2020-01-31_titanic

August 6, 2022

1 Machine learning avec des catégories et du texte

Le jeu Titanic contient la liste des passages du Titanic, quelques informations comme le billet, la classe, et le fait qu'ils aient survécu. Il est possible d'utiliser le machine learning pour étudier la probabilité de survie ou plutôt de comprendre un peu mieux qui a eu la chance de survivre. S'il est possible de prédire, alors il existe une sorte de règle.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[2]:
```
%matplotlib inline
```

1.1 Les données

On peut les récupérer manuellement ou utiliser la fonction *load_titanic_dataset*. On pourra prendre l'un des deux jeux suivants.

[3]:
```
from papierstat.datasets import load_titanic_dataset
data1 = load_titanic_dataset(subset="A")
data1.head(n=2)
```

[3]:
```
   pclass  survived                            name     sex    age  sibsp  \
0       1         1   Allen, Miss. Elisabeth Walton  female  29.00      0
1       1         1  Allison, Master. Hudson Trevor    male   0.92      1

   parch  ticket      fare    cabin embarked boat  body  \
0      0   24160  211.3375       B5        S    2   NaN
1      2  113781  151.5500  C22 C26        S   11   NaN

                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
```

[4]:
```
data2 = load_titanic_dataset(subset="B")
data2.head(n=2)
```

[4]:
```
   row.names pclass  survived                             name   age  \
0          1    1st         1  Allen, Miss Elisabeth Walton  29.0
1          2    1st         0   Allison, Miss Helen Loraine   2.0

      embarked                      home.dest room      ticket boat     sex
0  Southampton                   St Louis, MO  B-5  24160 L221    2  female
```

```
1  Southampton  Montreal, PQ / Chesterville, ON  C26         NaN  NaN  female
```

1.2 Premier modèle de prédiction

On veut savoir si la survie de chaque personne était plutôt aléatoire où certaines personnes ont été privilégiées. Plutôt que de se lancer dans une étude de statistique descriptive, on cale un modèle de prédiction. Si celui-ci fonctionne, cela signifie qu'il existe un lien entre la survie et certaines des informations connues sur chaque passager. On considère les variables `age, sex, pclass`.

[5]:
```
df = data1
```

[6]:
```
from sklearn.model_selection import train_test_split

X = df[["age", "sex", "pclass"]]
y = df['survived']
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[7]:
```
set(df.sex), set(df.pclass)
```

[7]:
```
({'female', 'male'}, {1, 2, 3})
```

Il faut d'abord transformer les variables catégorielles en variables numériques. C'est le rôle d'un OneHotEncoder : chaque catégorie devient une variable binaire.

[8]:
```
from sklearn.preprocessing import OneHotEncoder
one = OneHotEncoder()
one.fit(X_train[['sex', 'pclass']])
```

[8]:
```
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,
              handle_unknown='error', sparse=True)
```

[9]:
```
one.transform(X_train[['sex', 'pclass']])
```

[9]:
```
<981x5 sparse matrix of type '<class 'numpy.float64'>'
        with 1962 stored elements in Compressed Sparse Row format>
```

Le résultat est une matrice sparse ou creuse qui ne contient que les valeurs non nulles. Pour voir la matrice, il faut la rendre dense.

[10]:
```
one.transform(X_train[['sex', 'pclass']]).todense()
```

[10]:
```
matrix([[1., 0., 0., 1., 0.],
        [0., 1., 0., 0., 1.],
        [0., 1., 1., 0., 0.],
        …,
        [1., 0., 0., 0., 1.],
        [0., 1., 0., 0., 1.],
        [0., 1., 0., 0., 1.]])
```

Cinq colonnes correspondant aux cinq catégories dont les noms sont les suivants :

[11]:
```
names = one.get_feature_names()
names
```

[11]:
```
array(['x0_female', 'x0_male', 'x1_1', 'x1_2', 'x1_3'], dtype=object)
```

[12]:
```
import numpy
cats = one.transform(X_train[['sex', 'pclass']]).todense()
age = X_train[['age']]
feat = numpy.hstack([age, cats])
feat[:10]
```

[12]:
```
matrix([[31.,  1.,  0.,  0.,  1.,  0.],
        [28.,  0.,  1.,  0.,  0.,  1.],
        [21.,  0.,  1.,  1.,  0.,  0.],
        [28.,  0.,  1.,  0.,  0.,  1.],
        [nan,  0.,  1.,  0.,  1.,  0.],
        [80.,  0.,  1.,  1.,  0.,  0.],
        [39.,  0.,  1.,  1.,  0.,  0.],
        [12.,  1.,  0.,  0.,  1.,  0.],
        [nan,  0.,  1.,  1.,  0.,  0.],
        [31.,  1.,  0.,  0.,  1.,  0.]])
```

La colonne age contient des valeurs manquantes. On les remplace par la moyenne.

[13]:
```
df[['age']].shape, df[['age']].dropna().shape
```

[13]:
```
((1309, 1), (1046, 1))
```

[14]:
```
from sklearn.impute import SimpleImputer
imp = SimpleImputer()
new_age = imp.fit_transform(X_train[['age']])
feat = numpy.hstack([new_age, cats])
feat[:10]
```

[14]:
```
matrix([[31.        ,  1.        ,  0.        ,  0.        ,  1.        ,
          0.        ],
        [28.        ,  0.        ,  1.        ,  0.        ,  0.        ,
          1.        ],
        [21.        ,  0.        ,  1.        ,  1.        ,  0.        ,
          0.        ],
        [28.        ,  0.        ,  1.        ,  0.        ,  0.        ,
          1.        ],
        [29.60563707,  0.        ,  1.        ,  0.        ,  1.        ,
          0.        ],
        [80.        ,  0.        ,  1.        ,  1.        ,  0.        ,
          0.        ],
        [39.        ,  0.        ,  1.        ,  1.        ,  0.        ,
          0.        ],
        [12.        ,  1.        ,  0.        ,  0.        ,  1.        ,
          0.        ],
        [29.60563707,  0.        ,  1.        ,  1.        ,  0.        ,
          0.        ],
        [31.        ,  1.        ,  0.        ,  0.        ,  1.        ,
          0.        ]])
```

On peut maintenant caler une forêt aléatoire.

[15]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(feat, y_train)
```

