École Centrale de Nantes

INFO IA - STASC
Prévision immobilier : Institut Louis Bachelier
Rapport

Real estate price prediction

Élèves :
Clément Auclin
Félix Doublet

Enseignant :
Bertrand Michel

20 mars 2023

Table des matières

1 Présentation du jeu de données **2**
1.1 Objectif du challenge . . . 2
1.2 Caractéristiques du dataset . . . 2

2 Outils utilisés **2**

3 Manipulation des données **3**
3.1 Feature Engineering . . . 3
3.1.1 Données catégorielles et valeurs manquantes . . . 3
3.1.2 Correlation entre les features . . . 4
3.2 Clustering . . . 5
3.3 Traitement des images . . . 6
3.3.1 Création d'un score . . . 6
3.3.2 Traitement n'ayant pas été retenu . . . 7
3.4 Normalisation . . . 7
3.5 Agrégation à des datasets du gouvernement . . . 7

4 Analyse et résultats **8**
4.1 Modèle retenu . . . 8
4.1.1 Démarche . . . 8
4.1.2 Principe du gradient boosting . . . 8
4.1.3 Tuning du modèle . . . 8
4.2 Résultats obtenus . . . 9

5 Conclusion **9**

1 Présentation du jeu de données

Le challenge consiste en un problème de régression pour la prédiction de la valeur de biens immobiliers. Cette compétition est à l'initiative de l'institut Louis Bachelier qui a récolté un jeu de données sur les ventes immobilières sur le territoire français.

1.1 Objectif du challenge

Au delà d'un problème classique de régression sur l'immobilier (un des challenges les plus populaires de Kaggle porte sur l'immobilier américain) l'objectif est de mesurer si l'ajout de 1 à 6 photos pour chaque maison permet d'améliorer la prédiction.

Afin de mesurer nos résultats, nous avons à disposition X_{train} et y_{train} pour entraîner notre modèle avant de réaliser des prédictions sur le dataset X_{test}.

La métrique d'évaluation est la Mean Pourcentage Absolute Error qui mesure l'écart en pourcentage de nos prédictions par rapport à la réalité.

1.2 Caractéristiques du dataset

Le dataset se décompose alors en approximativement 40000 données d'entraînement et 10000 de test.

On retrouve des features attendues pour de l'estimation immobilière :

— Des données géographiques (ville, position géographique, code postal)
— Des caractéristiques du logement (type de logement, superficie, superficie du jardin s'il y en a un, nombre de pièces ...)
— Des informations énergétiques (indicateurs énergétiques, exposition)

La particularité de ce dataset se trouve alors dans la présence de 1 à 6 photos pour chaque ligne du dataset. Nous essaierons dans la suite d'en tirer profit pour améliorer nos prédictions.

2 Outils utilisés

Afin de partager entre nous les codes de nos différentes tentatives et de collaborer au mieux, nous avons créé un repository GitHub à l'intérieur duquel nous avons créé plusieurs dossiers pour segmenter le projet.

Nous avons travaillé sous la forme de notebook afin de faciliter la relecture et la compréhension mais également pour faciliter les différents tests.

Aussi, nous avons créé un fichier .txt dans lequel nous écrivions certains résultats importants (les hyperparamètres optimisés par exemple), les tâches à réaliser et les pistes à approfondir.

Enfin, ce rapport est écrit en collaboration sur Overleaf en format Latex.

