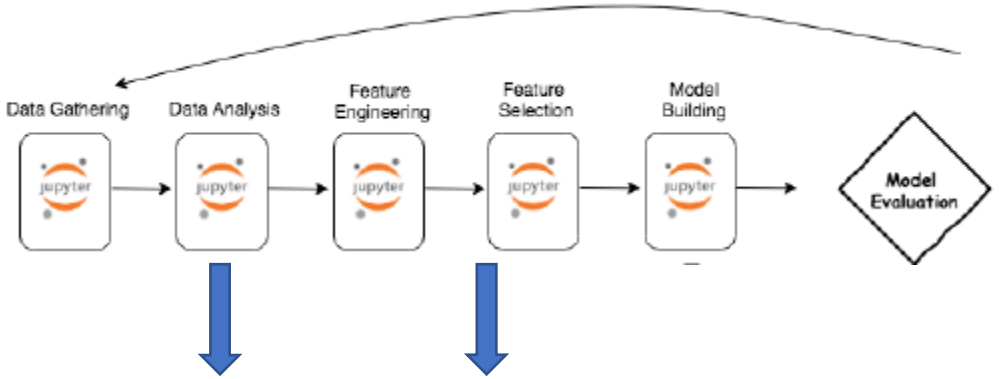
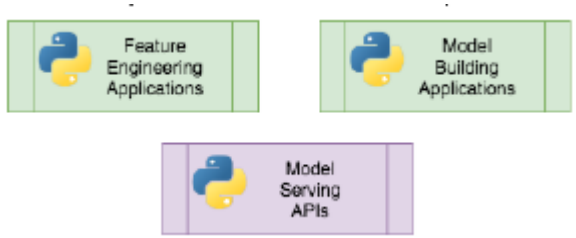
Architecture
Productibilité
Code prêt pour la production

Architecture Système ML-Étapes

Env. de recherche/dev01



Environnement de développement



Intégration continue(CI) Déploiement de pipeline

Test

Test

Env-production

HEROKU

Platform as a Service

aws

Infrastructure as a Service

docker

Suite-Pourquoi sélectionnons-nous des fonctionnalités?

- Les modèles sont plus faciles à interpréter
- Temps d'entrainement plus courts
- Généralisation améliorée en réduisant le surajustement
- Plus facile à mettre en œuvre par les développeurs de logiciels
  - Production de modèles
- Réduction du risque d'erreurs de données lors de l'utilisation du modèle
- Redondance des données

Suite-Réduction des fonctionnalités pour le déploiement de modèles

- Messages json plus petits envoyés au modèle
  - Messages Json ne contient que les variables / entrées nécessaires
- Moins de lignes de code pour la gestion des erreurs
- Des gestionnaires d'erreurs doivent être écrits pour chaque variable / entrée
- Moins d'informations à consigner
- Moins de code d'ingénierie des fonctionnalités

Architecture

- L'architecture du système ML fait référence à la manière dont les composants logiciels qui constituent le système sont agencés et à l'interaction entre eux pour atteindre un objectif prédéfini. Le développement et le déploiement de modèles ML sont relativement rapides, mais leur maintien efficace dans le temps est difficile et coûteux.
- Choisir la meilleure architecture peut être un peu difficile car elle nécessite la synchronisation de l'entreprise, de la science des données, de l'ingénierie et du DevOps.

Suite-Architecture

- L'apprentissage automatique en production nécessite plusieurs différents composants pour fonctionner:
  - Infrastructure
  - Applications
  - Données
  - Documentation
  - Configuration et plus encore.

➢ Ensemble, ces pièces forment le système global.

Celles-ci vont des simples applications Web aux pipelines incroyablement complexes construits par des centaines de personnes.

Défis et principes clés des systèmes ML

Le besoin de reproductibilité (versionnage partout)

- Dépendances des données
- Problèmes de configuration
- Préparation des données et des fonctionnalités
- Les erreurs de modèle peuvent être difficiles à détecter avec les tests traditionnels
- Séparation des compétences

Contributeurs du système ML

- L'architecture d'un système d'apprentissage automatique de production doit prendre en compte les exigences de l'entreprise, ainsi que les défis uniques à l'intersection de la science des données, du génie logiciel et des devops.

