svm-with-kernels

August 14, 2020

1 SVMs and Kernels

In this notebook you will fit a Support Vector Machine (SVM) classifer to data using scikit-learn. You will use different kernels and see how kernels produce nonlinear decision surfaces. You will also predict the labels for datapoints and measure the performance of the SVM.

1.1 Before you start

- In order for the notebooks to function as intended, modify only between lines marked "### begin your code here (__ lines)." and "### end your code here.".
- The line count is a suggestion of how many lines of code you need to accomplish what is asked.
- You should execute the cells (the boxes that a notebook is composed of) in order.
- You can execute a cell by pressing Shift and Enter (or Return) simultaneously.
- You should have completed the previous Jupyter notebooks before attempting this one as the concepts covered there are not repeated, for the sake of brevity.

1.2 Loading the appropriate packages

We will import SVM classifier class (SVC) as well some other packages you will use.

[1]:
```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import plotly.graph_objs as go
```

Let's turn off the scientific notation for floating point numbers.

[2]:
```
np.set_printoptions(suppress=True)
```

1.3 Loading and examining the data

We will load our data from a CSV file and put it in a pandas an object of the `DataFrame` class.

This dataset is the breast cancer Wisconsin (diagnostic) dataset which contains 30 different features computed from a images of a fine needle aspirate (FNA) of breast masses for 569 patients with each example labeled as being a *benign* or *malignant* mass.

- This was taken and modified from the Machine Learning dataset repository of School of Information and Computer Science of University of California Irvine (UCI):

  Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3]:
```
df = pd.read_csv('data_svms_and_kernels.csv')
```

Let's take a look at the data:

[4]:
```
df
```

[4]:
```
     Feature 1  Feature 2 Label
0     0.109478  -0.168109     B
1    -0.590131  -0.153594     B
2     0.157117   0.059849     A
3    -0.623096  -0.638964     B
4    -0.296598   0.315015     B
5    -0.498929  -0.406902     B
6     0.950000   0.445315     A
7    -0.520393  -0.692691     B
8    -0.441623  -0.425413     B
9    -0.054541  -0.419501     A
10    0.408677  -0.246902     A
11    0.006048  -0.416950     A
12    0.087732  -0.158091     B
13   -0.357941  -0.757369     B
14   -0.061865  -0.602031     A
15    0.298028  -0.036692     A
16    0.615872  -0.341219     A
17   -0.364023   0.688679     B
18    0.462646  -0.430153     A
19   -0.429533  -0.351926     B
20    0.201710  -0.298721     A
21    0.305349   0.280865     A
22   -0.182093  -0.173068     B
23   -0.094301   0.554999     B
24   -0.354164  -0.325979     B
25    0.123313  -0.253753     B
26    0.121789  -0.447799     A
27   -0.830984   0.138626     B
28    0.186753   0.626921     A
29    0.237747  -0.216044     A
```

```
..        ...       ...  ...
262  -0.558191  0.606706   B
263  -0.199311  0.694422   B
264  -0.105989 -0.261404   B
265   0.062324 -0.094640   B
266  -0.351144 -0.223643   B
267   0.579056 -0.372607   A
268  -0.551362  0.337245   B
269  -0.336412  0.399937   B
270   0.032174  0.666256   A
271   0.575696  0.303118   A
272  -0.314009  0.200799   B
273  -0.267233 -0.282806   B
274  -0.329739 -0.763111   B
275  -0.333549 -0.519628   A
276  -0.435829 -0.336952   B
277  -0.405941 -0.616722   B
278  -0.647305  0.372530   B
279  -0.390971 -0.476825   B
280  -0.465107  0.310568   B
281   0.324442 -0.561250   A
282  -0.384929 -0.203485   B
283  -0.424816  0.182371   B
284   0.011478  0.550888   A
285  -0.647072  0.289507   B
286  -0.347576 -0.261193   B
287  -0.218594  0.302159   B
288  -0.283552  0.481753   B
289   0.299025  0.073648   A
290   0.489956 -0.092362   A
291   0.059036 -0.128766   B

[292 rows x 3 columns]
```

To do that we first need to extract our data from the dataframe in NumPy arrays. We use `LabelEncoder` from scikit-learn to transform labels into $\{-1, +1\}$:

[5]:
```
X = df.drop('Label', axis=1).to_numpy()
y_text = df['Label'].to_numpy()
y = (2 * LabelEncoder().fit_transform(y_text)) - 1
```

Let's check `X`, `y_text` and `y`:

[6]:
```
X
```

[6]:
```
array([[ 0.10947806, -0.16810863],
       [-0.59013059, -0.15359437],
       [ 0.15711731,  0.0598494 ],
       [-0.62309568, -0.6389643 ],
       [-0.29659811,  0.31501529],
       [-0.49892949, -0.40690183],
       [ 0.95      ,  0.44531528],
       [-0.52039318, -0.69269133],
       [-0.44162284, -0.42541306],
       [-0.05454123, -0.41950099],
       [ 0.40867744, -0.24690175],
       [ 0.00604817, -0.41695007],
       [ 0.08773186, -0.15809074],
       [-0.35794094, -0.75736888],
       [-0.06186495, -0.60203068],
       [ 0.29802828, -0.0366922 ],
       [ 0.61587189, -0.34121949],
       [-0.36402275,  0.68867891],
       [ 0.46264578, -0.43015301],
       [-0.42953288, -0.35192589],
       [ 0.20170953, -0.29872133],
       [ 0.30534941,  0.28086526],
       [-0.18209344, -0.17306767],
       [-0.09430147,  0.55499874],
       [-0.35416442, -0.32597868],
       [ 0.12331348, -0.25375275],
       [ 0.12178891, -0.44779879],
       [-0.83098402,  0.13862644],
       [ 0.18675303,  0.62692114],
       [ 0.23774694, -0.2160437 ],
       [ 0.01280136, -0.10306835],
       [-0.58788614,  0.2328265 ],
       [-0.33981138,  0.42402853],
       [ 0.50424463, -0.09850316],
       [ 0.31847989, -0.01378183],
       [-0.24185941,  0.3906835 ],
       [-0.34905506,  0.65051595],
       [-0.24976651, -0.12806583],
       [ 0.11487393, -0.0677395 ],
       [ 0.21029666,  0.57550662],
       [ 0.15305979,  0.73257694],
       [-0.64699855,  0.01804888],
       [-0.38276538,  0.35483302],
       [ 0.34184204, -0.46275368],
       [-0.25505921,  0.11084074],
       [ 0.41771808, -0.04210843],
       [-0.45537438,  0.17402679],
       [ 0.54522517,  0.00891035],
       [-0.29653574,  0.36615901],
       [-0.17168605,  0.00157501],
       [ 0.49246996, -0.40446088],
       [ 0.35105068, -0.26081764],
       [ 0.00658456, -0.34541823],
       [ 0.42143635, -0.36543951],
       [ 0.48475567,  0.14582741],
       [-0.65239676,  0.02702193],
       [-0.55105298,  0.44725619],
       [-0.18544122, -0.54559629],
       [ 0.36972037,  0.12483846],
       [ 0.61827958,  0.03048891],
       [ 0.58217985, -0.16352677],
       [ 0.53658606,  0.03455681],
       [ 0.36386885, -0.2100743 ],
       [-0.44488872,  0.52365989],
       [-0.29574299, -0.60437483],
       [-0.54359876,  0.07105291],
       [-0.19289551, -0.64355731],
       [-0.16035814, -0.49274627],
       [ 0.55323827, -0.16507608],
       [-0.1657153 ,  0.48366647],
       [ 0.00452934,  0.44071472],
       [ 0.36898605, -0.06377325],
       [-0.33355565,  0.01470178],
       [-0.35356622, -0.0694304 ],
       [-0.03369164, -0.11870204],
       [ 0.2922315 , -0.36504464],
       [ 0.16608386,  0.64938539],
       [-0.2369668 , -0.40407545],
       [-0.26812804,  0.35805673],
       [-0.50663352,  0.18965871],
       [-0.52347977,  0.19373447],
       [-0.21617044,  0.21614382],
       [-0.53201056,  0.51510594],
       [ 0.30841119, -0.49019986],
       [-0.56886384,  0.16891848],
       [-0.55017633, -0.33706866],
       [-0.02582336, -0.05380214],
       [ 0.25728228, -0.63211531],
       [-0.57403706, -0.41289154],
       [ 0.37259612,  0.00671809],
       [-0.40075173,  0.44281998],
       [-0.15100252,  0.61971203],
       [ 0.44446585,  0.31542248],
       [-0.36485107, -0.55212242],
       [ 0.36083537,  0.30006144],
       [-0.62020671,  0.2986686 ],
       [ 0.22469588,  0.6624873 ],
       [ 0.5103637 , -0.09946942],
       [ 0.42103489, -0.72994983],
       [-0.22103095,  0.51717954],
       [ 0.28511551, -0.49693248],
       [ 0.94370792,  0.38083883],
       [ 0.31078368, -0.66258554],
       [ 0.71759366,  0.38631055],
       [ 0.1758217 , -0.62737634],
       [-0.58731298, -0.0199587 ],
       [-0.86101569, -0.25795087],
       [ 0.16127931, -0.56096963],
       [ 0.06156973, -0.54297549],
       [ 0.84620131,  0.45548557],
       [-0.09254279, -0.01315666],
       [ 0.26962303, -0.05243128],
       [-0.12453081,  0.00178644],
       [ 0.35522457,  0.59158232],
       [ 0.5466359 , -0.09612192],
       [-0.55699478,  0.59714698],
       [-0.24267904,  0.10754823],
       [-0.26854335, -0.32727168],
       [ 0.33571793, -0.39497837],
       [-0.30354934, -0.07808274],
       [-0.37126358,  0.66393038],
       [-0.47245256,  0.15216231],
       [-0.53319886,  0.38027243],
       [ 0.71406493,  0.33589508],
       [ 0.12242774, -0.40841125],
       [-0.32310544,  0.53645147],
       [-0.30641556, -0.33539996],
       [ 0.71164653,  0.25917957],
       [ 0.47511619,  0.08995622],
       [-0.33341163, -0.4389798 ],
       [-0.95      ,  0.17912225],
       [-0.28019476, -0.06935188],
       [-0.18253304,  0.45237541],
       [ 0.82683842,  0.43688497],
       [ 0.27747116, -0.6621803 ],
       [ 0.10359266,  0.57713052],
       [ 0.52786296, -0.28763214],
       [-0.41246136,  0.3586479 ],
       [ 0.14370368,  0.54425715],
       [ 0.27733967,  0.54311787],
       [-0.45042965,  0.18289014],
       [ 0.14596343, -0.48043336],
       [-0.559731  ,  0.00786543],
       [ 0.5726081 , -0.04598575],
       [-0.4900432 , -0.7673623 ],
       [-0.45709495, -0.22459855],
       [-0.03644853, -0.15055228],
       [-0.47135937, -0.54114883],
       [-0.77765276, -0.19766355],
       [-0.03873147, -0.33773999],
       [ 0.51372166, -0.17598387],
       [ 0.30922792,  0.55788742],
       [ 0.36370993,  0.38403647],
       [ 0.39116789, -0.19345306],
       [-0.42808901, -0.48258803],
       [ 0.32050244,  0.46737781],
       [-0.07469738,  0.02138221],
       [ 0.46541007,  0.25252525],
       [-0.37488273, -0.23727361],
       [-0.40653295,  0.76025886],
       [ 0.39822857, -0.18909212],
       [ 0.10605463, -0.46535594],
       [ 0.23537156,  0.00054866],
       [ 0.04081042, -0.63916267],
       [-0.02285654, -0.52117849],
       [ 0.53604812, -0.0551825 ],
       [ 0.35034194,  0.06032118],
       [ 0.22182354, -0.33486145],
       [-0.2718561 , -0.03408501],
       [-0.37497291,  0.24779763],
       [-0.38072976,  0.13920051],
       [-0.44033314, -0.7212575 ],
       [ 0.37204359, -0.48007533],
       [-0.58616224, -0.06528723],
       [-0.36708384, -0.73942022],
       [ 0.14499755, -0.18475441],
       [-0.36883542,  0.49074375],
       [ 0.14285159, -0.69858607],
       [-0.2050539 , -0.01836178],
       [ 0.28452175,  0.49235246],
       [-0.22190634,  0.09335128],
       [ 0.25574639, -0.03581583],
       [ 0.58892084,  0.33238703],
       [-0.22924671, -0.45900246],
       [ 0.44126719, -0.03626334],
       [ 0.45108545, -0.09842275],
       [ 0.6986282 ,  0.45200422],
       [ 0.41812118,  0.02878464],
       [-0.56482546, -0.68914992],
       [-0.1480793 , -0.47467886],
       [-0.2533188 ,  0.57693531],
       [ 0.02926851,  0.65237165],
       [ 0.05544668, -0.66386534],
       [-0.56256855, -0.83631991],
       [ 0.10887484, -0.1891335 ],
       [ 0.43219286,  0.50941504],
       [-0.45662014, -0.00178694],
       [ 0.12274991,  0.73413976],
       [ 0.38192669,  0.26769968],
       [-0.122057  ,  0.09671452],
       [-0.41030691, -0.01411626],
       [ 0.27409202,  0.00832594],
       [ 0.32510458, -0.4126832 ],
       [-0.38692513, -0.01562016],
       [ 0.28183606, -0.45911068],
       [-0.48237333, -0.45253242],
       [ 0.17038796, -0.59227358],
       [ 0.30998515, -0.39524186],
       [ 0.30413988,  0.64863975],
       [-0.4945288 ,  0.04652849],
       [-0.03619394, -0.48365973],
       [-0.52673481, -0.55064066],
       [-0.48695518, -0.12863599],
       [ 0.01734916,  0.60077814],
       [-0.80220752, -0.78594304],
       [-0.40748619,  0.42392262],
       [ 0.09887797, -0.5128542 ],
       [ 0.57737283,  0.19890415],
       [ 0.50200853, -0.16210132],
       [ 0.20826014, -0.48271211],
       [-0.38897649,  0.55628685],
       [-0.4122792 , -0.53180102],
       [ 0.66450328,  0.0086695 ],
       [-0.60772909,  0.01855978],
       [ 0.40855985, -0.12705581],
       [-0.4830364 ,  0.35816959],
       [ 0.39733466,  0.34039937],
       [ 0.4848405 , -0.47584074],
       [ 0.14910589, -0.71341879],
       [ 0.00626717, -0.71141276],
       [-0.05860654, -0.57427448],
       [-0.22911849,  0.39592907],
       [ 0.85551584,  0.30045243],
       [-0.75157323,  0.15230302],
       [-0.55353949,  0.47897151],
       [-0.34670668, -0.09926295],
       [ 0.34872461, -0.12814485],
       [-0.20912419, -0.57410369],
       [-0.43085449,  0.22602256],
       [-0.35736901,  0.56172411],
       [-0.09423767, -0.26405508],
       [-0.45756735,  0.51889855],
       [ 0.18704539, -0.65502821],
       [-0.10603405,  0.14787656],
       [ 0.36128554, -0.37890829],
       [-0.02004174, -0.39005356],
       [ 0.20346049, -0.49465809],
       [-0.6793655 , -0.23421794],
       [-0.44046487, -0.62083552],
       [-0.39890035,  0.00655325],
       [-0.48078389,  0.05215678],
       [ 0.86684966,  0.41346103],
       [-0.53653532,  0.06974244],
       [-0.22646654, -0.28176582],
       [ 0.21704757,  0.5676625 ],
       [ 0.35446191, -0.46046418],
       [-0.48025652, -0.84597349],
       [ 0.0488998 , -0.60809186],
       [-0.41618818, -0.36297431],
       [ 0.08670551, -0.04640797],
       [-0.22458851, -0.4419616 ],
       [-0.41307522,  0.31435526],
       [-0.55819068,  0.60670584],
       [-0.19931058,  0.69442242],
       [-0.10598892, -0.26140434],
       [ 0.06232402, -0.09463954],
       [-0.35114423, -0.22364289],
       [ 0.5790559 , -0.37260679],
       [-0.55136244,  0.33724535],
       [-0.33641219,  0.39993685],
       [ 0.03217362,  0.66625605],
       [ 0.57569646,  0.30311808],
       [-0.31400859,  0.2007991 ],
       [-0.26723284, -0.28280559],
       [-0.32973854, -0.76311052],
       [-0.33354858, -0.51962772],
       [-0.4358291 , -0.33695216],
       [-0.4059409 , -0.61672197],
       [-0.64730506,  0.37252971],
       [-0.39097064, -0.47682452],
       [-0.46510747,  0.31056764],
       [ 0.32444237, -0.5612503 ],
       [-0.38492934, -0.20348534],
       [-0.42481612,  0.18237085],
       [ 0.01147798,  0.55088838],
       [-0.64707214,  0.28950725],
       [-0.34757588, -0.2611931 ],
       [-0.21859391,  0.30215913],
       [-0.28355182,  0.48175277],
       [ 0.29902474,  0.07364831],
       [ 0.48995608, -0.09236156],
       [ 0.05903629, -0.12876579]])
```

[7]:
```
X.shape
```

[7]:
```
(292, 2)
```

Let's do the same thing for `y_text`:

[8]:
```
y_text
```

[8]:
```
array(['B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B',
       'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B',
       'A', 'B', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'B', 'B', 'B', 'B',
       'A', 'A', 'B', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'A',
       'A', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B',
       'B', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'B',
       'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'B',
       'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A',
       'A', 'B', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B',
       'B', 'A', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'B',
       'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'B',
       'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A',
       'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B',
       'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'A',
       'A', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B',
       'A', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'A', 'A',
       'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'B',
       'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B',
       'B', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'A',
       'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'B',
       'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'B',
       'B', 'B', 'B', 'A', 'A', 'B'], dtype=object)
```

…and for shape of `y_text`:

[9]:
```
y_text.shape
```

[9]:
```
(292,)
```

Finally, let's check y:

[10]:
```
y
```

[10]:
```
array([ 1,  1, -1,  1,  1,  1, -1,  1,  1, -1, -1, -1,  1,  1, -1, -1, -1,
        1, -1,  1, -1, -1,  1,  1,  1,  1, -1,  1, -1, -1,  1,  1,  1, -1,
       -1,  1,  1,  1,  1, -1, -1,  1,  1, -1,  1, -1,  1, -1,  1,  1, -1,
       -1, -1, -1, -1,  1,  1, -1, -1, -1, -1, -1, -1,  1,  1,  1, -1, -1,
       -1,  1,  1, -1,  1,  1,  1, -1, -1,  1,  1,  1,  1,  1,  1, -1,  1,
        1,  1, -1,  1, -1,  1,  1, -1,  1, -1,  1, -1, -1, -1,  1, -1, -1,
       -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,  1, -1, -1,  1,  1,  1, -1,
        1,  1,  1,  1, -1, -1,  1,  1, -1, -1,  1,  1,  1,  1, -1, -1, -1,
       -1,  1, -1, -1,  1, -1,  1, -1,  1,  1,  1,  1,  1, -1, -1, -1, -1,
       -1,  1, -1,  1, -1,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1,  1,  1,
        1,  1, -1,  1,  1,  1,  1, -1,  1, -1,  1, -1, -1,  1, -1, -1, -1,
       -1,  1, -1,  1, -1, -1,  1,  1, -1,  1, -1, -1,  1,  1, -1, -1,  1,
       -1,  1, -1, -1, -1,  1, -1,  1,  1, -1,  1,  1, -1, -1, -1, -1,  1,
        1, -1,  1, -1,  1, -1, -1, -1, -1, -1,  1, -1,  1,  1,  1, -1, -1,
        1,  1,  1,  1, -1,  1, -1, -1, -1,  1,  1,  1,  1, -1,  1,  1, -1,
       -1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1, -1, -1,
        1,  1,  1, -1,  1,  1,  1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,
       -1, -1,  1])
```

[11]:
```
y.shape
```

[11]:
```
(292,)
```

Let's also do a scatter plot of our data:

[12]:
```
points_colorscale = [
                     [0.0, 'rgb(239, 85, 59)'],
                     [1.0, 'rgb(99, 110, 250)'],
                    ]

points = go.Scatter(
                    x=df['Feature 1'],
                    y=df['Feature 2'],
                    mode='markers',
                    marker=dict(color=y,
                                colorscale=points_colorscale)
                   )
layout = go.Layout(
                   xaxis=dict(range=[-1.05, 1.05]),
                   yaxis=dict(range=[-1.05, 1.05])
                  )

fig = go.Figure(data=[points], layout=layout)
fig.show()
```

1.4 Splitting data

Let's split our data into training, validation and test sets. Let's use 60% for training, 20% for validation and 20% for test data.

[13]:
```
(X_train, X_vt, y_train, y_vt) = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=0)
(X_validation, X_test, y_validation, y_test) = train_test_split(X_vt, y_vt,␣
 ↪test_size=0.5, random_state=0)
```

1.5 Building and visualizing a SVM

Finally, let's build our SVM. We will use the `SVC` class from scikit-learn. For now, we are not using kernels, so we should set the `kernel` argument of `SVC` to `'linear'`. We don't need to specify any other parameters for now. You can find the documentation for `SVC` here:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
So, make an object of class `SVC` and assign it to the name `svm`:

[14]:
```
### begin your code here (1 line).
svm = SVC(kernel='linear')
### end your code here.
```

Now, fit `svm` to `X_train` and `y_train`:

[15]:
```
### begin your code here (1 line).
svm.fit(X_train, y_train)
### end your code here.
```

[15]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

You will get a summary for the model:

> SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

- You may also get a warning because you have not explicitly set gamma and the default setting for that is going to change in newer versions of scikit-learn. Don't worry about that warniong.

Let's visualize the decision surface our `svm` with its supprt vectors:

[16]:
```
decision_colorscale = [
                        [0.0, 'rgb(239,  85,  59)'],
                        [0.5, 'rgb(  0,   0,   0)'],
                        [1.0, 'rgb( 99, 110, 250)']
                      ]

detail_steps = 100

(x_vis_0_min, x_vis_1_min) = (-1.05, -1.05) #X_train.min(axis=0)
(x_vis_0_max, x_vis_1_max) = ( 1.05,  1.05) #X_train.max(axis=0)

x_vis_0_range = np.linspace(x_vis_0_min, x_vis_0_max, detail_steps)
x_vis_1_range = np.linspace(x_vis_1_min, x_vis_1_max, detail_steps)

(XX_vis_0, XX_vis_1) = np.meshgrid(x_vis_0_range, x_vis_0_range)

X_vis = np.c_[XX_vis_0.reshape(-1), XX_vis_1.reshape(-1)]
```

```
YY_vis = svm.decision_function(X_vis).reshape(XX_vis_0.shape)

points = go.Scatter(
                    x=df['Feature 1'],
                    y=df['Feature 2'],
                    mode='markers',
                    marker=dict(
                                color=y,
                                colorscale=points_colorscale),
                    showlegend=False
                   )
SVs = svm.support_vectors_
support_vectors = go.Scatter(
                             x=SVs[:, 0],
                             y=SVs[:, 1],
                             mode='markers',
                             marker=dict(
                                         size=15,
                                         color='black',
                                         opacity = 0.1,
                                         colorscale=points_colorscale),
                             line=dict(dash='solid'),
                             showlegend=False
                            )

decision_surface = go.Contour(x=x_vis_0_range,
                              y=x_vis_1_range,
                              z=YY_vis,
                              contours_coloring='lines',
                              line_width=2,
                              contours=dict(
                                            start=0,
                                            end=0,
                                            size=1),
                              colorscale=decision_colorscale,
                              showscale=False
                             )

margins = go.Contour(x=x_vis_0_range,
                     y=x_vis_1_range,
                     z=YY_vis,
                     contours_coloring='lines',
                     line_width=2,
                     contours=dict(
                                   start=-1,
                                   end=1,
                                   size=2),
                     line=dict(dash='dash'),
                     colorscale=decision_colorscale,
                     showscale=False
                    )

fig2 = go.Figure(data=[margins, decision_surface, support_vectors, points],␣
 ↪layout=layout)
fig2.show()
```

The datapoints, the decision surface (which is a line here), the margins and the support vectors are shown in the plot.

1.6 Kernels

As you can see, the decision surface is underfiiting the data. Let's use a polynomial kernel. Define `svm_p2` to be an instance of class `SVC` but this time with arguments `kernel='poly'` and `degree=2` to define a degree-2 polynomial kernel:

[17]:
```
### begin your code here (1 line).
svm_p2 = SVC(kernel='poly', degree=2)
### end your code here.
```

..and fit it to your training data:

[18]:
```
### begin your code here (1 line).
svm_p2.fit(X_train, y_train)
### end your code here.
```

[18]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=2, gamma='auto_deprecated',
  kernel='poly', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

You will get a summary of your model:

> SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=2, gamma='auto_deprecated', kernel='poly', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

Now, let's visualize this model:

[19]:
```
YY_vis_p2 = svm_p2.decision_function(X_vis).reshape(XX_vis_0.shape)

SVs_p2 = svm_p2.support_vectors_
support_vectors_p2 = go.Scatter(
                                x=SVs_p2[:, 0],
                                y=SVs_p2[:, 1],
                                mode='markers',
                                marker=dict(
                                            size=15,
                                            color='black',
                                                   opacity = 0.1,
                                                   colorscale=points_colorscale),
                                      line=dict(dash='solid'),
                                      showlegend=False
                                     )

decision_surface_p2 = go.Contour(x=x_vis_0_range,
                                 y=x_vis_1_range,
                                 z=YY_vis_p2,
                                 contours_coloring='lines',
                                 line_width=2,
                                 contours=dict(
                                              start=0,
                                              end=0,
                                              size=1),
                                 colorscale=decision_colorscale,
                                 showscale=False
                                )

margins_p2 = go.Contour(x=x_vis_0_range,
                        y=x_vis_1_range,
                        z=YY_vis_p2,
                        contours_coloring='lines',
                        line_width=2,
                        contours=dict(
                                     start=-1,
                                     end=1,
                                     size=2),
                        line=dict(dash='dash'),
                        colorscale=decision_colorscale,
                        showscale=False
                       )

fig3 = go.Figure(data=[margins_p2, decision_surface_p2, support_vectors_p2,␣
 ↪points], layout=layout)
fig3.show()
```

Looks much better. But let's try a degree 3 model. Define svm_p3 like svm_p2 but with degree=3 this time:

[20]:
```
### begin your code here (1 line).
svm_p3 = SVC(kernel='poly', degree=3)
### end your code here.
```

Next, fit your svm_p3 model to the training data:

[21]:
```
### begin your code here (1 line).
svm_p3.fit(X_train, y_train)
### end your code here.
```

[21]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='poly', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

Your model summary will be:

> SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='poly', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

Let's visualize svm_p3:

[22]:
```
YY_vis_p3 = svm_p3.decision_function(X_vis).reshape(XX_vis_0.shape)

SVs_p3 = svm_p3.support_vectors_
support_vectors_p3 = go.Scatter(
                                x=SVs_p3[:, 0],
                                y=SVs_p3[:, 1],
                                mode='markers',
                                marker=dict(
                                            size=15,
                                            color='black',
                                            opacity = 0.1,
                                            colorscale=points_colorscale),
                                line=dict(dash='solid'),
                                showlegend=False
                               )

decision_surface_p3 = go.Contour(x=x_vis_0_range,
                                 y=x_vis_1_range,
                                 z=YY_vis_p3,
                                 contours_coloring='lines',
                                 line_width=2,
                                 contours=dict(
                                               start=0,
                                               end=0,
                                               size=1),
                                 colorscale=decision_colorscale,
                                 showscale=False
                                )

margins_p3 = go.Contour(x=x_vis_0_range,
                        y=x_vis_1_range,
                        z=YY_vis_p3,
                        contours_coloring='lines',
                        line_width=2,
                        contours=dict(
                                              start=-1,
                                              end=1,
                                              size=2),
                                line=dict(dash='dash'),
                                colorscale=decision_colorscale,
                                showscale=False
                               )

fig4 = go.Figure(data=[margins_p3, decision_surface_p3, support_vectors_p3,␣
 ↪points], layout=layout)
fig4.show()
```

Let's try a RBF (Radial Basis Function) kernel as well. RBFs are the default kernel for scikit-learn's SVC. So build a model `svm_r` with either `kernel=rbf` argument setting or just skip the `kernel` (also the `degree` argument is uselss here, since we are not using a polynomial kernel, so just skip that):

[23]:
```
### begin your code here (1 line).
svm_r = SVC(kernel='rbf')
### end your code here.
```

Fit your `svm_r` model to the training data as well:

[24]:
```
### begin your code here (1 line).
svm_r.fit(X_train, y_train)
### end your code here.
```

[24]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

This will be the parameter summary:

> SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

We will visualize this model as well:

[25]:
```
YY_vis_r = svm_r.decision_function(X_vis).reshape(XX_vis_0.shape)

SVs_r = svm_r.support_vectors_
support_vectors_r = go.Scatter(
                                x=SVs_r[:, 0],
                                y=SVs_r[:, 1],
                                mode='markers',
                                marker=dict(
                                            size=15,
                                            color='black',
                                            opacity = 0.1,
                                            colorscale=points_colorscale),
                                line=dict(dash='solid'),
                                showlegend=False
                               )

decision_surface_r = go.Contour(x=x_vis_0_range,
                                 y=x_vis_1_range,
                                 z=YY_vis_r,
                                 contours_coloring='lines',
                                 line_width=2,
                                 contours=dict(
                                               start=0,
                                               end=0,
                                               size=1),
                                 colorscale=decision_colorscale,
                                 showscale=False
                                )

margins_r = go.Contour(x=x_vis_0_range,
                        y=x_vis_1_range,
                        z=YY_vis_r,
                        contours_coloring='lines',
                        line_width=2,
                        contours=dict(
                                      start=-1,
                                      end=1,
                                      size=2),
                        line=dict(dash='dash'),
                        colorscale=decision_colorscale,
                        showscale=False
                       )

fig5 = go.Figure(data=[margins_r, decision_surface_r, support_vectors_r,␣
 ↪points], layout=layout)
fig5.show()
```

1.7 Model selection

Let's pick the best model then. We will use the validation data for that. Let's predict using `svm` and `X_train` and assign it the name `yhat_train`. Also, predict `X_validation` and assign it the name `yhat_validation` (. The closeness of the accuracy of predictions on these two datasets will be helpful to us):

[28]:
```
### begin your code here (2 lines).
yhat_train = svm.predict(X_train)
yhat_validation = svm.predict(X_validation)
### end your code here.
```

Let's measure the accuracy:

[29]:
```
print(accuracy_score(yhat_train, y_train), accuracy_score(yhat_validation,␣
 ↪y_validation))
```

```
0.9142857142857143 0.9482758620689655
```

We got 91.42% and 94.83%.

Let's repeat thge predictions for `svm_p2` and put the results in `yhat_train_p2` and `yhat_validation_p2`:

[30]:
```
### begin your code here (2 lines).
yhat_train_p2 = svm_p2.predict(X_train)
yhat_validation_p2 = svm_p2.predict(X_validation)
### end your code here.
```

We can calculate the accuracies:

[31]:
```
print(accuracy_score(yhat_train_p2, y_train),␣
 ↪accuracy_score(yhat_validation_p2, y_validation))
```

```
0.5371428571428571 0.46551724137931033
```

We got 53.71% and 46.55%.

Let's try predicting with `svm_p3` and put it in `yhat_train_p3` and `yhat_validation_p3`:

[32]:
```
### begin your code here (2 lines).
yhat_train_p3 = svm_p3.predict(X_train)
yhat_validation_p3 = svm_p3.predict(X_validation)
### end your code here.
```

Now, if we predict the accuracy on these:

[33]:
```
print(accuracy_score(yhat_train_p3, y_train),␣
 ↪accuracy_score(yhat_validation_p3, y_validation))
```

```
0.5828571428571429 0.5172413793103449
```

The accuracy is 58.28% and 51.72%.

Finally, let's predict `yhat_train_r` and `yhat_validatin_r` using `svm_r`:

[34]:
```
### begin your code here (2 lines).
yhat_train_r = svm_r.predict(X_train)
yhat_validation_r = svm_r.predict(X_validation)
### end your code here.
```

We can measure the accuracy of the SVM with RBF kernel:

[35]:
```
print(accuracy_score(yhat_train_r, y_train), accuracy_score(yhat_validation_r,␣
 ↪y_validation))
```

```
0.9314285714285714 0.9827586206896551
```

We got 93.14% and 98.27% when using the RBF kernel.

From all these number we can see that the RBF model works best as the accuracy on validation data is high and also the gap between the accuracy on training and validation data is not big. We can further tune the generalization power of our model by tuning the argument `C` of `SVC` which is the inverse of a regularization coefficient. We won't do that here now though.

1.8 Final assessment

Finally, let's check accuracy on the test data to get a final performance number. Predict `yhat_test_r` from `X_test` on `svm_r`:

[37]:
```
### begin your code here (1 line).
yhat_test_r = svm_r.predict(X_test)
### end your code here.
accuracy_score(yhat_test_r, y_test)
```

[37]:
```
0.9491525423728814
```

We got 94.91%. We have good performance of test data.