3 Manipulation des données

3.1 Feature Engineering

Afin de traiter au mieux les données, de nombreux traitements successifs ont été implémentés, et parfois supprimés. Nous avons testé l'efficacité des différents traitements des données en créant un modèle de XGBoost -que nous aborderons plus tard- que nous avons entrainé sur 80% des valeurs de train du site, afin de pouvoir le tester sur 20% des valeurs et vérifier ses résultats :

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size
    =0.2,random_state=1)
```

Il est important également de noter, qu'après un rapide état de l'art sur des problématiques similaires, un logarithme a été appliqué sur les valeurs de "y" qui correspondent donc aux prix des biens immobiliers. Ce logarithme a permis un gain significatif de précision.

$$log_{1p}(x) = log(1 + x) \tag{1}$$

En effet, on passe d'une distribution de type "Johnson SU" à une distribution qui se rapproche très bien d'une loi normale. Ainsi, les moyennes et variances prennent plus de sens et les valeurs à prédire sont moins distantes les unes des autres.

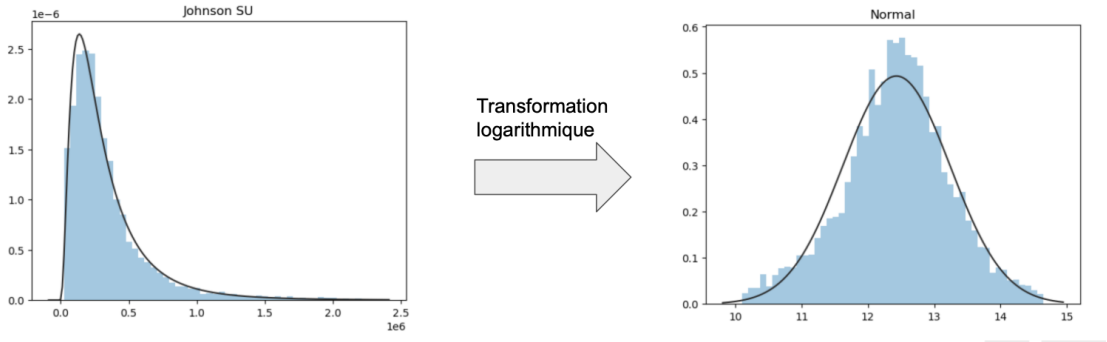

FIGURE 1 – Distribution de prix avant et après l'application logarithmique

3.1.1 Données catégorielles et valeurs manquantes

Nous avons traité en premier les données non-numériques. Pour cela, nous avons créé un tableau récapitulant plusieurs informations importantes à leur sujet, afin de pouvoir ensuite choisir de supprimer ces colonnes, ou bien de les modifier. Voici le tableau :

	Nb de valeurs différentes	Nb de valeurs manquantes	Ratio de valeurs manquantes
property_type	[22]	0	0.000000
city	[8643]	0	0.000000
energy_performance_category	[7]	18300	0.489724
ghg_category	[7]	18838	0.504121
exposition	[12]	28274	0.756637

Les choix qui ont été faits ensuite sont les suivants :

— suppression de la colonne *exposition* car il manquait plus de 75% des valeurs

— suppression de *energy_performance_category* et *ghg_category* car il manque de nombreuses valeurs et car *ghg_value* et *energy_performance_value* donnent des informations plus précises
— Application d'un **One Hot Encoding** à *property_type*, afin de séparer les catégories en plusieurs features
— Application d'un **Label Encoder** à *city*

Pour les valeurs manquantes restantes des colonnes à valeurs numériques, les données ont été remplies en faisant la moyenne du département en question, afin d'obtenir une valeur la plus proche de ce qu'elle serait réellement (sauf *floor* et *land* qui ont été complétées par des 0).

3.1.2 Correlation entre les features

Une fois les traitements préliminaires effectués, nous avons réalisé des matrices de corrélations entre les features du dataset (en comptant le prix à prédire), afin de déterminer quelles seraient les données les plus à même d'être modifiées, ou mélangées. Pour cela, la première matrice de corrélation a été réalisée sur l'ensemble du dataset (et est disponible en Annexe 1). Ensuite, nous avons regardé de manière plus précise les données qui présentaient le plus de corrélation avec le prix. On obtient alors ceci :

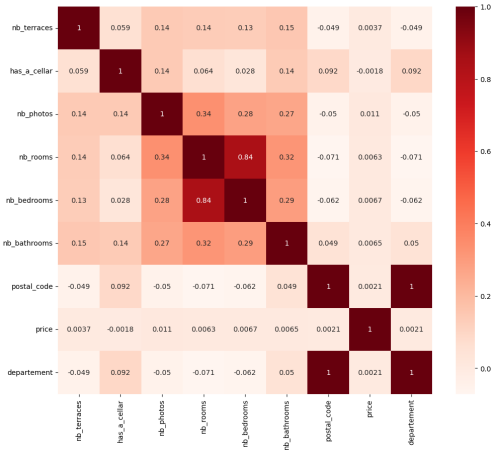

FIGURE 2 – Matrice de corrélation sur quelques variables

A partir des résultats fournis par cette matrice, on choisit de créer de nouvelles colonnes afin d'aider le modèle à comprendre certaines informations :
— La somme du nombre de pièces et de chambres
— La somme du nombre de pièces et de salles de bains
— La différence entre le nombre de chambres et de salles de bains (afin de s'avoir si certaines chambres possèdent des salles de bains privatives)
— Un rapport entre la taille du bien immobilier et le nombre de pièces

D'autres features avaient été créées initialement, mais elles n'avaient finalement aucun impact sur les prédictions. Elles ont donc été supprimées. On obtient alors finalement comme matrice de corrélation :

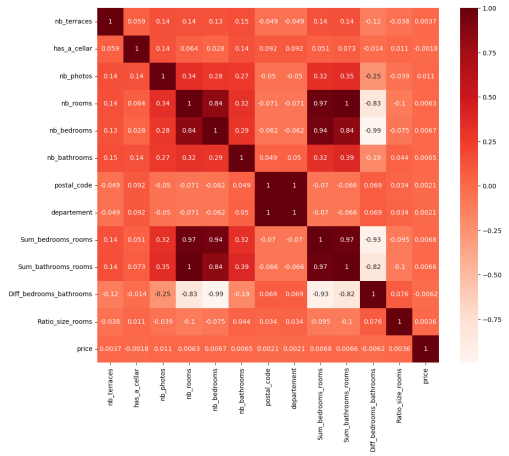

FIGURE 3 – Matrice de corrélation après rajout de features

De plus, deux autres features ont été rajoutées, l'une correspondant à la racine carrée de *size* et l'autre correspondant à la racine carré de *land_size*.

3.2 Clustering

Ensuite, nous avons réalisé du clustering pour regrouper les valeurs afin que le modèle de Machine Learning soit plus efficace. Pour cela, un dataset a été ajouté à celui de départ afin d'obtenir des features sur le prix moyen au m^2 en 2019. Ce rajout sera détaillé plus loin, dans la partie sur l'agrégation d'autres datasets. Ce qu'il faut retenir pour le moment, c'est qu'une variable de prix "approximatif" a ainsi pu être créée.

