Adopt a Pet

You are in charge of an animal shelter and you want to predict if the animals you have in your possession can be adopted within 30 days or not.

The dataset at your disposal contains different information about the animals in the shelter: data about the breed or color, data about a cost, data about its health. You even have a short description written by the former owner and a picture of the animal.

We provide you only with the train part and a small test subset so that you can test the whole process.

Deadline: Jannuary 15, 2022.

You must submit a zip archive to LMS that contains 3 documents: - A pdf report that outlines the various stages of your work. You will insist on the different hyperparameters of your treatment and for each of them, you will specify on which ranges of values you have tested them. This report will also contain the precision obtained on the train set and on the test set. - the executable notebook containing only the chosen hyper-parameters and not their research. You will leave in this one the execution traces. - A ".joblib" file so that we can execute your code. Of course, the test dataset will be modified and only the predict function of the pipeline will be executed.

The final grade will be based on the quality of the prediction (accuracy score) for 25% and the quality of the work for 75%.

Table of Contents

In [ ]:

```
import os
from tqdm import tqdm
```

```
import warnings
warnings.filterwarnings("ignore")

import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

In [ ]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

Load train data

In [ ]:

```
path = "https://www.i3s.unice.fr/~riveill/dataset/petfinder-adoption-prediction/"
```

In [ ]:

```
breeds = pd.read_csv(path+'breed_labels.csv')
colors = pd.read_csv(path+'color_labels.csv')
states = pd.read_csv(path+'state_labels.csv')

train = pd.read_csv(path+'train.csv')

train['dataset_type'] = 'train'
```

In [ ]:

```
len(train)
```

Out[ ]:

```
8168
```

In [ ]:

```
# remove to train on the whole set
# N = 4
# train = train[:N]
```

In [ ]:

```
if 'dataset_type' in train.columns:
    train = train.drop(labels='dataset_type', axis=1)
train.columns
```

Out[ ]:

```
Index(['Type', 'Age', 'Gender', 'Color1', 'Color2', 'Color3', 'MaturitySize',
       'FurLength', 'Vaccinated', 'Dewormed', 'Sterilized', 'Health', 'Fee',
       'Description', 'Images', 'Breed', 'target'],
      dtype='object')
```

In [ ]:

```
y_train = train['target']
X_train = train.drop(['target'], axis=1)
X_train.head()
```

Out[ ]:

	Type	Age	Gender	Color1	Color2	Color3	MaturitySize	FurLength	Vaccinated	Dewormed	Sterilized	Health	Fee	Description	Image
0	Cat	12.0	Female	White	Unknown	Unknown	Medium	Yes	Unknown	Unknown	Unknown	Healthy	0.0	We got Luna when she was a kitten in Feb 15'. ...	880e1378 4.jp
1	Cat	4.0	Male	Golden	White	Unknown	Medium	Yes	No	Yes	No	Healthy	0.0	Ginger Boy was found starving and hungry so I ...	7abe9a0a 2.jp
2	Cat	12.0	Female	Black	Golden	White	Medium	No	No	No	No	Healthy	0.0	An indoor cat with nice green/ yellowish eyes....	605d07d3 5.jp
3	Dog	60.0	Male	Black	Gray	White	Medium	No	Yes	Unknown	Unknown	Healthy	0.0	My dog name called boo. He is a male. I feedin...	7ed568ab 1.jp
4	Cat	36.0	Female	Cream	Gray	White	Large	No	No	No	Yes	Healthy	0.0	1) Foxy is a stray cat which I feed regularly,...	8969b314 5.jp

In [ ]:

```
cat_cols = ['Type', 'Gender', 'Breed', 'Color1', 'Color2', 'Color3',
           'MaturitySize', 'FurLength', 'Vaccinated', 'Dewormed',
       'Sterilized', 'Health']
num_cols = ['Age', 'Fee']
txt_cols = ['Description']
img_cols = ['Images']
```

Load the images

In [ ]:

