd (temps de traitement très long), on décide de ne dummifier que la colonne « contact_type » et d'encoder les autres en utilisant un LabelEncoder.
- L'inconvénient du LabelEncoder est qu'il va transformer les différentes chaînes en valeurs numériques et qu'il peut induire une relation d'ordre entre celles-ci, relation d'ordre qui n'existe pas en réalité. On décide de tout de même prendre ce risque.

Analyse et transformation du jeu de données

- Il aurait pu être intéressant de calculer la durée pendant laquelle un ticket est en mode « Awaiting XXX ».
- Une fonction a commencé à être développée dans ce sens mais elle a été abandonnée puisque trop gourmandes en ressources et surtout compliquée à développer. Il faut réussir à repérer les changements de status, en sachant qu'il peut y avoir plusieurs entrées de log pour un même statut si d'autres infos ont été modifées.

API Django

- Afin de simplifier l'utilisation de l'API, on a généré un modèle basé sur les features les plus importantes d'un précédent modèle.