Principes clés de l'architecture du système ML

- Lors de la conception d'une architecture de système ML, les points suivants doivent être gardés à l'esprit.

Principes clés de l'architecture du système ML

- **Reproductibilité**: possibilité de répliquer une prédiction ML donnée, la sortie de chaque composant doit être réplicable pour n'importe quelle version dans le temps.
- **Automatisation**: éliminer les étapes manuelles dans la mesure du possible pour réduire les risques d'erreur.
- **Extensibilité**: avez la possibilité d'ajouter et de mettre à jour facilement des modèles. il devrait être facile à modifier pour les tâches futures.
- **Modularité**: le code de prétraitement / d'ingénierie des fonctionnalités utilisé dans la formation doit être organisé en pipelines clairs et complets.
- **Évolutivité**: le modèle doit être utilisable par un grand nombre de clients avec un temps de réponse minimal.
- Test: la possibilité de tester la variation entre les versions du modèle.

Vue d'ensemble des architectures ML

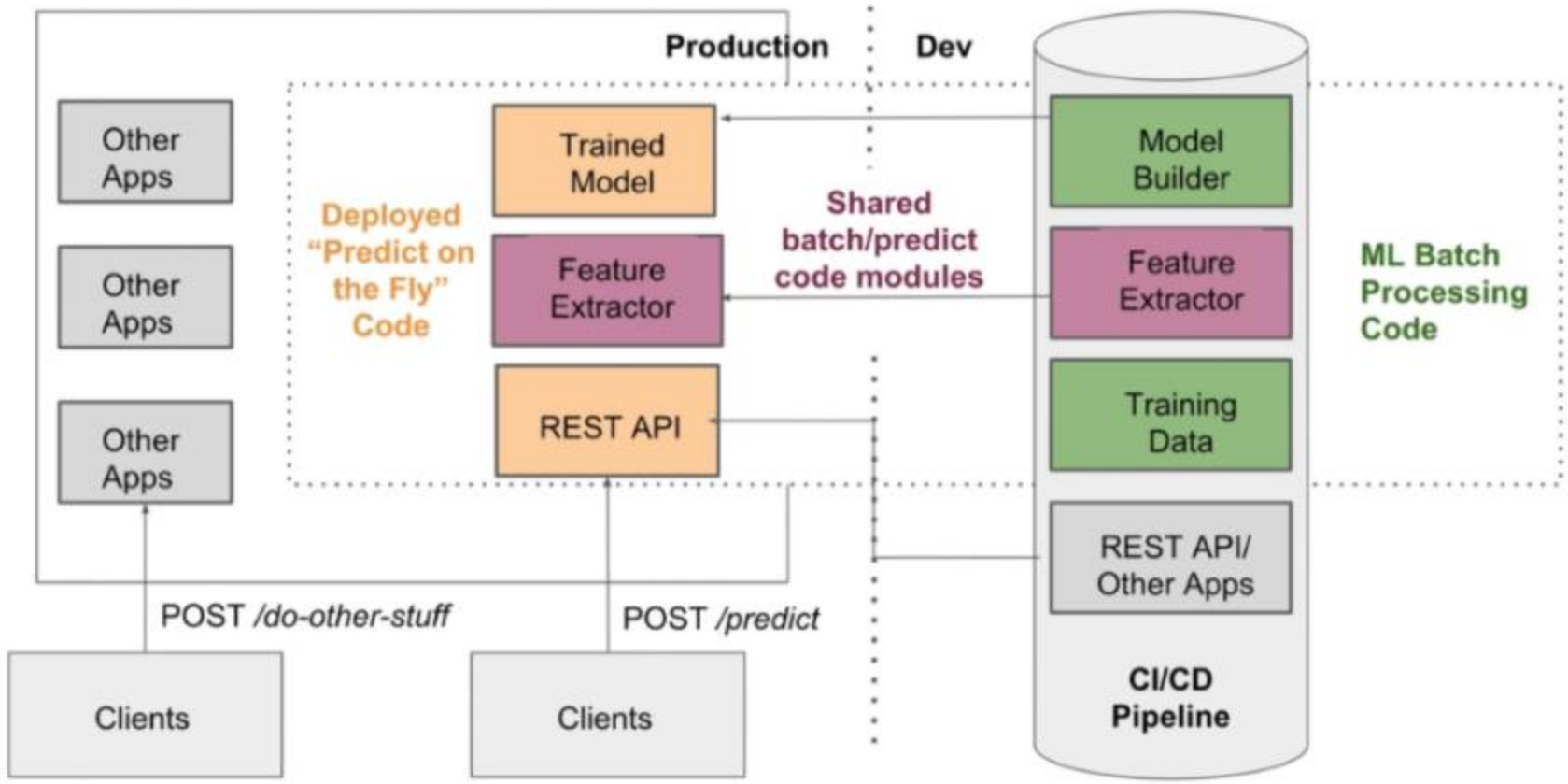
- **Entraîner par lots, prédire à la volée, servir via l'API REST** : l'entraînement et la persistance se font hors ligne tandis que la prédiction se fait en temps réel.
- **Entraîner par lot, prédire par lot, servir via une base de données partagée** : l'entraînement et la persistance se font hors ligne tandis que les prédictions sont effectuées dans une file d'attente distribuée qui est presque similaire à une prédiction en temps réel
- **Entraîner, prédire en streaming** : l'entraînement et la prédiction se font sur des flux différents mais connectés.
- **Entraîner par lots, prédire sur mobile (ou autres clients)** : similaire au type 1 mais la prédiction se fait sur le gadget client.