```
X["nearly_price"]=X["PrixMoyen_M2"]*X["size"]
```

Afin de se rapprocher le plus possible de ce clustering inutilisable, nous nous sommes servis de la feature nouvellement créée *nearly_price*, qui correspond à l'approximation la plus précise du prix que nous ayons. Un nouveau clustering à 15 catégories a alors été généré :

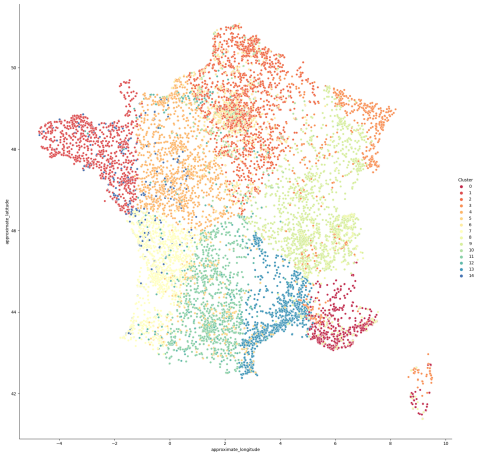

FIGURE 4 – Clustering avec la localisation et des prix approximés

Il est intéressant de noter que les découpages s'apparentent à ceux des régions françaises.

Nous avions commencé initialement par réaliser un clustering entre trois features : la longitude, la latitude et le prix. Ce clustering a été réalisé dans l'unique but d'essayer de voir ce qu'il était possible d'obtenir, mais ne peut bien évidemment pas être utilisé puisqu'il fait intervenir directement la colonne à prédire (on pourrait parler alors de **Data Leakage**). Les résultats de ce clustering sont disponibles en Annexe 2.

3.3 Traitement des images

Les coeur de ce sujet était de pouvoir améliorer les résultats des données tabulaires à l'aide d'images que nous pouvions traitées. Notre approche est basée ici sur deux points, le second n'ayant pas été retenu.

3.3.1 Création d'un score

La première approche, et la plus efficace, a été de créer des scores par images. Pour cela, un modèle de Deep Learning pré entrainé, RoomNet, a été utilisé. La méthodologie a alors été la suivante :

Pour chaque annonce (repérée par un id d'annonce et qui possède entre 0 et 6 images), on vient réaliser une prédiction sur chacune des ses images. La prédiction obtenue correspond à une pièce parmi 6 possibles : "Backyard", "Bathroom", "Bedroom", "Frontyard", "Kitchen" ou "LivingRoom". On enregistre alors pour chaque image la pièce obtenue, ainsi que la fiabilité de la prédiction (un nombre entre 0 et 1). Une fois l'opération réalisée sur chaque image, on stocke le résultat dans un fichier excel, et on réitère sur l'annonce suivante.

Une fois l'ensemble des annonces traitées, on crée un vecteur de poids : chaque pièce se voit attribuer un poids en fonction de si on pense qu'une photo de cette pièce a une importance plus élevé qu'une autre. On obtient alors un vecteur de 6 poids qui restera fixe pour l'ensemble du traitement.

Il ne reste plus qu'à calculer pour chaque annonce le score final, en multipliant la fiabilité de chaque image par le poids associé à la pièce prédite. Pour cela, on parcours le vecteur de poids et pour chaque pièce, on regarde le nombre d'images correspondant, qu'on multiplie par le score et par la fiabilité moyenne de ces images.

On peut résumer le processus pour une annonce de la manière suivante :

FIGURE 5 – Calcul du score des images pour une annonce

On rajoute donc finalement à notre dataset pour chaque annonce le score obtenu, ainsi

que le nombre de chaque pièce. On remarque que la feature du score est la plus utilisée par XGBoost, ce qui montre l'importance qu'a celle-ci.

Le lien vers le GitHub de Roomnet ainsi que la partie modifiée pour notre usage (non visible dans le notebook final) sont dans l'annexe 3.

3.3.2 Traitement n'ayant pas été retenu

Un second traitement avait été envisagé au début, mais celui-ci déteriorait les prédictions finales et n'a par conséquent pas été retenu. Il consistait à calculer avec OpenCV pour chaque image d'une annonce la moyenne, le premier quartile et le troisième quartile des valeurs des pixels une fois l'image passée en nuances de gris. Une moyenne de ces valeurs par image était ensuite effectuée par annonce.

