Deep Learning for dummies book

Capítol 1.3: Machine Learning

Aina Palacios

Ara anem més enllà!

Com ser uns perfeccionistes?

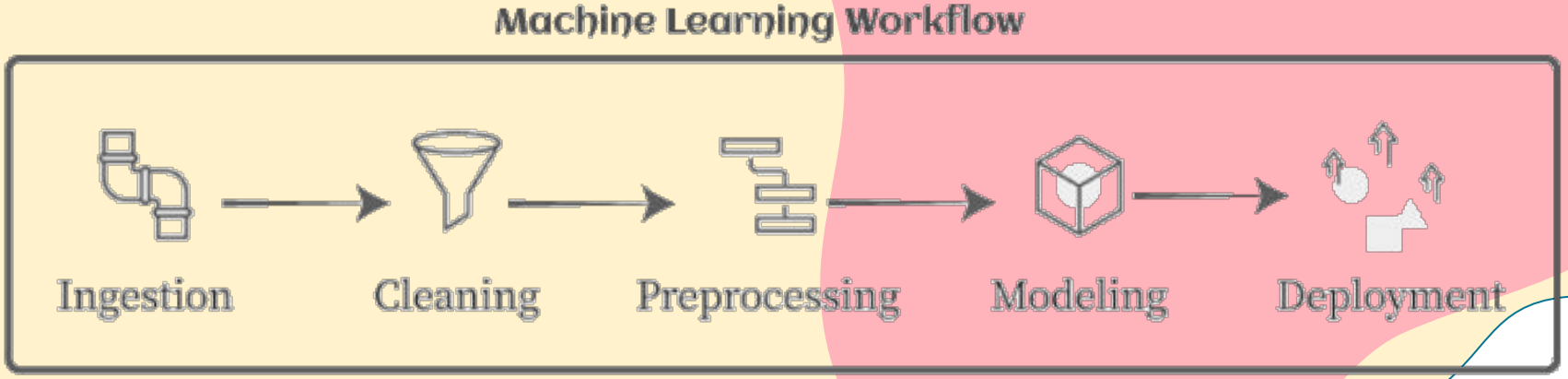
- Crea noves variables per facilitat el resultat!
- A l'hora de fer preprocessat, investiga diferents tècniques per millorar el resultat
- A l'hora d'escollir un model, mira primer quin s'adapta millor a la teva base de dades
- Comprova les teves mètriques utilitzant un test que no hagis tingut en compte en el preprocessat!

Pipeline!

Una manera de codificar i

automatitzar el **workflow** per produir

un model d'aprenentatge automàtic!



Per a què utilitzar Pipeline?

1. Et permet posar els passos de preprocessament en el modelatge

2. T'impedeix fer servir cap test a l'hora d'obtenir les mètriques i avaluar el model!

3. Garanteix que les teves dades siguin preprocessades de la mateixa forma!

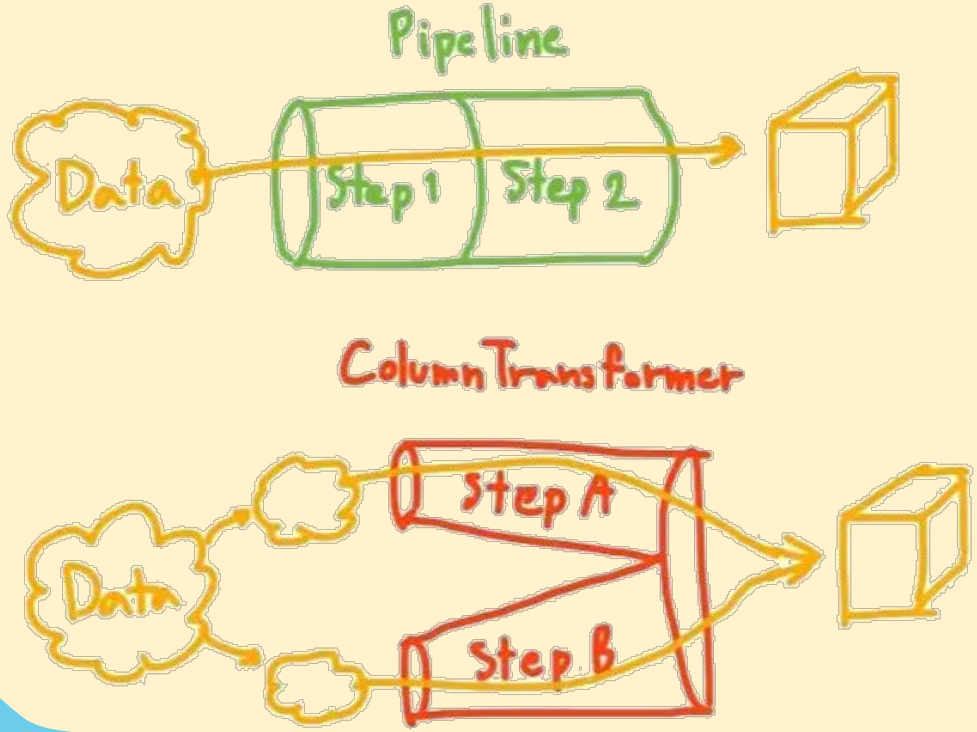
```
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.naive_bayes import MultinomialNB
>>> from sklearn.preprocessing import Binarizer
>>> make_pipeline(Binarizer(), MultinomialNB())
Pipeline(steps=[('binarizer', Binarizer()), ('multinomialnb', MultinomialNB())])
```

Column transformer

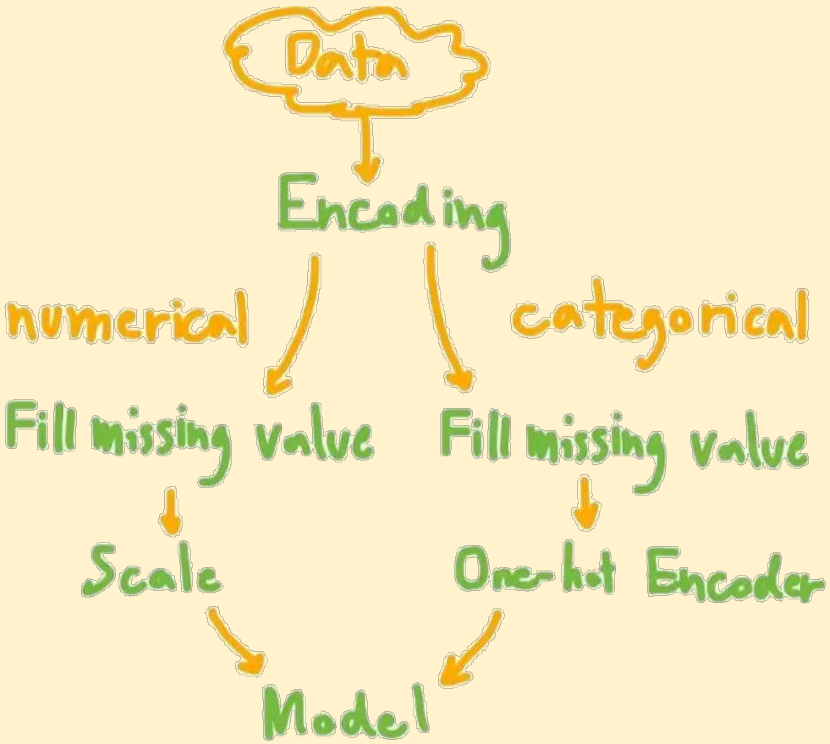
En el preprocessat, no sempre voldrem aplicar el mateix a cada una de les variables, per poder fer-ho, utilitzarem ColumnTransformer!

Aquesta classe ens permet crear Pipelines que aplicarem només a les classes que nosaltres vulguem!

Sempre haurem d'indicar que fer amb els missing values, per evitar errors!



https://medium.com/mlearning-ai/neat-data-preprocessing-with-pipeline-and-columntransformer-2a0468865b6b



1. Definim columnes

```
num_cols = ['city_development_index','relevent_experience',
'experience','last_new_job', 'training_hours']

cat_cols = ['gender', 'enrolled_university', 'education_level',
'major_discipline', 'company_size', 'company_type']
```

2. Creem Pipeline per cada columna

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='mean')),
    ('scale',MinMaxScaler())
])

cat_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='most_frequent')),
    ('one-hot',OneHotEncoder(handle_unknown='ignore', sparse=False))
])
```

3. Apliquem ColumnTransofmer

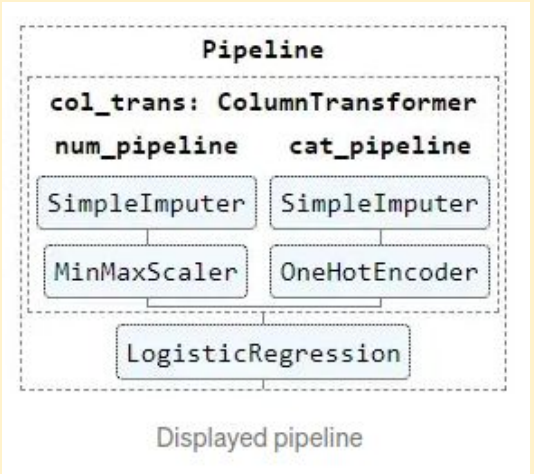
```
from sklearn.compose import ColumnTransformer

col_trans = ColumnTransformer(transformers=[
    ('num_pipeline',num_pipeline,num_cols),
    ('cat_pipeline',cat_pipeline,cat_cols)
    ],
    remainder='drop',
    n_jobs=-1)
```

4. Creem Pipeline final

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0)

clf_pipeline = Pipeline(steps=[
    ('col_trans', col_trans),
    ('model', clf)
])
```

5. Display Pipeline

```
from sklearn import set_config
set_config(display='diagram')

display(clf_pipeline)
```



Displayed pipeline

Fer servir la Pipeline

```
pipeline = Pipeline(
    [
        ('preprocessing', preprocessor),
        ('model', LogisticRegression())
    ]
)

params = {
    'model__solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky'],
    'model__penalty': ['l1', 'l2', 'elasticnet', None],
    'model__C': [0.01, 0.1, 0.5, 1, 2, 10, 100],
    'model__random_state': [42]
}

rskf = RepeatedStratifiedKFold(n_splits = 5, n_repeats = 2, random_state = 42)

cv = GridSearchCV(pipeline, params, cv = rskf, scoring = ['f1', 'accuracy'], refit = 'f1', n_jobs = -1)



cv.fit(X, y)

print(f'Best F1-score: {cv.best_score_:.3f}\n')
print(f'Best parameter set: {cv.best_params_}\n')
print(f'Scores: {classification_report(y, cv.predict(X))}')
```

```
clf_pipeline.fit(X_train, y_train)
# preds = clf_pipeline.predict(X_test)
score = clf_pipeline.score(X_test, y_test)
print(f"Model score: {score}") # accuracy
```

Ja hem acabat la part 3!

Gràcies a tots!