Comparaison d'architecture/Pattern

	Pattern 1 (REST API)	Pattern 2 (Shared DB)	Pattern 3 (Streaming)	Pattern 4 (Mobile App)
Training	Batch	Batch	Streaming	Streaming
Prediction	On the fly	Batch	Streaming	On the fly
Prediction result delivery	Via REST API	Through the shared DB	Streaming via Message Queue	Via in-process API on mobile
Latency for prediction	So so	High	Very Low	Low
System Management Difficulty	So so	Easy	Very Hard	So so

Diagramme:

Former par lots, prédire à la volée, servir via l'API REST



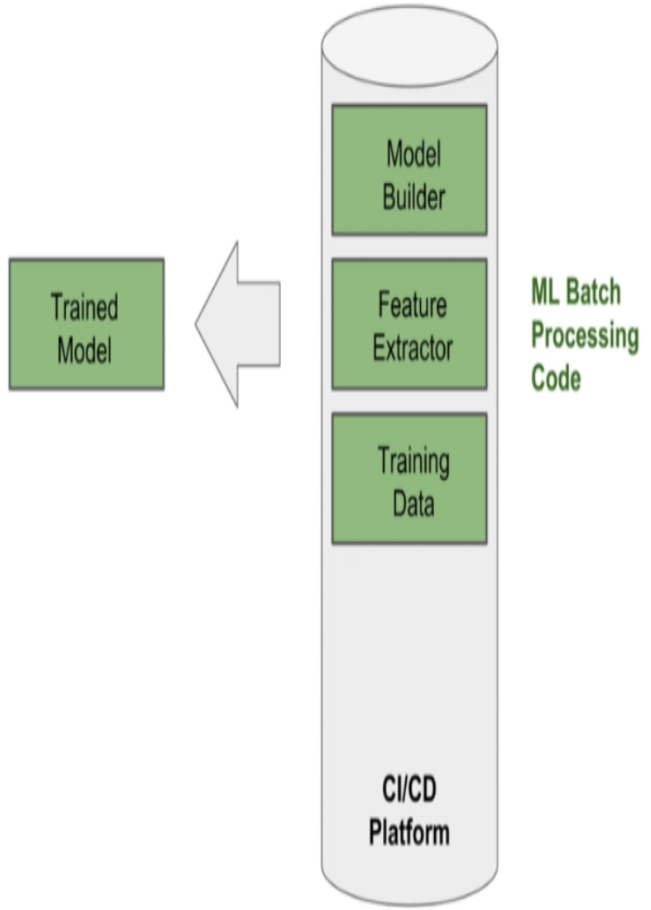
Suite-

- Dans la figure ci-dessus, on peut voir que l'ensemble du système est divisé en développement et production. dans le développement, la section Données de formation, Extracteur de fonctionnalités et Générateur de modèles se fait hors ligne alors que la production est en ligne.