[15]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[16]:
```
cats_test = one.transform(X_test[['sex', 'pclass']])
age_test = imp.transform(X_test[['age']])
feat_test = numpy.hstack([age_test, cats_test.todense()])
```

On regarde les prédictions sur la base de test et ça a l'air de marcher.

[17]:
```
from sklearn.metrics import confusion_matrix

pred = rf.predict(feat_test)
confusion_matrix(y_test, pred)
```

[17]:
```
array([[177,  25],
       [ 49,  77]], dtype=int64)
```

La survie n'est pas donc aléatoire. On met tout cela dans un pipeline car c'est toujours plus simple à lire.

[18]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

pipe = Pipeline([
    ('cats', ColumnTransformer([
        ('one', OneHotEncoder(), ['sex', 'pclass']),
        ('imp', SimpleImputer(), ['age'])
    ]))
])
pipe.fit(X_train)
pipe.transform(X_test)
```

[18]:
```
array([[ 0.        ,  1.        ,  0.        ,  1.        ,  0.        ,
         30.        ],
       [ 0.        ,  1.        ,  0.        ,  0.        ,  1.        ,
         29.60563707],
       [ 0.        ,  1.        ,  0.        ,  0.        ,  1.        ,
         29.60563707],
       …,
       [ 0.        ,  1.        ,  0.        ,  0.        ,  1.        ,
         25.        ],
       [ 1.        ,  0.        ,  1.        ,  0.        ,  0.        ,
         53.        ],
       [ 1.        ,  0.        ,  1.        ,  0.        ,  0.        ,
         17.        ]])
```

On modifie le pipeline pour ajouter la forêt aléatoire.

[19]:
```
pipe = Pipeline([
    ('cats', ColumnTransformer([
        ('one', OneHotEncoder(), ['sex', 'pclass']),
```

```
        ('imp', SimpleImputer(), ['age'])
    ])),
    ('rf', RandomForestClassifier(n_estimators=100))
])