```
# Build the image list of the training set
img_dir = "train_images/"
X_train['Images'] = [path+img_dir+img for img in train['Images']]
```

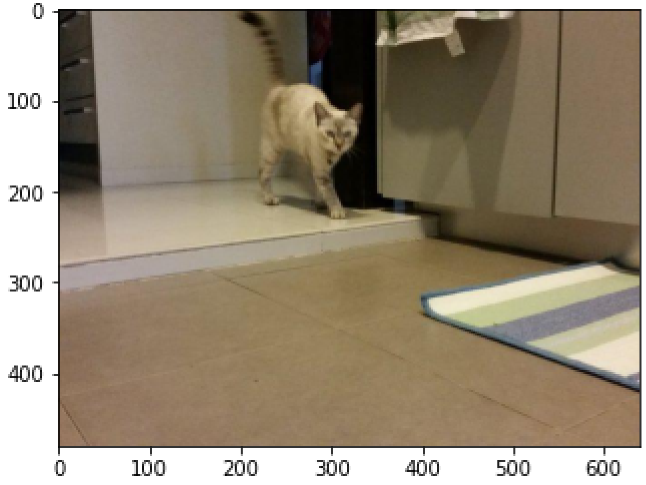
In [ ]:

```
from skimage import io

# Read the first image of the list
img = io.imread(X_train['Images'][0])
# have a look to the image
plt.imshow(img)
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f9fff65b6d0>
```



Compute SIFT detector and descriptors for one image

In [ ]:

```
# convert the image to grey levels
```

```
import cv2

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

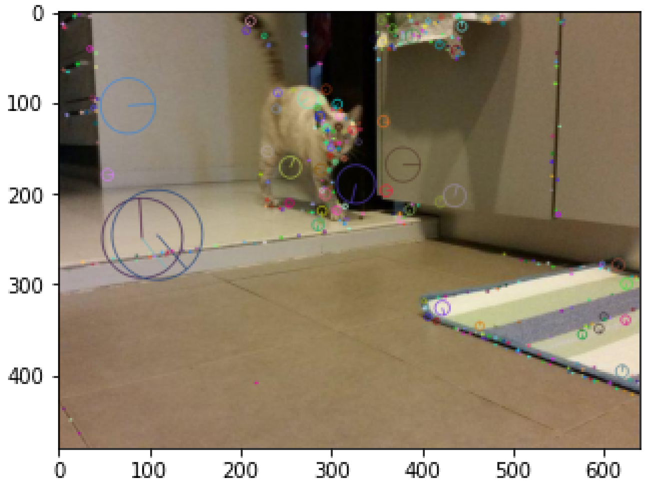
In [ ]:

```
# compute SIFT detector and descriptors
sift = cv2.SIFT_create()
kp,des = sift.detectAndCompute(gray,None)

# plot image and descriptors
cv2.drawKeypoints(img,kp,img,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(img)
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f9fec44f8e0>
```



Extract features and build BOFs

In [ ]:

```
# First step, extract the SIFTs of each image
# Be carefull: very long process

def extract_SIFT(img_lst):
    nbSIFTs = 0     # Nomber of SIFTs
    SIFTs = []  # List of SIFTs descriptors
    #dimImgs = []   # Nb of descriptors associated to each images

    for pathImg in tqdm(img_lst, position=0, leave=True):
        img = io.imread(pathImg)
        if len(img.shape)==2: # this is a grey level image
            gray = img
        else: # we expect the image to be a RGB image or RGBA
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        sift = cv2.SIFT_create()
        kp, des = sift.detectAndCompute(gray, None)
        if len(kp) == 0 and img.shape[2]==4: #some images are mask on alpha channel: we thus extract this channel if
            gray = img[:,:,3]
            sift = cv2.SIFT_create()
            kp, des = sift.detectAndCompute(gray, None)

        nbSIFTs += des.shape[0]
        SIFTs.append(des)
        #dimImgs.append(des.shape[0])
    return nbSIFTs, SIFTs#, dimImgs
```

In [ ]:

```
# nbSIFTs, SIFTs = extract_SIFT(X_train['Images'])
# print('nbSifts: ', nbSIFTs)
```