3.4 Normalisation

Normaliser les données est une manière de faciliter la convergence du modèle et de mettre les features "sur un pied d'égalité" en les poussant sur le même intervalle.

Nous avons essayé différentes techniques, notamment le RobustScaler avant d'opter pour le MinMaxScaler.

Il consiste simplement à "pousser" toute l'information dans l'intervalle [0,1] :

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

Cette transformation a effectivement amélioré nos résultats.

3.5 Agrégation à des datasets du gouvernement

Afin d'améliorer les résultats obtenus, nous avons eu recours à différents datasets provenant d'autres sources (souvent de l'INSEE).

L'idée derrière ces manipulations est que certaines informations qui nous semblaient importantes pour déterminer le prix d'une maison n'était pas présente parmi les features données. En particulier nous pensions qu'obtenir le prix moyen du mètre carré dans la ville augmenterait notre précision.

Après avoir tenté - sans succès - du scrapping sur internet (nous avons essayé de récupérer automatiquement dans le code html d'un site immobilier le prix au mètre carré pour chaque ville présente dans le dataset) nous avons trouvé sur le site du gouvernement ce que nous cherchions : les prix moyens de vente d'années précédentes (les mots précédents constituent un lien cliquable).

Nous avons alors effectué une jointure sur les villes entre X et ce dataset avec 90% de correspondance. Remplir les 10% restants artificiellement avec la moyenne par département a donné un moins bon résultat qu'en laissant les "NaN" que XGBoost gère bien.

D'autres datasets ont par la suite été agrégés, dont certains pour prendre en compte l'inflation.

4 Analyse et résultats

4.1 Modèle retenu

4.1.1 Démarche

Après avoir travaillé sur le dataset, il s'agissait ensuite d'entraîner un modèle de Machine Learning pour réaliser les prédictions. Après quelques tests et lecture des meilleurs travaux pour la compétition similaire de Kaggle, XGBoost est apparu comme le meilleur modèle.

Nous avons également fait des tentatives d'ensemble learning en entrainant plusieurs modèles (XGBoost, LightGBM, CatBoost, GradientBoostingRegressor) puis d'en faire une moyenne pondérée mais nous n'avons pas constaté d'amélioration notable. Pour des raisons de praticité, nous sommes retournés sur XGBoost (hyperparamètres à déterminer pour un seul modèle, temps de calcul réduit).

Des tentatives de Deep Learning ont aussi été effectuées avec des résultats inférieurs.

Avant d'évoquer les résultats que nous avons obtenus, revenons brièvement sur le principe du Boosting.

4.1.2 Principe du gradient boosting

En substance, le gradient boosting construit une série d'arbres, où chaque arbre est ajusté sur l'erreur de l'arbre précédent. À chaque tour, l'ensemble d'arbres s'améliore car chaque arbre est poussé dans la bonne direction par le biais de petites mises à jour. Ces mises à jour sont basées sur la perte du gradient.

XGBoost est une implémentation gloutonne (réaliser le minimum local à chaque étape en espérant réaliser le minimum global à la fin) de cet algorithme et est reconnue pour son efficacité et sa rapidité.

4.1.3 Tuning du modèle

Afin de trouver de bons hyperparamètres pour améliorer nos résultats, nous avons principalement utilisé la fonction GridSearchCV de scikit-learn. Elle calcule l'erreur de cross validation pour chaque combinaisons des hyperparamètres et renvoie la meilleure. Nous avons notamment fait tourner un PC pendant 20 heures afin d'optimiser simultanément n, $learning\_rate$ et max_depth.

Nous avons aussi essayer RandomGridSearch qui prend un nombre déterminé de combinaisons aléatoires des hyperparamètres et renvoie la meilleure, ou HalvingGridSearch qui fait la même chose que GridSearchCV mais en éliminant de manière "intelligente" des combinaisons pour gagner du temps d'exécution. Nous sommes finalement restés sur les résultats de GridSearch.

4.2 Résultats obtenus

A l'aide des méthodes explicitées plus haut, nous avons pu améliorer nos résultats au fur et à mesure. Deux transformations nous ont permis d'améliorer significativement la précision. Dans un premier temps, le passage au logarithme du prix, afin d'obtenir des écarts moins importants qui nous a permis à lui tout seul de passer d'environ 32 à 27, soit un gain de 5 points. Ensuite, le second bond est apparu lors de l'agrégation de prix de 2019 et d'autres années, grâce auxquels nous avons amélioré le score d'entre 3 et 4 points. De plus, afin d'utiliser au mieux les images, une fois un score proche de 23 obtenu, nous avions décidé d'effectuer un traitement similaire à celui de l'Institut Louis Bachelier (en utilisant OpenCV et des moyennes de pixels par exemple...). Cependant ce traitement là a dégradé nos résultats, il n'a donc pas été conservé finalement, et c'est l'utilisation améliorée et remaniée de RoomNet qui a été préférée comme méthode de traitement. Il est intéressant de noter, que nous n'avions pas pris en compte initialement la métrique demandée pour l'entraînement des modèles, c'était alors la méthode par défaut qui était utilisée, et la métrique du Data Challenge ne servait que pour tester les résultats.

Finalement, notre score final prend en compte tous les traitements vus dans les parties précédentes et est de 22.0982 avec la métrique choisi par l'ILB. Ce score nous place ainsi à la seconde place de ce Data Challenge, sur l'ensemble des équipes et personnes ayant participées depuis début 2022.

5 Conclusion

Ce Data Challenge nous a permis de nous familiariser avec de nombreuses méthodes de prédictions et de traitements des données. Nous avons pu prendre en main les bibliothèques les plus importantes dans ces domaines sur Python afin d'améliorer au mieux nos résultats. Les données de l'Institut Louis Bachelier nous ont servi de base pour travailler mais nous nous sommes assez rapidement mis à la recherche de nouvelles données qui pourraient nous être utiles. Une remarque que nous pourrions faire sur les données fournies initialement est le manque d'une date approximative (tel qu'un trimestre par exemple), pour la vente des biens. En effet, cette date aurait peut-être permis d'améliorer les résultats en prenant en compte une inflation plus précise.

Enfin, ce challenge a surtout été l'occasion d'expérimenter et de mettre en pratique les méthodes de Machine Learning. Nous avons effectué de nombreux tests - pas toujours concluants - et les progressions successives de notre score final en sont le reflet. Nous avons effectué au total plus de 50 soumissions dont une trentaine pour tenter d'améliorer notre meilleur score mais sans succès.

Annexe

Annexe 1

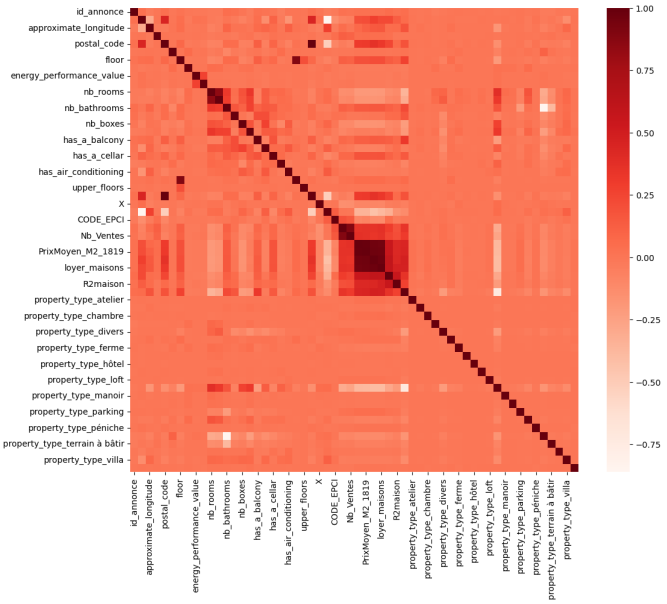

FIGURE 6 – Matrice de corrélation sur l'ensemble du dataset

Annexe 2

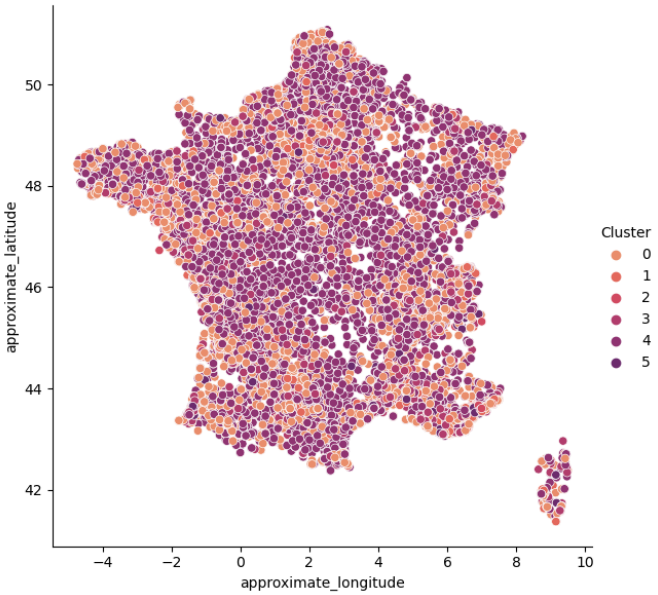

FIGURE 7 – Clustering du prix avec la localisation

Annexe 3

Lien vers le GitHub de RoomNet : https://github.com/GitBoSun/roomnet
Partie modifiée de RoomNet (non visible dans le notebook rendu) :