pipe.fit(X_train, y_train)
confusion_matrix(y_test, pipe.predict(X_test))
```

[19]:
```
array([[174,  28],
       [ 47,  79]], dtype=int64)
```

C'est plus clair. Quelques explications sur les variables, mais il faut vérifier en les retirant une à une pour vérifier leur importance dans l'histoire.

[20]:
```
cols = ['age'] + list(names)
list(zip(cols, pipe.steps[-1][1].feature_importances_))
```

[20]:
```
[('age', 0.2279931985187704),
 ('x0_female', 0.21516489235386238),
 ('x0_male', 0.04301082695552079),
 ('x1_1', 0.019360189479876846),
 ('x1_2', 0.06817992681764856),
 ('x1_3', 0.426290965874321)]
```

1.3 Utiliser d'autres variables

La variable `ticket` contient d'autres informations mais comme chaque ticket est unique, il est difficile de l'utiliser telle quelle. Il faut chercher des redondances d'une ligne à l'autre autrement c'est inexploitable. Et la redondance vient des mots que cette colonne contient. Il faut découper en mots :

- On fait l'inventaire de tous les mots uniques : ce sont des catégories.
- On crée une variable par mot : 1 si l'expression contient le mot, 0 sinon.

C'est une approche dite *bag of words* ou *sac de mots*.

[21]:
```
[_ for _ in set(df.ticket) if ' ' in _][:10]
```

[21]:
```
['SOTON/O.Q. 3101310',
 'A/4 48873',
 'STON/O2. 3101283',
 'PC 17611',
 'CA 31352',
 'C 17368',
 'A/5 3594',
 'W./C. 6608',
 'C.A. 34050',
 'C.A. 24580']
```

[22]:
```
X = df[["age", "sex", "pclass", "ticket"]]
y = df['survived']
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[23]:
```
from sklearn.feature_extraction.text import CountVectorizer

pipe = Pipeline([
```

```
    ('cats', ColumnTransformer([
        ('one', OneHotEncoder(), ['sex', 'pclass']),
        ('imp', SimpleImputer(), ['age']),
        ('bow', CountVectorizer(), ['ticket']),
    ])),
])

try:
    pipe.fit(X_train)
except ValueError as e:
    print(e)
```

```
all the input array dimensions for the concatenation axis must match exactly,
but along dimension 0, the array at index 0 has size 981 and the array at index
2 has size 1
```

Le modèle CountVectorizer est un peu problèmatique car il ne traite qu'une seule colonne. Il faut ruser pour l'utiliser dans un pipeline.

[24]:
```
CountVectorizer().fit_transform(X_train['ticket'])
```

[24]:
```
<981x756 sparse matrix of type '<class 'numpy.int64'>'
        with 1170 stored elements in Compressed Sparse Row format>
```

On essaye un petit tour de magie avec la classe *TextVectorizerTransformer*.

[25]:
```
from papierstat.mltricks import TextVectorizerTransformer

pipe = Pipeline([
    ('cats', ColumnTransformer([
        ('one', OneHotEncoder(), ['sex', 'pclass']),
        ('imp', SimpleImputer(), ['age']),
        ('bow', TextVectorizerTransformer(CountVectorizer()), ['ticket']),
    ])),
])

pipe.fit(X_train)
```

```
C:\xavierdupre\__home_\github_fork\scikit-
learn\sklearn\utils\deprecation.py:144: FutureWarning: The
sklearn.cluster.k_means_ module is  deprecated in version 0.22 and will be
removed in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.cluster. Anything that cannot be imported from
sklearn.cluster is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[25]:
```
Pipeline(memory=None,
         steps=[('cats',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('one',
 OneHotEncoder(categories='auto',
                                                 drop=None,
```

```
                                                                    dtype=<class
'numpy.float64'>,
handle_unknown='error',
                                                                    sparse=True),
                                                  ['sex', 'pclass']),
                                                 ('imp',
SimpleImputer(add_indicator=False,
                                                                    copy=True,
                                                                    fill_value=None,
                                                                    missing_…
TextVectorizerTransformer(estimator=CountVectorizer(analyzer='word',
                        binary=False,
                        decode_error='strict',
                        dtype=<class 'numpy.int64'>,
                        encoding='utf-8',
                        input='content',
                        lowercase=True,
                        max_df=1.0,
                        max_features=None,
                        min_df=1,
                        ngram_range=(1,
                                     1),
                        preprocessor=None,
                        stop_words=None,
                        strip_accents=None,
                        token_pattern='(?u)\\b\\w\\w+\\b',
                        tokenizer=None,
                        vocabulary=None)),
                                                  ['ticket'])],
                                    verbose=False))],
          verbose=False)
```

[26]:
```
pipe.transform(X_test)
```

[26]:
```
<328x762 sparse matrix of type '<class 'numpy.float64'>'
        with 1174 stored elements in Compressed Sparse Row format>
```

On peut ajouter un classifier au pipeline.

[27]:
```
pipe = Pipeline([
    ('cats', ColumnTransformer([
        ('one', OneHotEncoder(), ['sex', 'pclass']),
        ('imp', SimpleImputer(), ['age']),
        ('bow', TextVectorizerTransformer(CountVectorizer()), ['ticket']),
    ])),
    ('rf', RandomForestClassifier(n_estimators=100))
])

pipe.fit(X_train, y_train)
confusion_matrix(y_test, pipe.predict(X_test))
```

[27]:
```
array([[191,  14],
       [ 35,  88]], dtype=int64)
```

C'est un meilleur modèle.

[28]: