Travail dirigé : Tests orientés data Troisième partie

Gaetan Corin

Écrivez des tests unitaires pour le composant Cleaner

La première étape a été de bien comprendre le fonctionnement du composant cleaner avant d'écrire des tests.

Le composant cleaner prend en entrée un tableau d'objet appelé des "tuplesale", qui sont des données préformaté avec des attributs uniquement en format string.
Le composant cleaner tente de transformer ces "tuplesale" en tuplepropre", qui est un objet très similaire, mais avec un typage mieu défini en int pour la plupart des attributs, ainsi que l'attribut "typebien" qui va devenir un objet "BienType" appartement ou maison.

Le but de ce composant cleaner est donc de s'assuré que la donné récupéré est propre et formaté comme un "tuplepropre" , et de ne pas prendre en compte les données mal formaté.

Pour écrire mon premier test sur le composant cleaner, je me suis aider du test_provider (de la session3) ainsi que tu test d'exemple test_eq qui donne un exemple de comparaison entre 2 tuples propre:

```
def test_eq() -> None:
    assert TuplePropre(
        type_bien=BienType.Maison, prix=100, surface=200, pieces=6
    ) == TuplePropre(type_bien=BienType.Maison, prix=100, surface=200,
pieces=6)
```

Mon premier test consiste a verifier que un tuplesale bien formaté qui est transformé par le cleaner ressorte bien en tuplepropre.

```
def test_cleaner_good_tuplesale(cleaner: Cleaner) -> None:
    tuplesale =
TupleSale(surface=10,pieces=1,prix=10,type_bien="appartement")
    tupleclean = cleaner.clean([tuplesale])
    assert [TuplePropre(type_bien=BienType.Appartement, prix=10,
surface=10, pieces=1)] == tupleclean
```

Mon second test consiste a donner une valeur de type_bien qui ne respecte pas les attente pour devenir un tuplepropre(valeur qui n'est pas "appartement" ou "maison"). On vérifie ensuite que le résultat du cleaner n'est pas un tuplepropre.

```
def test_cleaner_bad_insert(cleaner: Cleaner) -> None:
    tuplesale =
TupleSale(surface=10,pieces=1,prix=10,type_bien="bas_insert")
    tupleclean = cleaner.clean([tuplesale])
    assert [TuplePropre(type_bien=BienType.Appartement, prix=10,
surface=10, pieces=1)] != tupleclean
```

Mon troisième test consiste a donner un tuplesale bien formaté au cleaner, mais sans le donner en format de tableau. On vérifie ensuite que nous avons bien un message d'erreur.

```
def test_cleaner_not_receive_table(cleaner: Cleaner) -> None:
    tuplesale =
TupleSale(surface=10,pieces=1,prix=10,type_bien="appartement")
    with pytest.raises(TypeError, match="'TupleSale' object is not
iterable"):
        cleaner.clean(tuplesale)
```

Mon quatrième test consiste a donner au cleaner un tuplesale qui contient un string a la place d'un int. On vérifie ensuite que le cleaner ne revois pas un tuplepropre.

```
def test_cleaner_string_parameter_instead_of_int(cleaner: Cleaner) ->
None:
    tuplesale =
TupleSale(surface="string",pieces=1,prix=10,type_bien="appartement")
    tupleclean = cleaner.clean([tuplesale])
    assert tupleclean == []
```

Faites en sorte que seul le meilleur modèle soit utilisé par le pipeline

Lorsque nous utilisons la pipeline en réalisant la commande "python src/main.py train", la fonction pipeline.get_data_and_train_model() est lancé.

Cette fonction va créer un modèle, l'entraîner, puis sauvegarder le modèle et ces metadada dans un fichier pkl.
Ensuite, le composant "saver" va enregistrer ce modèle dans un fichier json "comparatif.json" simplement en l'ajoutant à la liste.
Le score de cohérence du modèle entraîné n'est donc pas calculé ni sauvegardé.

Afin de pouvoir classer les modèles entraînés en fonction de leurs performances, et de choisir le modèle qui a les meilleures performances lors de l'appel de la fonction "predict", il va falloir faire des modifications au code existant.

Ces modifications consiste à réalisé un score de cohérence par le modèle, le sauvegarder, puis s'en servir lors de l'appel de la fonction "predict"

La première étape est de ne pas donner 100% de la donnée au modèle, mais de fournir 90% des données en test, puis 10% de la donnée pour tester le modèle.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

On évalue ensuite le modèle avec la fonction mean_quared_error, puis on insère ce résultat directement dans le Metadata.

```
pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        print("mse", mse)

        return Model(pipeline=pipeline,
metadata=Metadata(name=training_id, score=mse))
```

Cela nous permet de sauvegarder dans le fichier "comparatif.json" le nom des modèles ainsi que le score de chacun d'entre eux.
De nombreuses modifications sur le composant "InMemoryModelComparator" et le composant "InMemoryModelComparator" ont dû être établies pour que ces modèles puissent lire et écrire sur le fichier comparatif.json avec des Metadata contenant le score.

exemples de Metadata du nouveau fichier "comparatif.json"


```
[{"name": "intersecting-misery-paddled-unimpressively", "score":
1322348505.9486446}, {"name": "dishonest-earnings-exist-continually",
"score": 1586510169.8821278}]
```

Par la suite, lors du lancement de la fonction predict, le programme va aller chercher les données du fichier “comparatif.json”, filtrer sur le programme avec le score le plus intéressant dans la fonction “get_best_model” du composant “InMemoryModelComparator”, puis va utiliser le modèle avec le score le plus intéressant (**dans notre cas, celui avec le score d’erreur le plus bas**).

```
def get_best_model(self) -> Metadata:
        if not self.metadata_list:
            raise ValueError("No models have been added.")
        self.metadata_list = sorted(self.metadata_list, key=lambda
meta: meta.score, reverse=True)
        print("BEST MODEL IS ", self.metadata_list[-1].name)
        return self.metadata_list[-1]
```

Rajoutez des logs sur les différents composants pour garder une trace explicite des ux de données, les taux de transformations

Un logger est déja installé dans le programme. Je vais donc me servir de ce programme pour réaliser des logs afin de suivre le comportement du programme durant la phase d’entrainement et la phase de prédiction.

Commençons par les logs de debug permettant de vérifier que les instances sont correctement créé au lancement du programme

```
    logging.debug("Instantiating JsonProvider")
    provider = JsonProvider("data/input/prix.json", logger)
    logging.debug("Instantiating Cleaner")
    cleaner = Cleaner(logger)
    logging.debug("Instantiating ScikitLearnTrainer")
    trainer = ScikitLearnTrainer(test_size=0.2, random_state=50,
logger=logger)
    logging.debug("Instantiating ScikitModelSaver")
    saver = ScikitModelSaver(save_path="data/output/models", logger=logger)
    logging.debug("Instantiating FilesystemModelComparator")
    comparator = FilesystemModelComparator(
        wrapped=InMemoryModelComparator(),
json_path="data/output/comparatif.json", logger=logger
    )
    logging.debug("Instantiating Pipeline")
```

Ensuite, nous devons nous occuper des logs de la fonction d'entrainement du modèle(train). J'entoure la fonction "train" de logs permettant de savoir quand la fonction d'entrainement commence et quand elle s'arrête.

```
logger.info(f"Start Train Function")
logger.info(f"Done Train Function")
```

Je réalise un log permettant de savoir combien le cleaner a nettoyé et conservé de données par rapport à la données d'entrée

```
self.logger.info(f"Cleaner to keep {len(cleaned_data)} pieces of data out of {len(input_data)}")
```

Je rajoute un score au modèle qui s'est entraîné et qui s'est évalué afin d'afficher le résultat de cette évaluation.

```
self.logger.info(f"The model {training_id} has a score of {mse}")
```

Dans le composant "saver" sur la fonction "save", on indique que le modèle a bien était sauvegarder en fichier .pkl

```
self.logger.info(f"Save Model {model.metadata.name} on .pkl file done")
```

On se concentre ensuite sur les logs de la fonction de prédiction du modèle le plus performant.

J'entoure la fonction "predict" de logs permettant de savoir quand la fonction d'entrainement commence et quand elle s'arrête.

```
logger.info(f"Start Predict Function")
logger.info(f"Done Predict Function")
```

J'affiche le modèle le plus performant qui a été choisi

```
logger.info(f"Best Model choose is {best_models_metadata.name}")
```

Enfin, je modifie l'affichage du résultat pour que celui ci soit dans un logger et pas dans un print

```
logger.info(f"Prediction for {input} is {output}")
```

Voici le résultat final du terminal de la fonction “train”:

```
2024-12-10 20:34:39,648 INFO: logging with configuration: var/app.log at level: DEBUG
2024-12-10 20:34:39,648 DEBUG: starting application
2024-12-10 20:34:39,648 DEBUG: Instantiating JsonProvider
2024-12-10 20:34:39,648 DEBUG: Instantiating Cleaner
2024-12-10 20:34:39,648 DEBUG: Instantiating ScikitLearnTrainer
2024-12-10 20:34:39,648 DEBUG: Instantiating ScikitModelSaver
2024-12-10 20:34:39,648 DEBUG: Instantiating FilesystemModelComparator
2024-12-10 20:34:39,648 DEBUG: Instantiating Pipeline
2024-12-10 20:34:39,663 INFO: Start Train Function
2024-12-10 20:34:39,727 INFO: Cleaner to keep 11903 pieces of data out of 12976
2024-12-10 20:34:39,727 INFO: training model warmhearted-cougar-rotted-dimly
2024-12-10 20:34:39,759 INFO: The model warmhearted-cougar-rotted-dimly has a score of 1906392051.6455538
2024-12-10 20:34:39,765 INFO: Save Model warmhearted-cougar-rotted-dimly on .pkl file done
2024-12-10 20:34:39,766 INFO: Done Train Function
2024-12-10 20:34:39,768 DEBUG: ending application
```

Voici le résultat final du terminal de la fonction “predict”:

```
2024-12-10 20:39:06,516 INFO: logging with configuration: var/app.log at level: DEBUG
2024-12-10 20:39:06,516 DEBUG: starting application
2024-12-10 20:39:06,516 DEBUG: Instantiating JsonProvider
2024-12-10 20:39:06,516 DEBUG: Instantiating Cleaner
2024-12-10 20:39:06,516 DEBUG: Instantiating ScikitLearnTrainer
2024-12-10 20:39:06,516 DEBUG: Instantiating ScikitModelSaver
2024-12-10 20:39:06,516 DEBUG: Instantiating FilesystemModelComparator
2024-12-10 20:39:06,516 DEBUG: Instantiating Pipeline
2024-12-10 20:39:06,516 INFO: Start Predict Function
2024-12-10 20:39:06,516 INFO: Best Model choose is enlarged-envoy-knocked-far
2024-12-10 20:39:06,532 INFO: Done Predict Function
2024-12-10 20:39:06,532 INFO: Prediction for TupleSansPrix(surface: 138, pieces: 2, type: BienType.Appartement) is 55178.86199739118
2024-12-10 20:39:06,532 INFO: Prediction for TupleSansPrix(surface: 138, pieces: 3, type: BienType.Appartement) is 55246.88566179709
2024-12-10 20:39:06,532 INFO: Prediction for TupleSansPrix(surface: 138, pieces: 2, type: BienType.Maison) is 66161.22596062627
2024-12-10 20:39:06,532 INFO: Prediction for TupleSansPrix(surface: 138, pieces: 3, type: BienType.Maison) is 66229.24962503219
2024-12-10 20:39:06,532 DEBUG: ending application
```

Vérifiez que la distribution de donnée que vous récupérez est homogène avec un snapshot préalable (Pour garantir une stabilité des données dans le temps)

Durant la génération du train et test lors de l'entrainement de mon modèle dans la fonction "train", je rajoute le paramètre "random_state=42".
Cela me permet de m'assurer que la répartition des données allant vers le train et les données allant vers le test sont toujours les mêmes.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42)
```

Comme nous pouvons le voir dans le fichier "comparatif.json", cela donne le même score a chaque fois que l'on entraîne un nouveau modèle, simplement car un modèle similaire avec une donnée similaire s'entrainera de la même manière.


```
[{"name": "ambient-staging-expected-evenly", "score":
78751115142.9513}, {"name":
"unidentified-juncture-snored-victoriously", "score":
78751115142.9513}, {"name":
"logical-balls-thank-boastfully", "score":
78751115142.9513}]
```


Afin de rendre le TP plus pertinent, je vais laisser commenter cette variable "random_state=42", afin que la selection du modèle le plus pertinent fonctionne correctement avec des scores différents.

Je vous invite maintenant à tester la programme d'entrainement et d'utilisation du meilleur modèle avec le meilleur score en utilisant les commandes suivantes :

"python src/main.py train" (lancer 3 fois par exemple)
"python src/main.py predict" (lancer 1 fois par exemple)