```
# le param tre "type_image" est rajout  pour diff rencier les
    images de train et de test
# Suppression du param tre overlay de la fonction (car il ne nous
    est pas utile ici)
def classify_im_dir(nn, imgs_dir,dossier,type_image):
    print('Classifying images in', imgs_dir)
    all_im_paths = glob(imgs_dir + '/*')
    num_fpaths = len(all_im_paths)

    # path chang , il y avait avant : out_dir = imgs_dir + '
       _classified'
    out_dir = f"./STASC/Data_Challenge/_classified/{type_image}"
    # Changement du = out_dir + "_results.xls" en = out_dir + "/" + '
       _results.xls'
    xl_fpath = out_dir + "/" + dossier + '_results.xls'
    class_dirs = [out_dir + os.sep + CLASS_LABELS[i] for i in range(
       len(CLASS_LABELS))]

    excel_file = xlwt.Workbook()
    sheet = excel_file.add_sheet('classification_results')
    sheet.write(0, 0, 'IMAGE_NAME')
    sheet.write(0, 1, 'PREDICTED_LABEL')
    for i in tqdm(range(num_fpaths)):
        fpath = all_im_paths[i]
        im = cv2.imread(fpath)
        infer_outs = nn.infer_optimized(im)
        pred_label = CLASS_LABELS[infer_outs[0][0]]
        pred_conf = infer_outs[1][0][infer_outs[0][0]]
        out_fpath_dir = out_dir + os.sep + pred_label
        # print(fpath, '--->', pred_label, pred_conf)

        shutil.copy(fpath, out_fpath_dir)
        sheet.write(i + 1, 0, fpath.split(os.sep)[-1])
        sheet.write(i + 1, 1, pred_label)
        sheet.write(i + 1, 2, str(pred_conf))
    excel_file.save(xl_fpath)
    return xl_fpath


if __name__ == '__main__':
    nn = RoomNet(num_classes=len(CLASS_LABELS), im_side=IMG_SIDE,
       compute_bn_mean_var=False,
                 optimized_inference=True)
    nn.load(INPUT_MODEL_PATH)

    # listes des dossiers dans les dossiers train et test
    FOLDER_TRAIN = 'C:/Users/maila/Documents/Centrale Nantes/EI2/
       INFOIA/STASC/Data_Challenge/reduced_images_ILB/reduced_images/
       train'
    list_folders_train = os.listdir(FOLDER_TRAIN)
    FOLDER_TEST = 'C:/Users/maila/Documents/Centrale Nantes/EI2/
       INFOIA/STASC/Data_Challenge/reduced_images_ILB/reduced_images/
```

```
        test'
    list_folders_test = os.listdir(FOLDER_TEST)

    for dossier in list_folders_train :
        # path absolu vers le folder
        folder_path = FOLDER_TRAIN + "/" + dossier
        xl_out_path = classify_im_dir(nn, folder_path,dossier,"train"
            )

    for dossier in list_folders_test :
        # path absolu vers le folder
        folder_path = FOLDER_TEST + "/" + dossier
        xl_out_path = classify_im_dir(nn, folder_path,dossier,"test")
```