TICS-411 Minería de Datos

Práctico 0: Introducción a Scikit-Learn

Alfonso Tobar-Arancibia
alfonso.tobar.a@edu.uai.cl

UNIVERSIDAD ADOLFO IBÁÑEZ

¿Qué es Scikit-Learn?

- `Scikit-Learn` (`sklearn` para los amigos) es una librería creada por David Cournapeau, como un Google Summer Code Project y luego Matthieu Brucher en su tesis.
- En 2010 queda a cargo de INRIA y tiene un ciclo de actualización de 3 meses.
- Es la librería más famosa y poderosa para hacer Machine Learning hoy en día.
- Su API es tan famosa, que hoy se sabe que una librería es de `calidad` si sigue los estándares implementados por `Scikit-Learn`.
- Para que un algoritmo sea parte de `Scikit-Learn` debe poseer 3 años desde su publicación y 200+ citaciones mostrando su utilidad y amplio uso (ver acá).
- Además es una librería que obliga a que sus algoritmos tengan la capacidad de generalizar.

Diseño

- `Scikit-Learn` sigue un patrón de `Programación Orientada a Objetos (POO)` basado en clases.

- En programación, una clase es un objeto que internamente contiene estados que pueden ir cambiando en el tiempo.
  - Una clase posee:
    - **Métodos**: Funciones que cambian el comportamiento de la clase.
    - **Atributos**: Datos propios de la clase.

`Scikit-Learn` sigue el siguiente estándar:

- Todas las Clases se escriben en `CamelCase`: Ej: `KMeans`,`LogisticRegression`, `StandardScaler`.
- Las clases en Scikit-Learn pueden representar algoritmos, o etapas de un preprocesamiento.
  - Los algoritmos se denominan `Estimators`.
  - Los preprocesamientos se denominan `Transformers`.
- Las funciones se escriben como `snake_case` y permiten realizar algunas operaciones básicas en el proceso de modelamiento. Ej: `train_test_split()`, `cross_val_score()`.
- Normalmente se utilizan letras mayúsculas para denotar `Matrices` o `DataFrames`, mientras que las letras minúsculas denotan `Vectores` o `Series`.

Estimadores No supervisados

```
1 from sklearn.sub_modulo import Estimator
2 model = Estimator(hp1=v1, hp2=v2,...)
3 model.fit(X)
4
5 y_pred = model.predict(X)
6
7 ## Opcionalmente se puede entrenar y predecir a la vez.
8 model.fit_predict(X)
```

L1. Importar la clase a utilizar.

L2. `Instanciar` el modelo y sus `hiperparámetros`.

L3. `Entrenar` o ajustar el modelo (Requiere sólo de X).

L5. `Predecir`. Los modelos de clasificación tienen la capacidad de generar probabilidades.

L7-8. Este tipo de modelos permite entrenar y predecir en un sólo paso.

Estimadores Predictivos

```
1 from sklearn.sub_modulo import Estimator
2 model = Estimator(hp1=v1, hp2=v2,...)
3 model.fit(X_train, y_train)
4
5 y_pred = model.predict(X_test)
6 y_pred_proba = model.predict_proba(X_test)
7
8 model.score(X_test,y_test)
```

L1. Importar la clase a utilizar.

L2. `Instanciar` el modelo y sus `hiperparámetros`.

L3. `Entrenar` o ajustar el modelo (Ojo, requiere de `X` e `y`).

L5–6. `Predecir` en datos nuevos. (Algunos modelos pueden predecir probabilidades).

L8. `Evaluar` el modelo en los datos nuevos.

Output de un Modelo

- Los modelos no entregan directamente un output sino que los dejan almacenados en su interior como un estado.
- Los Estimators tienen dos estados:
  - **Not Fitted**: Modelo antes de ser entrenado
  - **Fitted**: Una vez que el modelo ya está entrenado. (Después de aplicar `.fit()`)

Muchos modelos pueden entregar información sólo luego de ser entrenados (su atributo termina con un _).

Ej: `model.coef_`, `model.intercept_`.

El modelo es una herramienta a la cual le entregamos datos (Input), y nos devuelve datos (Predicciones).

