Modeling exercise

General Instructions

- Submission date: 25.4.2022
- Submission Method: Link to your solution notebook in this sheet.

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import sys; sys.path.append('../Modles and Modeling/src')
import numpy as np
import plotly_express as px
```

In [3]:
```
import pandas as pd
import ipywidgets as widgets
```

In [6]:
```
from datasets import make_circles_dataframe, make_moons_dataframe
```

Fitting and Overfiting

The goal of the following exercise is to:

- Observe overfitting due to insuffient data
- Observe Overfitting due to overly complex model
- Identify the overfitting point by looking at Train vs Test error dynamic
- Observe how noise levels effect the needed data samples and model capacity

To do so, you'll code an experiment in the first part, and analyze the experiment result in the second part.

Building an experiment

Code:

1. Create data of size N with noise level of magnitude NL from datasets DS_NAME.
2. Split it to training and validation data (no need for test set), use 80%-20%.
3. Use Logistic regression and Choose one complex model of your choice: KNN, SVM with RBF kernel with different `gamma` values or Random forest classifier with differnt number of `min_samples_split`.
4. Train on the train set for different hyper parameter values. compute:
   A. Classification accuracy on the validation set (TRE)
   B. Classification accuracy on the validation set (TESTE)

   C. The difference beteen the two above (E_DIFF)
5. Save DS_NAME, N, NL, CLF_NAME, K, TRE, TESTE, E_DIFF and the regularization/hyper param (K, gamma or min_samples_split and regularization value for the linear regression classifier)

Repeat for:

- DS_NAME in Moons, Circles
- N (number of samples) in [5, 10, 50, 100, 1000, 10000]
- NL (noise level) in [0, 0.1, 0.2, 0.3, 0.4, 0.5]
- For the complex model: 10 Values of hyper parameter of the complex model you've chosen.
- For the linear model: 5 values of ridge (l2) regularization - [0.001, 0.01, 0.1, 1, 10, 100, 1000]

Analysing the expermient results

1. What is the best model and model params? How stable is it?
2. What is the most stable model and model params? How good is it?
3. Does regularization help for linear models?
4. For a given noise level of your chioce, How does the train, test and difference error changes with increasing data sizes?
5. For a given noise level of your chioce, How does the train, test and difference error changes with increasing model complexity?
6. Are the noise level effect the number of datapoints needed to reach minimal test results?
7. Select the best model param to fit the data (The one minimizing the test error) and for it, observe the number of data points needed for optimal test results in different noise levels.

Tips and Hints

For buliding the experiment:

- Start with one dataframe holding all the data for both datastes with different noise level. Use the `make_<dataset_name>_dataframe()` functions below, and add two columns, dataset_name and noise_level, before appending the new dataset to the rest of the datasets. Use `df = pd.DataFrame()` to start with an empty dataframe and using a loop, add data to it using `df = df.append(<the needed df here>)`. Verify that you have 10k samples for each dataset type and noise level by a proper `.value_counts()`. You can modify the
- When you'll need an N samples data with a specific noise level, use `query()` and `head(n)` to get the needed dataset.
- Use sklearn `train_test_split()` method to split the data with `test_size` and `random_state` parameters set correctly to ensure you are always splitting the data the same why for a given fold `k`. Read the docs if needed.

- You can also not create your own data splitter, and instead use `model_selection.cross_validate()` from sklearn. You'll need to ask for the train erros as well as the test errors, see here.
- Use prints in proper location to ensure the progress of the experiment.

If you get stuck, and need refernce, scroll to the end of the notebook to see more hints!

Moons dataset

In [1]:
```
from sklearn.datasets import make_moons
```

In [4]:
```
moons_df = make_moons_dataframe(n_samples=1000, noise_level=0.1)
moons_df.head()
```

Out[4]:

	x	y	label
0	-0.104463	0.163270	B
1	0.875719	0.397855	A
2	0.280194	-0.130234	B
3	0.728661	0.676592	A
4	0.193158	-0.163336	B

In [5]:
```
@widgets.interact
def plot_noisy_moons(noise_level = widgets.FloatSlider(value=0, min=0, max=0.5, step=0
    moons_df = make_moons_dataframe(n_samples=1000, noise_level=noise_level)
    return px.scatter(moons_df, x='x', y='y', color = 'label')
```

```
interactive(children=(FloatSlider(value=0.0, description='noise_level', max=0.5, step
=0.05), Output()), _dom_c…
```

Circles Dataset

In [44]:
```
circles_df = make_circles_dataframe(n_samples=500, noise_level=0)
circles_df.head()
```

Out[44]:

	x	y	label
0	0.762443	0.647056	A
1	0.162637	0.986686	A
2	0.954865	0.297042	A
3	0.285019	-0.958522	A
4	0.769622	0.218362	B

In [45]:
```
@widgets.interact
def plot_noisy_circles(noise_level = widgets.FloatSlider(value=0, min=0, max=0.5, step
    df = make_circles_dataframe(1000, noise_level)
    return px.scatter(df, x='x', y='y', color = 'label')
```

```
interactive(children=(FloatSlider(value=0.0, description='noise_level', max=0.5, step
=0.05), Output()), _dom_c…
```

Appendix

More hints!

If you'll build the datasets dataframe correctly, you'll have **one** dataframe that has dataset_name and noise_level colmuns, as well as the regular x,y,label colmns. To unsure you've appended everything correctly, groupby the proper colmuns and look at the size:

In [51]:
```
# Use proper groupby statement to ensure the datasets dataframe contains data as expec
```

Out[51]:
```
dataset_name  noise_level
circles       0.0            10000
              0.1            10000
              0.2            10000
              0.3            10000
              0.4            10000
              0.5            10000
moons         0.0            10000
              0.1            10000
              0.2            10000
              0.3            10000
              0.4            10000
              0.5            10000
dtype: int64
```

Your

You experiment code should look something like that:

In [ ]:
```
datasets_type = ['circles', 'moons']
k_folds = 10
n_samples = [10, 50, 100, 1000, 10000]
noise_levels = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
clf_types = ['log_reg', 'svm']
hp_range = <'Your hyper parameters ranges here'>
regularization_values = <'Your regularization values here'>
results = []
for ds_type in datasets_type:
    print(f'Working on {ds_type}')
    for nl in noise_levels:
        for n in n_samples:
            ds = datasets.query(<'your query here'>).head(n)
            print(f'Starting {k_folds}-fold cross validation for {ds_type} datasets wi
            for k in range(k_folds):
                X, Y = <'Your code here'>
                x_train,x_test,y_train,y_test= <'Your code here'>
                for clf_type in clf_types:
                    if clf_type == 'log_reg':
                        for regularization_value in regularization_values:
                            train_acc, test_acc = <'Your code here'>
                            results.append(<'Your code here'>)
                    if clf_type == 'svm':
```

```
                        for gamma in hp_range:
                            train_acc, test_acc = <'Your code here'>
                            results.append(<'Your code here'>)
```

```
Working on circles
Starting 10-fold cross validation for circles datasets with 10 samples and noise leve
l 0. Going to train ['log_reg', 'svm'] classifiers.
Starting 10-fold cross validation for circles datasets with 50 samples and noise leve
l 0. Going to train ['log_reg', 'svm'] classifiers.
Starting 10-fold cross validation for circles datasets with 100 samples and noise lev
el 0. Going to train ['log_reg', 'svm'] classifiers.
Starting 10-fold cross validation for circles datasets with 1000 samples and noise le
vel 0. Going to train ['log_reg', 'svm'] classifiers.
Starting 10-fold cross validation for circles datasets with 10000 samples and noise l
evel 0. Going to train ['log_reg', 'svm'] classifiers.
```

Question 1 - Manual Classification

The purpose of this excercise is to examplify the need in a fitting algorithm. We will do so by trying to find only 2 models parameters by ourselves.

In [2]:
```
slope, intercept = 2.5, 6
```

In [3]:
```
x_1, x_2 = 0.2, 0.6
on_line = [[x, x*slope + intercept,'on_line'] for x in np.linspace(-1,2,100)]

above_line = [[x_1, x_1*slope + intercept + 2, 'Above'],
              [x_2, x_2*slope + intercept + 2, 'Above']]

below_line = [[x_1, x_1*slope + intercept - 2, 'Below'],
              [x_2, x_2*slope + intercept - 2, 'Below']]
```

In [4]:
```
columns = ['x','y','label']
data = pd.DataFrame(on_line + above_line + below_line, columns = columns)
```

In [5]:
```
px.scatter(data, x='x', y='y', color = 'label')
```

In [ ]:

In [ ]:

Solution

Building an experiment

We first start by **importing** the necessary libraries.

In [2]:
```
%load_ext autoreload
%autoreload 2
```

In [3]:
```
import sys; sys.path.append('../Modles and Modeling/src')
import numpy as np
import plotly_express as px
import pandas as pd
import ipywidgets as widgets
from ipywidgets import interact
from datasets import make_circles_dataframe, make_moons_dataframe
from sklearn.datasets import make_moons
```

```
from sklearn.datasets import make_circles
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
```

Note: Not all the libraries were use for the final project. In the process, I tried different options in order to understand how each library works (for example train_test_split vs cross_validate).

Secondly, we will **create our database** using different sample sizes and noise levels. We use two different patters: moon and circles.

In [4]:
```
def make_moons_dataframe(n_samples, noise_level):
    points, label = make_moons(n_samples=n_samples, noise=noise_level)
    moons_df = pd.DataFrame(points, columns=['x','y'])
    moons_df['label'] = label
    moons_df.label = moons_df.label.map({0:'A', 1:'B'})
    return moons_df
```

In [5]:
```
def make_circles_dataframe(n_samples, noise_level):
    points, label = make_circles(n_samples=n_samples, noise=noise_level)
    circles_df = pd.DataFrame(points, columns=['x','y'])
    circles_df['label'] = label
    circles_df.label = circles_df.label.map({0:'A', 1:'B'})
    return circles_df
```

In [6]:
```
num_samples_list = [5, 10, 50, 100, 1000, 10000]
noise_levels_list = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
circles_df = pd.DataFrame([])
for n in num_samples_list:
    for noise in noise_levels_list:
        circles = make_circles_dataframe(n_samples=n, noise_level=noise)
        circles['dataset_name'] = 'circles'
        circles['n_samples'] = n
        circles['noise_levels'] = noise
        circles_df = circles_df.append(circles)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1200794857.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  circles_df = circles_df.append(circles)
```

In [7]:
```
num_samples_list = [5, 10, 50, 100, 1000, 10000]
noise_levels_list = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
moons_df = pd.DataFrame([])
for n in num_samples_list:
    for noise in noise_levels_list:
        moons = make_moons_dataframe(n_samples=n, noise_level=noise)
        moons['dataset_name'] = 'moons'
        moons['n_samples'] = n
        moons['noise_levels'] = noise
        moons_df = moons_df.append(moons)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\1335896416.py:10: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  moons_df = moons_df.append(moons)
```

In [8]:

```
datasets = pd.DataFrame([])
datasets = datasets.append(moons_df)
datasets = datasets.append(circles_df)
datasets.tail(10)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13400\444765414.py:2: FutureWarning: The f
rame.append method is deprecated and will be removed from pandas in a future version.
Use pandas.concat instead.
  datasets = datasets.append(moons_df)
C:\Users\User\AppData\Local\Temp\ipykernel_13400\444765414.py:3: FutureWarning: The f
rame.append method is deprecated and will be removed from pandas in a future version.
Use pandas.concat instead.
  datasets = datasets.append(circles_df)
```

Out[8]:

	x	y	label	dataset_name	n_samples	noise_levels
9990	-1.313797	0.976905	B	circles	10000	0.5
9991	0.673651	-0.221128	B	circles	10000	0.5
9992	0.660133	0.092870	B	circles	10000	0.5
9993	-1.052245	-0.273841	A	circles	10000	0.5
9994	0.115216	0.258236	B	circles	10000	0.5
9995	-1.133363	0.617948	B	circles	10000	0.5
9996	0.182054	-1.709036	B	circles	10000	0.5
9997	-1.110905	1.823632	A	circles	10000	0.5
9998	0.718266	0.309295	B	circles	10000	0.5
9999	-0.500002	1.423726	A	circles	10000	0.5

In [9]:

```
check = datasets.groupby(['dataset_name','noise_levels'])['dataset_name'].agg(['count'
```

```
check
```

Out[9]:

dataset_name	noise_levels	count
circles	0.0	11165
	0.1	11165
	0.2	11165
	0.3	11165
	0.4	11165
	0.5	11165
moons	0.0	11165
	0.1	11165
	0.2	11165
	0.3	11165
	0.4	11165
	0.5	11165

Here, we will try to understand the **pattern of our data**. We will do this using a scatter plot and some widgets in order to choose a particular database. Sometimes from the pattern of the data it is clear which machine learning model is more appropriate.

In [10]:
```
data_set = widgets.Dropdown(options=datasets['dataset_name'].unique(), description="Se
sample_size = widgets.Dropdown(options=datasets['n_samples'].unique(), description="Se
noise_level = widgets.Dropdown(options=datasets['noise_levels'].unique(), description=
```

In [11]:
```
@widgets.interact
def plot_dataset(data = data_set, sample = sample_size, noise = noise_level):
    data_sel = datasets.query('dataset_name == @data & noise_levels == @noise & n_samp
    fig = px.scatter(data_sel, x='x', y='y', color = 'label')
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('moons', 'circle
s'), value='moons'), Dropdo…
```

We choose two machine learning algorithms to train: **logist regression with l2 regularization and svm with a RBF kernel.** As we saw before we are dealing with a **non-linear database.** Therefore, a linear kernel is not appropriate here. For the purpose of the experiment we set the C regularization parameter as 1.

In the next piece of code we will split all the database betweem training data and validation data. Then we will train the both models through different regularization parameters.

Note: I tried running different regularization parameters ranges. The final sets are those that best fit the models.

Another important point to **remark**. A sample of 10000 observations took a very long time to run!!! I split the loop in two: the first one for the noise range [0, 0.1, 0.2] and the second one for the noise range [0.3, 0.4, 0.5]. Then I appended both temporal resuls into a final table called results. Finally I exported the results as a csv file for future uses instead of re-run all the code from the start.

This is the loop for the first range of noise level: [0, 0.1, 0.2]

In [21]:
```
datasets_type = ['circles', 'moons']
k = 10
n_samples = [10, 50, 100, 1000, 10000]
noise_levels = [0, 0.1, 0.2]
clf_types = ['log_reg', 'svm']
gamma_range = [0.0001, 0.001, 0.01, 1,  3, 7, 11]
regularization_values = [0.001, 0.01, 0.1, 1, 10, 100] # l2-Penalty for the logistic r
results1 = []
for ds_type in datasets_type:
    print(f'Working on {ds_type}')
    for noise in noise_levels:
        for n in n_samples:
            ds = datasets.query('dataset_name == @ds_type & noise_levels == @noise & r
            print(f'Starting with {ds_type} datasets with noise level {noise} and {n}

            X = ds[['x','y']]
            Y = ds['label']

            for clf_type in clf_types:

                if clf_type == 'log_reg':
                    for regularization_value in regularization_values:
                        log_reg = LogisticRegression(penalty='l2', C=regularization_va
                        score_log = cross_validate(log_reg, X, Y, cv = k, return_trair
                        score_log
                        TRE = score_log['train_score'].mean()
                        TESTE = score_log['test_score'].mean()
                        E_DIFF = TRE - TESTE
                        Reg_HP = regularization_value
                        results1.append((ds_type, n, noise, k, clf_type, Reg_HP, TRE,

                if clf_type == 'svm':
                    for gamma in gamma_range:
                        svm_clf = SVC(C=1, gamma=gamma)
                        score_svm = cross_validate(svm_clf, X, Y, cv = k, return_trair
                        TRE = score_svm['train_score'].mean()
                        TESTE = score_svm['test_score'].mean()
                        E_DIFF = TRE - TESTE
                        Reg_HP = gamma
                        results1.append((ds_type, n, noise, k, clf_type, Reg_HP, TRE,
```

```
Working on circles
Starting with circles datasets with noise level 0 and 10 samples.
Starting with circles datasets with noise level 0 and 50 samples.
Starting with circles datasets with noise level 0 and 100 samples.
Starting with circles datasets with noise level 0 and 1000 samples.
Starting with circles datasets with noise level 0 and 10000 samples.
Starting with circles datasets with noise level 0.1 and 10 samples.
Starting with circles datasets with noise level 0.1 and 50 samples.
Starting with circles datasets with noise level 0.1 and 100 samples.
Starting with circles datasets with noise level 0.1 and 1000 samples.
Starting with circles datasets with noise level 0.1 and 10000 samples.
Starting with circles datasets with noise level 0.2 and 10 samples.
Starting with circles datasets with noise level 0.2 and 50 samples.
Starting with circles datasets with noise level 0.2 and 100 samples.
Starting with circles datasets with noise level 0.2 and 1000 samples.
Starting with circles datasets with noise level 0.2 and 10000 samples.
Working on moons
Starting with moons datasets with noise level 0 and 10 samples.
Starting with moons datasets with noise level 0 and 50 samples.
Starting with moons datasets with noise level 0 and 100 samples.
Starting with moons datasets with noise level 0 and 1000 samples.
Starting with moons datasets with noise level 0 and 10000 samples.
Starting with moons datasets with noise level 0.1 and 10 samples.
Starting with moons datasets with noise level 0.1 and 50 samples.
Starting with moons datasets with noise level 0.1 and 100 samples.
Starting with moons datasets with noise level 0.1 and 1000 samples.
Starting with moons datasets with noise level 0.1 and 10000 samples.
Starting with moons datasets with noise level 0.2 and 10 samples.
Starting with moons datasets with noise level 0.2 and 50 samples.
Starting with moons datasets with noise level 0.2 and 100 samples.
Starting with moons datasets with noise level 0.2 and 1000 samples.
Starting with moons datasets with noise level 0.2 and 10000 samples.
```

In [43]:
```
res1_df = pd.DataFrame(results1, columns = ['ds_type','n', 'noise','k','clf_type','Reg
res1_df
```

Out[43]:

	ds_type	n	noise	k	clf_type	Reg_HP	TRE	TESTE	E_DIFF
0	circles	10	0.0	10	log_reg	0.001	0.555556	0.0000	0.555556
1	circles	10	0.0	10	log_reg	0.010	0.555556	0.0000	0.555556
2	circles	10	0.0	10	log_reg	0.100	0.555556	0.0000	0.555556
3	circles	10	0.0	10	log_reg	1.000	0.555556	0.0000	0.555556
4	circles	10	0.0	10	log_reg	10.000	0.555556	0.0000	0.555556
...	...	...	...	...	...	...	...	...	...
385	moons	10000	0.2	10	svm	0.010	0.865222	0.8654	-0.000178
386	moons	10000	0.2	10	svm	1.000	0.966222	0.9658	0.000422
387	moons	10000	0.2	10	svm	3.000	0.966489	0.9661	0.000389
388	moons	10000	0.2	10	svm	7.000	0.966411	0.9659	0.000511
389	moons	10000	0.2	10	svm	11.000	0.966700	0.9651	0.001600

390 rows × 9 columns

This is the loop for the second range of noise level: [0.3, 0.4, 0.5]

In [23]:
```
datasets_type = ['circles', 'moons']
k = 10
n_samples = [10, 50, 100, 1000, 10000]
noise_levels = [0.3, 0.4, 0.5]
clf_types = ['log_reg', 'svm']
gamma_range = [0.0001, 0.001, 0.01, 1,  3, 7, 11]
regularization_values = [0.001, 0.01, 0.1, 1, 10, 100] # l2-Penalty for the logistic r
results2 = []
for ds_type in datasets_type:
    print(f'Working on {ds_type}')
    for noise in noise_levels:
        for n in n_samples:
            ds = datasets.query('dataset_name == @ds_type & noise_levels == @noise & r
            print(f'Starting with {ds_type} datasets with noise level {noise} and {n}

            X = ds[['x','y']]
            Y = ds['label']

            for clf_type in clf_types:

                if clf_type == 'log_reg':
                    for regularization_value in regularization_values:
                        log_reg = LogisticRegression(penalty='l2', C=regularization_va
                        score_log = cross_validate(log_reg, X, Y, cv = k, return_trair
                        score_log
                        TRE = score_log['train_score'].mean()
                        TESTE = score_log['test_score'].mean()
                        E_DIFF = TRE - TESTE
                        Reg_HP = regularization_value
                        results2.append((ds_type, n, noise, k, clf_type, Reg_HP, TRE,

                if clf_type == 'svm':
                    for gamma in gamma_range:
                        svm_clf = SVC(C=1, gamma=gamma)
                        score_svm = cross_validate(svm_clf, X, Y, cv = k, return_trair
                        TRE = score_svm['train_score'].mean()
                        TESTE = score_svm['test_score'].mean()
                        E_DIFF = TRE - TESTE
                        Reg_HP = gamma
                        results2.append((ds_type, n, noise, k, clf_type, Reg_HP, TRE,
```

```
Working on circles
Starting with circles datasets with noise level 0.3 and 10 samples.
Starting with circles datasets with noise level 0.3 and 50 samples.
Starting with circles datasets with noise level 0.3 and 100 samples.
Starting with circles datasets with noise level 0.3 and 1000 samples.
Starting with circles datasets with noise level 0.3 and 10000 samples.
Starting with circles datasets with noise level 0.4 and 10 samples.
Starting with circles datasets with noise level 0.4 and 50 samples.
Starting with circles datasets with noise level 0.4 and 100 samples.
Starting with circles datasets with noise level 0.4 and 1000 samples.
Starting with circles datasets with noise level 0.4 and 10000 samples.
Starting with circles datasets with noise level 0.5 and 10 samples.
Starting with circles datasets with noise level 0.5 and 50 samples.
Starting with circles datasets with noise level 0.5 and 100 samples.
Starting with circles datasets with noise level 0.5 and 1000 samples.
Starting with circles datasets with noise level 0.5 and 10000 samples.
Working on moons
Starting with moons datasets with noise level 0.3 and 10 samples.
Starting with moons datasets with noise level 0.3 and 50 samples.
Starting with moons datasets with noise level 0.3 and 100 samples.
Starting with moons datasets with noise level 0.3 and 1000 samples.
Starting with moons datasets with noise level 0.3 and 10000 samples.
Starting with moons datasets with noise level 0.4 and 10 samples.
Starting with moons datasets with noise level 0.4 and 50 samples.
Starting with moons datasets with noise level 0.4 and 100 samples.
Starting with moons datasets with noise level 0.4 and 1000 samples.
Starting with moons datasets with noise level 0.4 and 10000 samples.
Starting with moons datasets with noise level 0.5 and 10 samples.
Starting with moons datasets with noise level 0.5 and 50 samples.
Starting with moons datasets with noise level 0.5 and 100 samples.
Starting with moons datasets with noise level 0.5 and 1000 samples.
Starting with moons datasets with noise level 0.5 and 10000 samples.
```

In [44]:
```
res2_df = pd.DataFrame(results2, columns = ['ds_type','n', 'noise','k','clf_type','Reg
res2_df
```

Out[44]:

	ds_type	n	noise	k	clf_type	Reg_HP	TRE	TESTE	E_DIFF
0	circles	10	0.3	10	log_reg	0.001	0.555556	0.0000	0.555556
1	circles	10	0.3	10	log_reg	0.010	0.555556	0.0000	0.555556
2	circles	10	0.3	10	log_reg	0.100	0.555556	0.0000	0.555556
3	circles	10	0.3	10	log_reg	1.000	0.611111	0.0000	0.611111
4	circles	10	0.3	10	log_reg	10.000	0.600000	0.0000	0.600000
...	...	...	...	...	...	...	...	...	...
385	moons	10000	0.5	10	svm	0.010	0.805211	0.8055	-0.000289
386	moons	10000	0.5	10	svm	1.000	0.821044	0.8198	0.001244
387	moons	10000	0.5	10	svm	3.000	0.822011	0.8203	0.001711
388	moons	10000	0.5	10	svm	7.000	0.823067	0.8211	0.001967
389	moons	10000	0.5	10	svm	11.000	0.823767	0.8207	0.003067

390 rows × 9 columns

Here I appended the final results into a final dataframe called results.

In [47]:
```
results_final = pd.DataFrame([])
results_final = results_final.append(res1_df)
results_final = results_final.append(res2_df)
results_final.shape
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8836\873330506.py:2: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future ver
sion. Use pandas.concat instead.

C:\Users\User\AppData\Local\Temp\ipykernel_8836\873330506.py:3: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future ver
sion. Use pandas.concat instead.
```

Out[47]:
```
(780, 9)
```

Finally I exported the results as a csv for further uses. This was to save time running the loop everytime.

In [14]:
```
results_final.to_csv("results/res_df.csv")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [14], in <cell line: 1>()
----> 1 results_final.to_csv("results/res_df.csv")

NameError: name 'results_final' is not defined
```

Therefore for further investigations in future days I called this code instead of re-running the two loops above.

In [12]:
```
res_df = pd.read_csv("results/res_df.csv")
res_df.shape
```

Out[12]:
```
(780, 10)
```

In [13]:
```
res_df.sample(5)
```

Out[13]:

	Unnamed: 0	ds_type	n	noise	k	clf_type	Reg_HP	TRE	TESTE	E_DIFF
578	188	circles	10000	0.5	10	svm	0.0001	0.501178	0.4894	0.011778
728	338	moons	50	0.5	10	log_reg	0.0010	0.524444	0.2800	0.244444
188	188	circles	10000	0.2	10	svm	0.0001	0.501822	0.4836	0.018222
137	137	circles	10	0.2	10	svm	0.0010	0.555556	0.0000	0.555556
141	141	circles	10	0.2	10	svm	7.0000	0.911111	0.2000	0.711111

From the table above it is difficult to understand the results of our experiment. Therefore we will use some graphics in order to **evaluate which model works better**.

The advantage of using **widgets** is that the graphs are clearer and not overloaded. Moreover, they are a friendly tool to choose between the different options that we run before.

The results are visualized using a **line plot** from the plotly library. The graphs will show the train error and the test error (*metrics*).

In [14]:
```
data = widgets.Dropdown(options=res_df['ds_type'].unique(), description="Select Data")
sample = widgets.Dropdown(options=res_df['n'].unique(), description="Select Sample")
noise_lv = widgets.Dropdown(options=res_df['noise'].unique(), description="Select nois
model = widgets.Dropdown(options=res_df['clf_type'].unique(), description="Select Mode
param = widgets.Dropdown(options =res_df['Reg_HP'].unique(), description="Select Param
```

In [15]:
```
@interact
def res_plot(dataset = data, n_sample = sample, noise_level = noise_lv, model_cl =  mo
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & n == @
    fig = px.line(set_selection, x = 'Reg_HP', y = ['TRE','TESTE'], title = 'Model eva
    fig.update_xaxes(title_text='Regularization Parameter/Gamma', showgrid=False, line
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

The plot above shows us the metrics from the two models as a function of the regularization parameter (C for logistic regression and gamma for svm model). We will evaluate each model separately and then we will make a comparison between them.

Let's start with the **logistic regression** model. For this model we use the l2 ridge relarization choice with different hyperparamerts values. In general, we can say that using a larger number of folders in cross validation throws better results both for the train score and the test score as well as the difference between them. When we use a larger number of folders we are automatically forced to build a far less overfitted (and thus more generalisable model) because we are trying to maximise the average performance over many validation sets. For lower levels of regularization paramaters, ie non-penalty applied, the model tends to overfit (low bias between the train and the test but high variance, that is lower score values). Nevertheless, for higher regularization paramaters, the score of the train and the test improves but the difference between them is higher. After a regularization value of 1 we don't appreciate better results.

We will continue evaluating the **svm model**. Here we *play* with the hyperparameter gamma. Gamma decides that how much curvature we want in a decision boundary. Gamma high means more curvature. Gamma low means less curvature. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. If gamma is very small, the model is too constrained and cannot capture the complexity of the data. On the other hand, if the value of gamma is too large, then the model can overfit and be prone to low bias/high variance. As we could see in the graph, the lower values of gamma result in models with lower accuracy. However, after a certain point (gamma > 1 and onwards), the model accuracy decreases. **Therefore we conclude that a svm model with a gamma value of 1 gives an acceptable result.**

Now if we compare logistic regression with svm model we can see that svm is a better model of this experiment and throws better results. This is because the algorithms are trained based on non-linear databases (circles/moons). Therefore, from the start it is clear that logist regression is less apropriate for our database. In order words, when the dataset is non-linear, it is recommended to use kernel functions such as RBF instead of linear regression models

We will continue to evaluate the results as a **function of the sample size** of our database.

In [16]:
```
@interact
def res_plot(dataset = data, noise_level = noise_lv, model_cl =  model, hparam =param)
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & Reg_HP
    fig = px.line(set_selection, x = 'n', y = ['TRE','TESTE'], title = 'Model evaluati
                  labels={'TRE': "Train Error", 'TESTE': "Test Error"})
    fig.update_xaxes(title_text='Sample Size', showgrid=False, linecolor='black', mirr
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

Answers to the questions

1

Q1 - For SVM only, For dataset of size 10k and for each dataset, What are the best model params? How stable is it?

A1 - First we will try to solve the problem *visually* from the next interactive plot:

In [17]:
```
@interact
def res_plot(dataset = data, noise_level = noise_lv):
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & n == 1
    fig = px.line(set_selection, x = 'Reg_HP', y = ['TRE','TESTE'], title = 'Model eva
    fig.update_xaxes(title_text='Hyperparameter Gamma', showgrid=False, linecolor='bla
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

Hyperparameter Gamma controls overfitting in SVM. The higher the gamma, the higher the hyperplane tries to match the training data. Therefore, choosing an optimal gamma to avoid overfitting as well as underfitting is the key.

We are trying to find a gamma which results in a minimum difference between the training score and the test score (variance) but at the same time that gives a high score.

We will start with the **circles dataset**. We explored the results with different levels of noise and the conclusion is the same:

- for values of **gamma less than 1** the accuracy decreases and also the difference betweem the training score and the test score is greater
- for values of **gamma greater than 1** we can see that gamma leads to overfitting as the classifier tries to perfectly fit the training data (there isn't any difference between TRE and TESTE).

*Therefore for **circles** dataset we will choose a gamma equals to 1.*

Now, we will move to the **moons dataset**. Overall, we can see that the accuracy score is better than for circles. However the conclusions we reached for gamma are the same and we will choose a **gamma value of 1**.

2

Q2 - For SVM only, For dataset of size 10k and for each dataset, What is the most stable model and model params? How good is it in comparison to other models? Explain using bias and variance terminoligy.

A2 - As before we will try to understand the results with the help of some visualizations.

In [18]:
```
@interact
def res_plot(dataset = data, noise_level = noise_lv,model_cl =  model,):
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & n == 1
    fig = px.line(set_selection, x = 'Reg_HP', y = ['TRE','TESTE'], title = 'Model eva
    fig.update_xaxes(title_text='Hyperparameter Value', showgrid=False, linecolor='bla
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

When we investigated the data before we saw that we are dealing with **non-linear patterns**. That was a sign that the logistic regression model was less apropriate for this experiment and therefore svm was a better choice. *But not any svm...* We use a svm with a radial basis function that suits this kind of data.

Circles

- Logistic Regression - There is a high variance no matter which kind of noise level we choose. This is because the model is less apropriate. The variance is high and consistent for every regularization level. Nevertheless for smaller values of C (strong regularization) the accuracy score increases. This is because we constrain the complexity of the model during the training process. Anyway the variace between the TRE and the TEST in unacceptable.
- SVM RBF kernel - The accuracy level is higher than the result we saw for the logistic regression algorithm. For large values of gamma the model overfits: the performance of the

model on the training dataset is better than the performance on the test dataset.

Moons

- Logistics Regression - The results are better than for circles dataset and improve where the noise level is low. A high value of C tells the model to give high weight to the training data, and a lower weight to the complexity penalty. A low value tells the model to give more weight to this complexity penalty at the expense of fitting to the training data. We can see in the graph that for low regularization parameters of C the accuracy is worse than for higher values. This is because for higher values the model overfits and gives all the weight to the training data as we said before.
- SVM RBF kernel - As expected, the accuracy is higher and decreases as the noise level increases. We can see that for higher gamma values the model tries to exactly fit the training data set, ie less generalized (or overfitted). Thefore for higher values of gamma we can also appreciate that the difference between the training and the test scores are greater.

Conclusion - For both sets of data svm model throws better results than the logistic regression algorithm. A gamma value of 1 should be chosen.

3

Q3 - Does regularization help for linear models? consider different datasets sizes.

A3 - In the graph below we can see that for large values of C the accuracy is better. Large values of C mean that the power of regularization is weaker and thefore the model tends to overfit and gives more weight to the training data. NEVERTHELESS, we don't see a high variance betweem the train and the test (which is the opposite of overfitting, strange...). We don't see a big improvement for sample sizes greater than 1000 observations.

In [19]:
```
C = widgets.Dropdown(options = [0.001, 0.01, 0.1, 1, 10, 100], description="Select C")
```

In [20]:
```
@interact
def res_plot(dataset = data, noise_level = noise_lv, hparam =C):
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & Reg_HP
    fig = px.line(set_selection, x = 'n', y = ['TRE','TESTE'], title = 'Model evaluati
                  labels={'TRE': "Train Error", 'TESTE': "Test Error"})
    fig.update_xaxes(title_text='Sample Size', showgrid=False, linecolor='black', mirr
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

4

Q4 - For a given noise level of your chioce, How does the train, test and difference error changes with increasing data sizes? (answer for svm and LR seperatly)

A4 - In order to answer this question we will visualize the training score, the test score and the difference between them with a plot called: ***learning curve***.

In [21]:
```
@interact
def res_plot(dataset = data ,model_cl =  model, hparam = param):
    set_selection = res_df.query('ds_type == @dataset & noise == 0.3 & clf_type == @mc
    fig = px.line(set_selection, x = 'n', y = ['TRE','TESTE', 'E_DIFF'], title = 'Mode
    fig.update_xaxes(title_text='Sample Size', showgrid=False, linecolor='black', mirr
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

This plot shows us if the accuracy score can be maximized when we increase the sample size. We can state this differently and understand how the difference between the training score and the test score is minimazed when the sample size increases.

We will evaluate each model separately.

Circles

We will start with Logistic Regression. Altough it's not that easy to identify from the plot above. we can see that when n size is small there a high variance. The variance decreases as we reached a sample size of 1000 observations. For sample sizes greater than 1000 observations, the difference between the training and the test score increases very slowly and at the same time the score decreases. Thefore it doesn't seem that the model continues to learn when we increase the sample size above 1000 observations.

We will continue with the svm algorithm. Here, we see a different picture. The model rather improves when the same size increases. In other words, the accuracy score and also the variance of the model increases as the number of observations is greater. But when gamma is larger the model increases less when the size of the sample is greater.

Moons

Logistic Regression - For small values of C we see that the model improves when we increase the number of observations in the sample.

SVM - Overall we received better results than for circles dataset. These results become better when the sample size is greater. Nevertheless, when for larger gammas when the sample size increases the model stop improving that becomes stable for large n.

5

Q5- For a given noise level of your chioce, How does the train, test and difference error changes with increasing model complexity? (answer for svm and LR seperatly)

A5 - I chose a 0.3 noise level.

In [22]:
```
@interact
def res_plot(dataset = data ,model_cl =  model, n_sample = sample):
    set_selection = res_df.query('ds_type == @dataset & noise == 0.3 & clf_type == @mo
    fig = px.line(set_selection, x = 'Reg_HP', y = ['TRE','TESTE', 'E_DIFF'], title =
    fig.update_xaxes(title_text='Hyperparameter Value', showgrid=False, linecolor='bla
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

Logistic Regression - C and regularization strength are **negatively correlated** (smaller the C is stronger the regularization will be). Therefore for greater C we expect that the model becomes more complex and prone to overfitting. However, we don't see this for the range of parameter we chose (there is not a big difference between the train and the test scores when C increases). The results are better for the moons data set in comparison with circles dataset. This is as expected because the non-linear pattern of the data.

SVM - The performance of the model is also better for moons dataset in comparison with cicles dataset. It is hard to see for the range of gamma we chose but it seems that when gamma becomes very great the model becomes overfitted that the variance increases (difference between the training score and the test score). This is the model starts to fit itself to the noise and tries to fit the training set independently from the test performance.

In [ ]:
```
```

6

Q6 - Are the noise level effect the number of datapoints needed to reach optimal test results?

A6 - From the following plot we can see that when the noise level increases we need **more observations** to reach the same results (same accuracy).

In [23]:
```
@interact
def res_plot(dataset = data, noise_level = noise_lv, model_cl =  model, hparam =param)
    set_selection = res_df.query('ds_type == @dataset & noise == @noise_level & Reg_HP
    fig = px.line(set_selection, x = 'n', y = ['TRE','TESTE'], title = 'Model evaluati
                  labels={'TRE': "Train Error", 'TESTE': "Test Error"})
    fig.update_xaxes(title_text='Sample Size', showgrid=False, linecolor='black', mirr
    fig.update_yaxes(title_text='Accuracy', showgrid=False, linecolor='black', mirror=
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

In the following graphs we compare each level of noise separately. We can see how the noise level affects the accuracy of the prediction.

In [24]:
```
@interact
def res_plot(dataset = data,  model_cl =  model, hparam =param):
```

```
    set_selection = res_df.query('ds_type == @dataset & Reg_HP == @hparam & clf_type =
    fig = px.line(set_selection, x = 'n', y = ['TRE','TESTE'], title = 'Model evaluati
                  labels={'TRE': "Train Error", 'TESTE': "Test Error"},facet_col='noise
    fig.update_xaxes(title_text='Sample Size', showgrid=True, linecolor='black', mirro
    fig.update_yaxes(title_text='Accuracy', showgrid=True, linecolor='black', mirror=T
    fig.update_layout(legend_title="Metrics")
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Data', options=('circles', 'moon
s'), value='circles'), Drop…
```

For the **moons** dataset we see less variance of the accuracy compare to **circles** data.

This is because the pattern of the circles dataset is more complex.

It is interesting to see that for the circles dataset the logistic regression gives us worse results in comparison with the svm model but the accuracy level is more stable (**high variance, less bias**).

Overall **svm** throws better results.

BONUS

BONUS QUESTION - For SVM: Select one dataset and with 0.2 noise level. Identify the optimal model params, and visualize the decision boundry learned.

BONUS ANSWER

First we will take an specific dataset from our data in order to tune the model parameters. For this purpose I chose:

- moons
- noise level of 0.2
- sample size of 1000. But first we are going to return the label to its original values in order to plot the results afterwards.

In [25]:
```
datasets_plot = datasets.copy()
```

In [26]:
```
datasets_plot['num_label'] = datasets_plot['label'].apply(lambda x: 0 if x == 'A' else
datasets_plot.head(1)
```

Out[26]:

	x	y	label	dataset_name	n_samples	noise_levels	num_label
0	-1.0	1.224647e-16	A	moons	5	0.0	0

In [27]:
```
select_plot = datasets_plot.query('dataset_name == "moons" & n_samples == 1000 & noise
select_plot.head(1)
```

Out[27]:

	x	y	label	dataset_name	n_samples	noise_levels	num_label
0	1.536536	-0.838825	B	moons	1000	0.2	1

The next piece of code will help us to show the results in a graphic way.

In [28]:
```
def versiontuple(v):
    return tuple(map(int, (v.split("."))))


def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >= versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :], y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    alpha=1.0,
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
```

In [29]:
```
X_xor = select_plot[['x','y']].to_numpy()
y_xor = select_plot['num_label'].to_numpy()
```

In [30]:
```
gamma_widget = widgets.Dropdown(options = [0.0001, 0.001, 0.01, 1,  3, 7, 11], descrip
C_widget = widgets.Dropdown(options = [0.0001, 0.001, 0.01, 1,  10, 100], description=
```

In [31]:
```
@interact
def tunning_gamma(gamma = gamma_widget):
    svm = SVC(kernel = 'rbf', gamma = gamma)

    svm.fit(X_xor, y_xor) # train the model

    plot_decision_regions(X_xor, y_xor, classifier=svm) #Visualize the decision bounda
    plt.legend(loc='upper left')
```

```
    plt.tight_layout()
    plt.show()
```

```
interactive(children=(Dropdown(description='Select Gamma', options=(0.0001, 0.001, 0.
01, 1, 3, 7, 11), value=0…
```

We can the same picture in the following table:

In [32]:
```
tbl_res = res_df.query('ds_type == "moons" & noise == 0.2 & clf_type == "svm" & n == 1
tbl_res.sort_values(['Reg_HP', 'TRE' , 'TESTE','E_DIFF'],
              ascending = [False, True, True, False])
```

Out[32]:

	Unnamed: 0	ds_type	n	noise	k	clf_type	Reg_HP	TRE	TESTE	E_DIFF
376	376	moons	1000	0.2	10	svm	11.0000	0.967333	0.960	0.007333
375	375	moons	1000	0.2	10	svm	7.0000	0.967333	0.961	0.006333
374	374	moons	1000	0.2	10	svm	3.0000	0.968000	0.963	0.005000
373	373	moons	1000	0.2	10	svm	1.0000	0.965444	0.964	0.001444
372	372	moons	1000	0.2	10	svm	0.0100	0.844667	0.845	-0.000333
371	371	moons	1000	0.2	10	svm	0.0010	0.798333	0.798	0.000333
370	370	moons	1000	0.2	10	svm	0.0001	0.531667	0.500	0.031667

In [33]:
```
@interact
def tunning_gamma_and_c(gamma = gamma_widget, C = C_widget):
    svm = SVC(kernel = 'rbf', gamma = gamma, C = C)

    svm.fit(X_xor, y_xor) # train the model

    plot_decision_regions(X_xor, y_xor, classifier=svm) #Visualize the decision bounda
    plt.legend(loc='upper left')
    plt.tight_layout()
    plt.show()
```

```
interactive(children=(Dropdown(description='Select Gamma', index=3, options=(0.0001,
0.001, 0.01, 1, 3, 7, 11)…
```

Conclusion

1.Gamma hyperpamater tunning (gamma only)

Gamma less than 1 doesn't catch the pattern of our data. For models with gamma greater than 1 the algorithm adjust itself to the noise and becomes less generalized ("overfitting").

*Finally we will choose a **gamma value of 1** for the database selected with C = 1.*

2.Gamma and C hyperpamaters tunning Once we chose Gamma we tries to evaluate our model with different C's. C is the regularization term in our model. When we select C very small it penalizes our model for being over-complicated. BUT, on the other hand very small C gives poor classifications results. If we choose C very high then the model tries to fit the noise and match the training data "too well". Therefore we will choose the intermediate values of C.

Therefore we will conclude that a **C of 1**.