In [ ]:

```
# Step 2: clusterize the SIFT
from sklearn.cluster import MiniBatchKMeans

def clusterize(SIFTs, nb_img_features=5, verbose=False):
    clusterizer = MiniBatchKMeans(n_clusters=nb_img_features)   # nb_img_features is a hyperparameter
    # learning of the clustering
    flat_list = SIFTs[0]
    for des in SIFTs[1:]:
        flat_list = np.concatenate((flat_list, des))
        if verbose:
            print("shape:", des.shape, flat_list.shape)
    clusterizer.fit(flat_list)
    # we now know the label of each SIFT descriptor
    return clusterizer
```

In [ ]:

```
# clusterizer = MiniBatchKMeans(n_clusters=5)   # nb_img_features is a hyperparameter
# # learning of the clustering
# flat_list = SIFTs[0]
# for des in SIFTs[1:]:
```

```
#     flat_list = np.concatenate((flat_list, des))
# clusterizer.fit(flat_list)
```

In [ ]:

```
# clusterizer = clusterize(SIFTs, verbose=True)
```

In [ ]:

```
# Step 3: build the BOW representation of each images (i.e. construction of the BOFs)

def build_BOFs(SIFTs, clusterizer, verbose=False):
    ok, nok = 0, 0
    #BOF initialization
    nb_img_features = clusterizer.get_params()['n_clusters']
    BOFs = np.empty(shape=(0, nb_img_features), dtype=int)

    # Build label list
    flat_list = SIFTs[0]
    for des in SIFTs[1:]:
        flat_list = np.concatenate((flat_list, des))
        if verbose:
            print("shape:", des.shape, flat_list.shape)
    labels = clusterizer.predict(flat_list)

    # loop on images
    i = 0 # index for the loop on SIFTs
    for des in SIFTs:
        #initialisation of the bof for the current image
        tmpBof = np.array([0]*nb_img_features)
        j = 0
        # for every SIFT of the current image:
        nbs = des.shape[0]
        while j < nbs:
            tmpBof[labels[i]] += 1
            j+=1
            i+=1
        BOFs = np.concatenate((BOFs, tmpBof.reshape(1,-1)), axis=0)
    if verbose:
        print("BOFs : ", BOFs)

    return BOFs
```

In [ ]:

```
# BOFs = build_BOFs(SIFTs, clusterizer, verbose=True)
# BOFs.shape
```

In [ ]:

```
from sklearn.base import BaseEstimator,TransformerMixin

def list_comparaison(l1, l2):
    if not l1 is None \
        and not l2 is None \
        and len(l1)==len(l2) \
        and len(l1)==sum([1 for i,j in zip(l1, l2) if i==j]):
        return True
    return False

class BOF_extractor(BaseEstimator,TransformerMixin):
    X = None
    SIFTs = None
    nbSIFTs = 0

    def __init__(self, nb_img_features=10, verbose=False):
        self.nb_img_features = nb_img_features
        self.verbose = verbose
        self.path = path
        if self.verbose:
            print("BOF.init()")

    def fit(self, X, y=None):
        if self.verbose:
            print("BOF.fit()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)
        self.X = X
        self.SIFTs = SIFTs
        self.nbSIFTs = nbSIFTs
        self.clusterizer = clusterize(SIFTs, self.nb_img_features, self.verbose)

    def transform(self, X, y=None):
        if self.verbose:
            print("BOF.transform()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)

        if self.verbose:
            print("nbSIFTs:", nbSIFTs)
        return build_BOFs(SIFTs, self.clusterizer, self.verbose)

    def fit_transform(self, X, y=None):
        if self.verbose:
            print("BOF.fit_transform()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)
        self.X = X
        self.SIFTs = SIFTs
        self.nbSIFTs = nbSIFTs
        self.clusterizer = clusterize(SIFTs, self.nb_img_features, self.verbose)
        return build_BOFs(SIFTs, self.clusterizer, self.verbose)
```

In [ ]:

```
test_BOF_extractor = BOF_extractor(nb_img_features=5, verbose=True)
```

```
BOF.init()
```

In [ ]:

```
# test_BOF_extractor.fit(X_train['Images'])
```

In [ ]:

```
# BOFs = test_BOF_extractor.transform(X_train['Images'])
# BOFs.shape
```

In [ ]:

```
# BOFs = test_BOF_extractor.fit_transform(X_train['Images'])
# BOFs.shape
```