Transformers

- A diferencia de los `Estimators`, los `Transformers` no son modelos.
- Su input y su output son datos.
- Algunos `Transformers` permiten escalar los datos, transformar categorías en números, rellenar valores faltantes. (Veremos más acerca de esto en los `Preprocesamiento`).

```
1 from sklearn.preprocessing import Transformer
2 tr = Transformer(hp1=v1, hp2=v2,...)
3 tr.fit(X)
4
5 X_new = tr.transform(X)
6
7 ## Opcionalmente
8 X_new = tr.fit_transform(X)
```

L1. Importar la clase a utilizar (en este caso del submodulo `preprocessing`, aunque pueden haber otros como `impute`).

L2. `Instanciar` el Transformer y sus `hiperparámetros`.

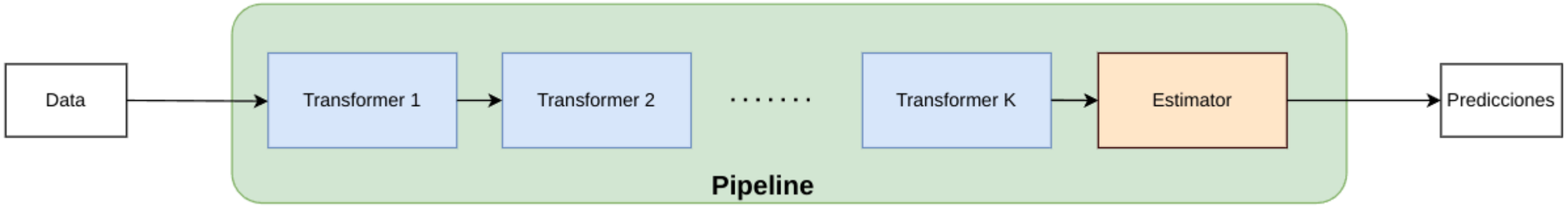
L3. `Entrenar` o ajustar el Transformer.

L5. `Transformar` los datos.

L7-8. Adicionalmente se puede `entrenar` y `transformar` los datos en un sólo paso.

Pipelines

- En ocasiones un Dataset requiere más de un preprocesamiento.
- Estas Transformaciones normalmente se hacen en serie de manera consecutiva.



- El Estimator es opcional, es decir, el Pipeline puede ser para combinar sólo `Transformers` o `Transformers + un Estimator`.

Un Pipeline puede tener **sólo un Estimator**.

Pipelines: Código

```
1  from sklearn.tree import DecisionTreeClassifier
2  from sklearn.preprocessing import StandardScaler, OneHotEncoder
3  from sklearn.pipeline import Pipeline
4
5  pipe = Pipeline(steps=[
6      ("ohe", OneHotEncoder()),
7      ("sc", StandardScaler()),
8      ("model", DecisionTreeClassifier())
9  ])
10
11 pipe.fit(X_train, y_train)
12 y_pred = pipe.predict(X_test)
13
14 pipe.score(X_test, y_test)
```

L1-2. Importo mi modelo y mis preprocesamientos

L3. Importo el `Pipeline`.

L5-9. Instancio un `Pipeline`.

L11. Entreno el `Pipeline`.

L12. Predigo utilizando el `Pipeline` entrenado.

L14. Evalúo el modelo en datos no vistos.

Documentación

> Probablemente `Scikit-Learn` tenga una de las mejores documentaciones existentes.

- Veamos el caso de la Documentación del One Hot Encoder

sklearn.preprocessing.OneHotEncoder

class sklearn.preprocessing.**OneHotEncoder**(*, *categories='auto'*, *drop=None*, *sparse_output=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*, *min_frequency=None*, *max_categories=None*, *feature_name_combiner='concat'*) [source]

Encode categorical features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the `sparse_output` parameter).

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the `categories` manually.

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

Note: a one-hot encoding of y labels should use a LabelBinarizer instead.

Read more in the User Guide. For a comparison of different encoders, refer to: Comparing Target

Preguntas para terminar

- ¿Cómo se importan las clases en Scikit-Learn?
- ¿Cuál es la diferencia entre un Transformer y un Estimator?
- ¿Cuándo es buena idea usar un Pipeline?

Class Dismissed