Suite-Phase d'entraînement

- **Données de formation** : cette unité traite de l'extraction de l'ensemble de données à utiliser pour entraîner le modèle dans l'environnement de formation et de la mise en forme pour la formation. cette unité peut être complexe ou simple en fonction de l'architecture ML sur laquelle vous travaillez.Cela peut inclure l'extraction de données à partir de plusieurs séquences, le système de fichiers distribué Hadoop ou faire des appels API
- **Extracteur de fonctionnalités** : c'est l'unité qui sélectionne / génère les fonctionnalités essentielles nécessaires à la formation du modèle

➢ une autre bibliothèque populaire qui peut être utilisée pour accomplir la même tâche est TensorFlow Extended (TFX)

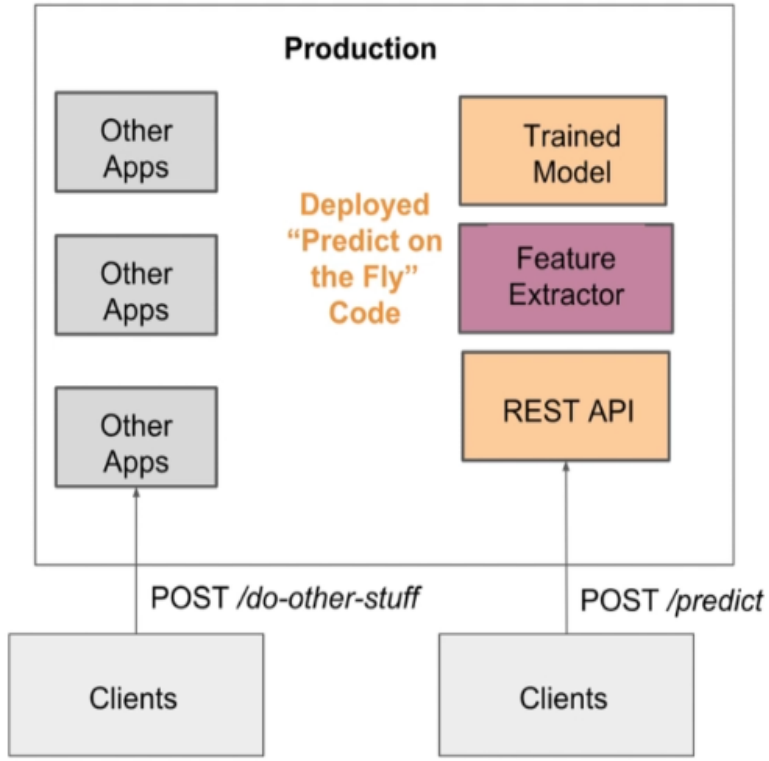


Suite-

- **Model Builder:** c'est l'unité où la sérialisation et la persistance des modèles entraînés, la gestion des versions et le formatage pour le déploiement. Dans le contexte python, cela peut être l'unité qui implique la création de packages à l'aide de setuptools ou simplement l'enregistrement du modèle dans un format pickle en fonction de l'architecture que vous adoptez.
- **Modèle Entrainé** : il s'agit du résultat des trois sous-sujets abordés ci-dessus qui peuvent être facilement déployés via l'API REST.

Suite- Production

- En production, une demande est envoyée via l'API REST par lots, nettoyée et prétraitée par l'extracteur de fonctionnalités, puis le modèle entraîné chargé est utilisé pour la prédiction.



Récap

- Il existe de nombreuses façons de déployer des modèles ML en production, de nombreuses façons de les stocker et différentes façons de gérer les modèles prédictifs après le déploiement. choisir la meilleure approche pour un cas d'utilisation peut être difficile, mais avec la maturité technique et analytique de l'équipe, la structure organisationnelle globale et ses interactions peuvent aider à sélectionner la bonne approche pour le déploiement de modèles prédictifs en production.

Convertir le notebook en code prêt pour la production

convertir le notebook en code prêt pour la production

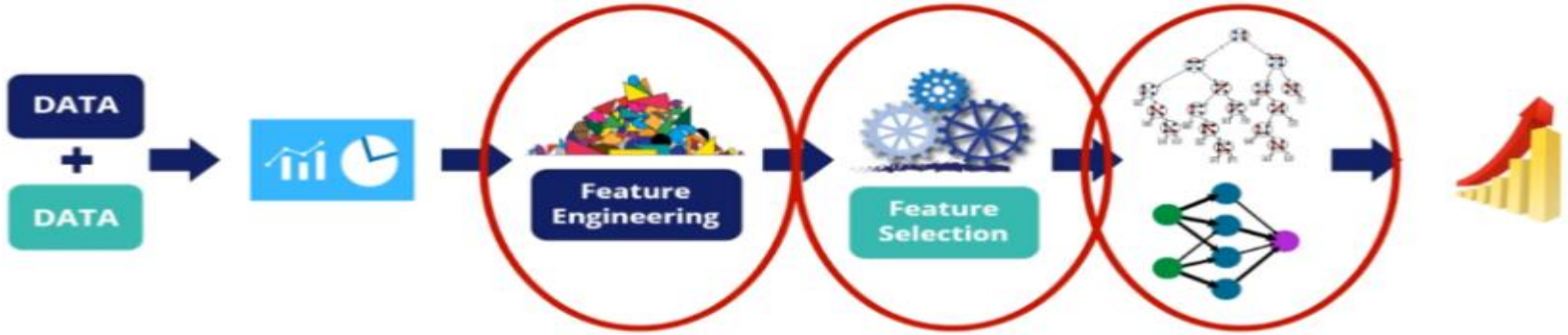
- Le premier objectif ici est de convertir le notebook en code prêt pour la production qui n'est rien d'autre que l'écriture de scripts et de modules python.

Suite-

- Dans le cadre de la configuration requise lors de l'écriture du code de production, il est recommandé de toujours avoir un environnement virtuel différent pour chaque projet afin d'éviter les problèmes de dépendances.

Quelle partie entre dans le code de production?

- La question suivante est de savoir quelle partie du pipeline d'apprentissage automatique entre dans le code de production?



- Seules les parties encerclées du pipeline doivent être converties en code de production.

Pipeline reproductible

Il existe 3 méthodes principales pour écrire du code de déploiement pour le ML:

- **Programmation procédurale**: Elle consiste à écrire une séquence de fonctions exactement comme celles écrites dans le cahier. ces fonctions sont ensuite appelées dans une séquence jusqu'à ce que la dernière fonction soit exécutée.

```
def load_data(df_path):
    # Function loads data for training
    return pd.read_csv(df_path)


def divide_train_test(df, target):
    # Function divides data set in train and test
    X_train, X_test, y_train, y_test = train_test_split(df,
                                                        df[target],
                                                        test_size=0.1,
                                                        random_state=0)
    return X_train, X_test, y_train, y_test


def impute_na(df, var, replacement='Missing'):
    # function replaces NA by value entered by user
    # or by string Missing (default behaviour)
    return df[var].fillna(replacement)
```

Programmation procédurale: script de train

```
# Load data
data = pf.load_data(config.PATH_TO_DATASET)

# divide data set
X_train, X_test, y_train, y_test = pf.divide_train_test(data, config.TARGET)

# impute categorical variables
for var in config.CATEGORICAL_TO_IMPUTE:
    X_train[var] = pf.impute_na(X_train, var, replacement='Missing')


# impute numerical variable
X_train[config.NUMERICAL_TO_IMPUTE] = pf.impute_na(X_train,
        config.NUMERICAL_TO_IMPUTE,
        replacement=config.LOTFRONTAGE_MODE)


# capture elapsed time
X_train[config.YEAR_VARIABLE] = pf.elapsed_years(X_train,
        config.YEAR_VARIABLE, ref_var='YrSold')
```

Programmation procédurale: fichier yaml(config)

```
# ====   PATHS ===================

PATH_TO_DATASET = "train.csv"
OUTPUT_SCALER_PATH = 'scaler.pkl'
OUTPUT_MODEL_PATH = 'lasso_regression.pkl'


# ======= PARAMETERS ===============

# imputation parameters
LOTFRONTAGE_MODE = 60


# encoding parameters
FREQUENT_LABELS = {
    'MSZoning': ['FV', 'RH', 'RL', 'RM'],
    'Neighborhood': ['Blmngtn', 'BrDale', 'BrkSide', 'ClearCr', 'CollgCr',
                     'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV',
                     'Mitchel', 'NAmes', 'NWAmes', 'NoRidge', 'NridgHt',
                     'OldTown', 'SWISU', 'Sawyer', 'SawyerW', 'Somerst',
                     'StoneBr', 'Timber'],
    'RoofStyle': ['Gable', 'Hip'],
    'MasVnrType': ['BrkFace', 'None', 'Stone'],
    'BsmtQual': ['Ex', 'Fa', 'Gd', 'Missing', 'TA'],
    'BsmtExposure': ['Av', 'Gd', 'Missing', 'Mn', 'No'],
    'HeatingQC': ['Ex', 'Fa', 'Gd', 'TA'],
    'CentralAir': ['N', 'Y'],
```

```
CATEGORICAL_TO_IMPUTE = ['MasVnrType', 'BsmtQual', 'BsmtExposure',
                         'FireplaceQu', 'GarageType', 'GarageFinish']
NUMERICAL_TO_IMPUTE = 'LotFrontage'

YEAR_VARIABLE = 'YearRemodAdd'

# variables to transofmr
NUMERICAL_LOG = ['LotFrontage', '1stFlrSF', 'GrLivArea', 'SalePrice']


# variables to encode
CATEGORICAL_ENCODE = ['MSZoning', 'Neighborhood', 'RoofStyle',
                      'MasVnrType', 'BsmtQual', 'BsmtExposure',
                      'HeatingQC', 'CentralAir', 'KitchenQual',
                      'FireplaceQu', 'GarageType', 'GarageFinish',
                      'PavedDrive']


# selected features for training
FEATURES = ['MSSubClass', 'MSZoning', 'Neighborhood', 'OverallQual',
            'OverallCond', 'YearRemodAdd', 'RoofStyle', 'MasVnrType',
            'BsmtQual', 'BsmtExposure', 'HeatingQC', 'CentralAir',
            '1stFlrSF', 'GrLivArea', 'BsmtFullBath', 'KitchenQual',
            'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageFinish',
            'GarageCars', 'PavedDrive', 'LotFrontage']
```

Suite- Programmation procédurale -Récap

(+)

- Simple depuis le notebook
- Aucune compétence en développement logiciel requise
- Facile à vérifier manuellement

(-)

- valeurs codées en dur
- paramètres de code en dur
- enregistrer plusieurs objets ou structures de données
- Difficile à tester
- Nécessité de sauvegarder de nombreux fichiers intermédiaires pour stocker les paramètres de transformation

Suite- Pipeline reproductible

- **Code de pipeline personnalisé**: il est similaire au pipeline tiers, car les instances sont exécutées à l'aide d'un pipeline personnalisé. Le pipeline personnalisé peut être créé à l'aide d'un pipeline tiers comme infrastructure de base, tandis que les personnalisations sont effectuées pour atteindre les objectifs ciblés.

Pipeline Machine Learning personnalisé

- Un pipeline Machine Learning personnalisé est donc une séquence d'étapes, visant à charger et transformer les données, pour les préparer pour la formation ou la notation, où:
  - Nous écrivons les étapes de traitement sous forme d'objets (POO)
  - Nous écrivons la séquence, c'est-à-dire le pipeline sous forme d'objets (POO)
  - Nous sauvegardons un objet, le pipeline, sous forme de pickle Python, avec toutes les informations nécessaires pour transformer les entrées brutes et obtenir les prédictions

Suite-Pipeline ML personnalisé

Programmation orientée objet - POO

- En programmation orientée objet (POO), nous écrivons du code sous la forme d '«objets».
- Ces «objets» peuvent stocker des données et peuvent également stocker des instructions ou des procédures pour modifier ces données.
- Données ⇒ attributs
- Instructions ou procédures ⇒ méthodes

Suite-Pipeline ML personnalisé: POO

En programmation orientée objet (POO), les «objets» peuvent apprendre et stocker ces paramètres

- Les paramètres sont automatiquement actualisés à chaque réapprentissage du modèle
- Pas besoin de codage manuel en dur
- Méthodes:
  - **Fit**: pour apprendre les paramètres
    - Enregistre le paramètre dans l'attribut d'objet
  - **Transformer**: pour transformer les données avec les paramètres appris
    - Appelle l'attribut object pour rappeler les paramètres
- Attributs: enregistrent les paramètres d'apprentissage

Suite-Pipeline ML personnalisé: Pipeline

- Un pipeline est un ensemble d'étapes de traitement de données connectées en série, où la sortie d'un élément est l'entrée du suivant.

- Les éléments d'un pipeline peuvent être exécutés en parallèle ou en tranches de temps. C’est utile lorsque nous avons besoin d'utiliser des données volumineuses ou une puissance de calcul élevée, par exemple pour les neurones réseaux.

Suite-pipeline Machine Learning personnalisé

(+)

- Peut être testé, versionné, suivi et contrôlé
- Peut construire de futurs modèles
- Bonnes pratiques de développement de logiciels
- Conçu pour répondre aux besoins de l'entreprise

(-)

- Nécessite une équipe de développeurs de logiciels pour créer et maintenir
- Frais généraux permettant à DS de se familiariser avec le code pour le débogage ou l'ajout de futurs modèles
- Le préprocesseur n'est pas réutilisable, il faut réécrire la classe du préprocesseur pour chaque nouveau modèle ML
- Besoin d'écrire un nouveau pipeline pour chaque nouveau modèle ML

Code de pipeline tiers

- **Code de pipeline tiers**: cela implique l'utilisation de la POO et les instances sont exécutées à l'aide d'un pipeline tiers tel que le pipeline sklearn.

¸Suite-Pourquoi Scikit-Learn?

- Scikit-Learn fournit des versions efficaces d'un grand nombred'algorithmes.
- Scikit-Learn se caractérise par une API propre, uniforme et rationalisée.
- Scikit-Learn est écrit de manière à ce que la plupart de ses algorithmes suivent la même fonctionnalité
- Une fois que vous avez compris l'utilisation et la syntaxe de base de Scikit-Learn pour un type de modèle, un nouveau modèle ou algorithme est très simple
- Scikit-Learn fournit une documentation en ligne utile et complète qui vous permet de comprendre
- https://scikit-learn.org/stable/modules/classes.html

Suite-Pipeline Scikit-Learn et sklearn

Objets Scikit-Learn

- **Transformers** - classe dotée d'une méthode d'ajustement(Fit) et de transformation(transform), elle transforme les données
  - Scalers
  - Sélecteurs de fonctionnalités
  - Un encodeur à chaud

```
pipln = Pipeline([("trsfm1",transformer_1),
                  ("trsfm2",transformer_2),
                  ("estmtr",estimator)])
```

- **Predictor** - classe qui a des méthodes d'ajustement et de prédiction, il ajuste et prédit.
  - Tout algorithme ML comme le lasso, les arbres de décision, svm, etc.
- **Pipeline** - classe qui vous permet de répertorier les transformateurs et les prédicteurs d'exécution en séquence
  - Toutes les étapes doivent être des transformateurs sauf la dernière
  - La dernière étape doit être un prédicteur

Liens sklearn

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

- https://scikitlearn.org/stable/modules/generated/sklearn.base.BaseEstimator.html#sklearn.base.BaseEstimator
- https://scikitlearn.org/stable/modules/generated/sklearn.base.TransformerMixin.html#sklearn.base.TransformerMixin

Pipeline Scikit-Learn et sklearn

(+)

- Peut être testé, versionné, suivi et contrôlé
- Peut construire de futurs modèles sur le dessus
- Bonnes pratiques de développement de logiciels
- Exploite la puissance de l'API reconnue
- Scientifiques des données familiers avec l'utilisation de Pipeline, frais généraux réduits
- Les étapes d'ingénierie peuvent être conditionnées et réutilisées dans les futurs modèles ML

(-)

- Nécessite une équipe de développeurs de logiciels pour créer et maintenir
- Frais généraux pour les développeurs de logiciels pour se familiariser avec le code de l'API sklearn ⇒ difficultés de débogage

Classe: Estimator-Transformer- Pipeline

```python
class Estimator(object):

    def fit(self, X, y=None):
        """
        Fits the estimator to data.
        """
        return self

    def predict(self, X):
        """
        Compute the predictions
        """
        return predictions
```

```python
class Transformer(object):

    def fit(self, X, y=None):
        """
        Learn the parameters to
        engineer the features
        """

    def transform(X):
        """
        Transforms the input data
        """
        return X_transformed
```

```python
class Pipeline(Transformer):

    @property
    def name_steps(self):
        """Sequence of transformers
        """
        return self.steps

    @property
    def _final_estimator(self):
        """
        Estimator
        """
        return self.steps[-1]
```

Pipeline: classe qui permet d'exécuter des **transformateurs** et des **estimateurs** en séquence.

- La plupart des étapes sont des transformateurs
- La dernière étape peut être un estimateur

suite

Le pipeline tiers sera adopté pour la suite pour les raisons suivantes:

- Il peut être testé, versionné, suivi et contrôlé. c'est-à-dire que des tests peuvent être effectués, on peut suivre la sortie en fonction de la version utilisée, ce qui permet à son tour de contrôler facilement l'ensemble du flux de travail
- Les futurs modèles peuvent être au top. c'est-à-dire qu'un pipeline construit peut être adapté pour d'autres modèles similaires lors de tests différentiels.

Reproductibilité dans Machine Learning

Qu'est-ce que la reproductibilité dans le Machine Learning?

- La reproductibilité est la capacité de dupliquer exactement un modèle d'apprentissage automatique, de manière à données brutes en entrée, les deux modèles renvoient la même sortie.

Reproductibilité lors de la collecte des données

Les données peuvent être le défi le plus difficile pour assurer la reproductibilité:

- Des problèmes surviennent si le jeu de données d'entraînement ne peut pas être reproduit ultérieurement
  - **Par exemple:**
    - Les bases de données sont constamment mis à jour et écrasé,

    donc des valeurs présentes à un certain point dans le temps diffère des valeurs ultérieurement.
    - L'ordre des données lors du chargement des données est aléatoire, par exemple lors de la récupération les lignes avec SQL.

Suite- Comment assurer la reproductibilité?

Solution

❑ Enregistrer un instantané(snapshot) des données d'entraînement (soit

les données réelles, ou une référence à un emplacement de stockage tel qu'AWS S3

✓ Bon si les données ne sont pas extraites trop de sources différentes

➢ Concevez des sources de données avec horodatages(timestamps), de sorte qu'une vue des données à tout moment peut être récupéré
✓ Situation idéale

➢il faut un gros effort pour reconcevoir les données sources

Reproductibilité lors de l'ingénierie de fonctionnalités

Le manque de reproductibilité peut résulter de:

- Remplacement des données manquantes par des valeurs générés (random)
- Suppression des étiquettes en fonction des pourcentages d'observations
- Calcul de valeurs statistiques comme moyen à utiliser pour remplacer la valeur manquante

Suite- Reproductibilité lors de l'ingénierie de fonctionnalités- solution

- Le code sur la manière dont une fonction est générée doit
- être suivi sous contrôle de version et publié avec incrémentation automatique ou horodatage(timestamp) des versions hachées.
- De nombreux paramètres extraits pour l'ingénierie des fonctionnalités dépend des données utilisé pour la formation ➔ s'assurer que les données sont reproductible
- En cas de remplacement par extraction d'échantillons aléatoires,

toujours mettre- (seed)

Comment assurer la reproductibilité? Sélection de fonctionnalité

- Enregistrez l'ordre des fonctionnalités
- Enregistrer les transformations de caractéristiques appliquées,

par exemple, la normalisation

- Enregistrer les hyperparamètres
- Pour les modèles qui nécessitent un élément de hasard formés (arbres de décision, réseaux de neurones, descentes en gradient), ajoutez toujours seed.

Reproductibilité lors du déploiement du modèle: Environnement logiciel et implémentation

- Pour une reproductibilité totale, les versions du logiciel doivent correspondre exactement

Les applications doivent répertorier toutes les dépendances de bibliothèques tierces et leurs versions.

- Utilisez un conteneur et suivez ses spécifications, telles que la version de l'image (qui

inclure des informations importantes telles que la version du système d'exploitation)

- Recherche, développement et déploiement en utilisant le même langage, par exemple, python
- Avant de construire le modèle, comprenez comment le modèle sera intégré dans production –comment le modèle sera consommé-, afin que vous puissiez vous assurer que le chemin

il a été conçu peut être entièrement intégré

  - Exemples de déploiement partiel: certaines données ne sont pas disponibles le temps de consommation du modèle en direct
  - Les filtres en place ne permettent pas à une certaine cohorte de données d'être vue par le modèle