In [ ]:

```
test = pd.read_csv(path+"test.csv")
y_test = test['target']
X_test = test.drop(['target'], axis=1)

img_dir = "test_images/"
X_test['Images'] = [path+img_dir+img for img in test['Images']]
print(len(X_test))
X_test.head()
```

```
250
```

Out[ ]:

	Type	Age	Gender	Color1	Color2	Color3	MaturitySize	FurLength	Vaccinated	Dewormed	Sterilized	Health	Fee	Description	
0	Dog	1.0	Male	Black	Gray	Unknown	Medium	Yes	No	No	No	Healthy	0.0	The second puppy of Megan's first litter. Puma...	https://www
1	Cat	3.0	Female	Black	Yellow	Unknown	Medium	Yes	No	No	No	Healthy	0.0	Tim is female kitten.active and playful.pls sm...	https://www
2	Dog	5.0	Female	Black	Brown	Unknown	Medium	Yes	Yes	Yes	Yes	Healthy	0.0	She was found wearing a red collar, wandering ...	https://www
3	Cat	3.0	Male	Cream	Unknown	Unknown	Small	Yes	No	Yes	No	Healthy	0.0	3 months old male kitten. Adopters have to vac...	https://www
4	Dog	0.0	Female	Black	Unknown	Unknown	Small	No	Unknown	Unknown	Unknown	Minor Injury	0.0	Please help her. She is an abandon victim. Ver...	https://www

In [ ]:

```
# BOFs = test_BOF_extractor.transform(X_test['Images'])
# BOFs.shape
```

Build a basic model

There are much more interesting things in the dataset and I'm going to explore them, but for now let's build a simple model as a baseline.

In [ ]:

```
import os
from sklearn import set_config

from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler, FunctionTransformer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline, FeatureUnion, Pipeline
from sklearn.decomposition import PCA, SparsePCA, TruncatedSVD

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score


set_config(display="text")


def nb_colors(df):
    """
    Compute the number of known (i.e. not =="Unkown") colors
    """
    return pd.DataFrame((df != "Unknown").sum(axis=1))
```

In [ ]:

```
categorical_preprocessor = OneHotEncoder(handle_unknown="ignore", sparse=False)
numerical_preprocessor = StandardScaler()
text_preprocessor = TfidfVectorizer()
image_preprocessor = BOF_extractor(nb_img_features=3, verbose=False)

nb_colors_transformer = FunctionTransformer(func=nb_colors, validate=False)

preprocessor = ColumnTransformer(transformers=[
    ("categorical encoding", categorical_preprocessor, cat_cols),
    ("numerical encoding", numerical_preprocessor, num_cols),
    ("text encoding", text_preprocessor, 'Description'),
    ("image encoding", image_preprocessor, 'Images'),
    ("compute nb colors", nb_colors_transformer,
     [f"Color{i}" for i in range(1, 4)]),
])

dim_red = FeatureUnion([("Truncated SVD", TruncatedSVD(n_components=100))])

preproc_pipe = Pipeline(steps=[
    ("preprocessing", preprocessor),
    ("dimensionality reduction", dim_red),
])
# print("\n--- start preproc_X_train ---")
# preproc_X_train = preproc_pipe.fit_transform(X_train)
# print(f"preproc_X_train shape: {preproc_X_train.shape}")

# print("\n--- start preproc_X_test ---")
# preproc_X_test = preproc_pipe.transform(X_test)
# print(f"preproc_X_test shape: {preproc_X_test.shape}")
# print("\n--- done ---")
```

Find decent models

In [ ]:

```
set_config(display="diagram")
set_config(display="text")

model = Pipeline(steps=[
   ("preprocessing", preprocessor),
    ("dimensionality reduction", dim_red),
    ("classifying", RandomForestClassifier(
        max_depth=200, min_samples_leaf=3, min_samples_split=4)),
])
model.fit(X_train, y_train)
model
```

```
100%|██████████| 8168/8168 [24:33<00:00,  5.54it/s]
```

Out[ ]:

```
Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('categorical encoding',
                                                  OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
                                                  ['Type', 'Gender', 'Breed',
                                                   'Color1', 'Color2', 'Color3',
                                                   'MaturitySize', 'FurLength',
                                                   'Vaccinated', 'Dewormed',
                                                   'Sterilized', 'Health']),
                                                 ('numerical encoding',
                                                  StandardScaler(),
                                                  ['Age', 'Fee']),
                                                 ('text encoding',
                                                  TfidfVectorize...
                                                  BOF_extractor(nb_img_features=3),
                                                  'Images'),
                                                 ('compute nb colors',
                                                  FunctionTransformer(func=<function nb_colors at 0x7fd9da5231f0>),
                                                  ['Color1', 'Color2',
                                                   'Color3'])])),
                ('dimensionality reduction',
                 FeatureUnion(transformer_list=[('Truncated SVD',
                                                 TruncatedSVD(n_components=100))])),
                ('classifying',
                 RandomForestClassifier(max_depth=200, min_samples_leaf=3,
                                        min_samples_split=4))])
```

In [ ]:

```python
# Save the model
from joblib import dump, load

dump(model, '/media/joris/Data/limonier.joblib') # Put your name as a model name
```

```
The history saving thread hit an unexpected error (OperationalError('database or disk is full')).History will not be
written to the database.
```

Out[ ]:

```
['/media/joris/Data/limonier.joblib']
```

Evaluation of the model

We will only execute the following cells.

In [ ]:

```
test = pd.read_csv(path+"test.csv")

y_test = test['target']
X_test = test.drop(['target'], axis=1)

img_dir = "test_images/"
X_test['Images'] = [path+img_dir+img for img in test['Images']]
print("Test size:", len(X_test))

model = load('/media/joris/Data/submission-complete-limonier/limonier.joblib')

y_pred = model.predict(X_train)
print("ACC on train", accuracy_score(y_train, y_pred))

y_pred = model.predict(X_test)
print("ACC on test", accuracy_score(y_test, y_pred))
```

```
Test size: 250
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [32], in <module>
      8 print("Test size:", len(X_test))
     10 model = load('/media/joris/Data/submission-complete-limonier/limonier.joblib')
---> 12 y_pred = model.predict(X_train)
     13 print("ACC on train", accuracy_score(y_train, y_pred))
     15 y_pred = model.predict(X_test)

File ~/.local/lib/python3.9/site-packages/sklearn/utils/metaestimators.py:113, in _AvailableIfDescriptor.__get__.<loc
als>.<lambda>(*args, **kwargs)
    110         raise attr_err
    112     # lambda, but not partial, allows help() to work with update_wrapper
--> 113     out = lambda *args, **kwargs: self.fn(obj, *args, **kwargs)  # noqa
    114 else:
    116     def fn(*args, **kwargs):

File ~/.local/lib/python3.9/site-packages/sklearn/pipeline.py:469, in Pipeline.predict(self, X, **predict_params)
    467 Xt = X
    468 for _, name, transform in self. iter(with_final=False):
--> 469     Xt = transform.transform(Xt)
    470 return self.steps[-1][1].predict(Xt, **predict_params)

File ~/.local/lib/python3.9/site-packages/sklearn/compose/_column_transformer.py:748, in ColumnTransformer.transform
(self, X)
    743 else:
    744     # ndarray was used for fitting or transforming, thus we only
    745     # check that n_features_in_ is consistent
    746     self. check n features(X, reset=False)
--> 748 Xs = self._fit_transform(
    749     X,
    750     None,
    751      transform one,
    752     fitted=True,
    753     column_as_strings=fit_dataframe_and_transform_dataframe,
    754 )
    755 self._validate_output(Xs)
    757 if not Xs:
    758     # All transformers are None

File ~/.local/lib/python3.9/site-packages/sklearn/compose/_column_transformer.py:606, in ColumnTransformer._fit_trans
form(self, X, y, func, fitted, column_as_strings)
    600 transformers = list(
    601     self._iter(
    602         fitted=fitted, replace_strings=True, column_as_strings=column_as_strings
    603     )
    604 )
    605 try:
--> 606     return Parallel(n_jobs=self.n_jobs)(
    607         delayed(func)(
    608             transformer=clone(trans) if not fitted else trans,
    609             X= safe_indexing(X, column, axis=1),
    610             y=y,
    611             weight=weight,
    612             message_clsname="ColumnTransformer",
    613             message=self._log_message(name, idx, len(transformers)),
    614         )
    615         for idx, (name, trans, column, weight) in enumerate(transformers, 1)
    616     )
    617 except ValueError as e:
    618     if "Expected 2D array, got 1D array instead" in str(e):

File ~/.local/lib/python3.9/site-packages/joblib/parallel.py:1046, in Parallel.__call__(self, iterable)
   1043 if self.dispatch_one_batch(iterator):
   1044     self._iterating = self._original_iterator is not None
-> 1046 while self.dispatch_one_batch(iterator):
   1047     pass
   1049 if pre_dispatch == "all" or n_jobs == 1:
   1050     # The iterable was consumed all at once by the above for loop.
   1051     # No need to wait for async callbacks to trigger to
   1052     # consumption.

File ~/.local/lib/python3.9/site-packages/joblib/parallel.py:861, in Parallel.dispatch_one_batch(self, iterator)
    859     return False
    860 else:
--> 861     self._dispatch(tasks)
    862     return True

File ~/.local/lib/python3.9/site-packages/joblib/parallel.py:779, in Parallel._dispatch(self, batch)
    777 with self._lock:
    778     job_idx = len(self._jobs)
--> 779     job = self._backend.apply_async(batch, callback=cb)
    780     # A job can complete so quickly than its callback is
    781     # called before we get here, causing self._jobs to
    782     # grow. To ensure correct results ordering, .insert is
    783     # used (rather than .append) in the following line
    784     self._jobs.insert(job_idx, job)
```

```
File ~/.local/lib/python3.9/site-packages/joblib/_parallel_backends.py:208, in SequentialBackend.apply_async(self, fu
nc, callback)
    206 def apply_async(self, func, callback=None):
    207     """Schedule a func to be run"""
--> 208     result = ImmediateResult(func)
    209     if callback:
    210         callback(result)

File ~/.local/lib/python3.9/site-packages/joblib/_parallel_backends.py:572, in ImmediateResult.__init__(self, batch)
    569 def __init__(self, batch):
    570     # Don't delay the application, to avoid keeping the input
    571     # arguments in memory
--> 572     self.results = batch()

File ~/.local/lib/python3.9/site-packages/joblib/parallel.py:262, in BatchedCalls.__call__(self)
    258 def __call__(self):
    259     # Set the default nested backend to self._backend but do not set the
    260     # change the default number of processes to -1
    261     with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262         return [func(*args, **kwargs)
    263                 for func, args, kwargs in self.items]

File ~/.local/lib/python3.9/site-packages/joblib/parallel.py:262, in <listcomp>(.0)
    258 def __call__(self):
    259     # Set the default nested backend to self._backend but do not set the
    260     # change the default number of processes to -1
    261     with parallel backend(self. backend, n_jobs=self._n_jobs):
--> 262         return [func(*args, **kwargs)
    263                 for func, args, kwargs in self.items]

File ~/.local/lib/python3.9/site-packages/sklearn/utils/fixes.py:216, in _FuncWrapper.__call__(self, *args, **kwargs)
    214 def __call__(self, *args, **kwargs):
    215     with config_context(**self.config):
--> 216         return self.function(*args, **kwargs)

File ~/.local/lib/python3.9/site-packages/sklearn/pipeline.py:876, in _transform_one(transformer, X, y, weight, **fit
_params)
    875 def _transform one(transformer, X, y, weight, **fit_params):
--> 876     res = transformer.transform(X)
    877     # if we have a weight for this transformer, multiply output
    878     if weight is None:

Input In [21], in BOF_extractor.transform(self, X, y)
```

```
     49 if self.verbose:
     50     print("nbSIFTs:", nbSIFTs)
---> 51 return build_BOFs(SIFTs, self.clusterizer, self.verbose)

Input In [19], in build_BOFs(SIFTs, clusterizer, verbose)
     10 flat_list = SIFTs[0]
     11 for des in SIFTs[1:]:
---> 12     flat_list = np.concatenate((flat_list, des))
     13     if verbose:
     14         print("shape:", des.shape, flat_list.shape)

File <__array_function__ internals>:180, in concatenate(*args, **kwargs)

KeyboardInterrupt:
```

In [ ]: