mlp-week08

March 16, 2021

1 Machine Learning in Python - Workshop 8

2 1. Setup

2.1 1.1 Packages

In the cell below we will load the core libraries we will be using for this workshop and setting some sensible defaults for our plot size and resolution.

[1]:
```
# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,8)
plt.rcParams['figure.dpi'] = 80
plt.rcParams['lines.markersize'] = 7.5

# sklearn modules
import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, KFold
```

2.2 1.2 Helper Functions

[2]:
```
def plot_margin(model, data, x='x', y='y', cat='z', show_support_vectors =␣
↪True, nx=50, ny=50):
    # Plot the data
    p = sns.scatterplot(x=x, y=y, hue=cat, data=data, legend=False)

    # Find the extent of x and y
    xlim = p.get_xlim()
    ylim = p.get_ylim()

    # Create a grid of points
    xx = np.linspace(xlim[0], xlim[1], nx)
    yy = np.linspace(ylim[0], ylim[1], ny)
    YY, XX = np.meshgrid(yy, xx)

    # Calculate the label for each point in the grid
    xy = np.c_[XX.ravel(), YY.ravel()]
    Z = model.decision_function(xy).reshape(XX.shape)

    # plot contours of decision boundary and margins
    p.contour(XX, YY, Z, colors='k',
              levels=[-1, 0, 1], alpha=0.5,
              linestyles=['--', '-', '--'])

    # highlight support vectors
    if (show_support_vectors):
        p.scatter(model.support_vectors_[:, 0],
                  model.support_vectors_[:, 1], s=100,
                  linewidth=1, facecolors='none', edgecolors='k')

    # Show confusion table in the title
    p.set_title(
        "TN: {0}, FP: {1}, FN: {2}, TP: {3}".format(
            *confusion_matrix(
                data[cat],
                m.predict(data.drop(cat, axis=1))
            ).flatten()
        )
    )

    plt.show()
```

2.3 1.3 Jupyter Advice

Some of the interactive plots below are large enough that Jupyter will attempt to add a scrollbar which will prevent you from seeing all the options and the plot at the same time. To avoid this you can select Cell -> Current Outputs -> Toggle Scrolling from the Jupyter menu above to remove this scrolling.

---

3 2. Support Vector Machine

In this section we will be exploring the basics of support vector machine models. SVMs have their own entire submodule of sklearn which includes more than we will be able to cover in this workshop. We will be focusing on the most straight forward case, which is a support vector machine classifier which is provide by sklearn as the `SVC` model.

[3]:
```
from sklearn.svm import SVC
```

3.1 2.1 Example 1 - Separable data

We will begin by examining several toy data problems to explore the basics of these models. To begin we will read in data for the first example from `ex1.csv`.

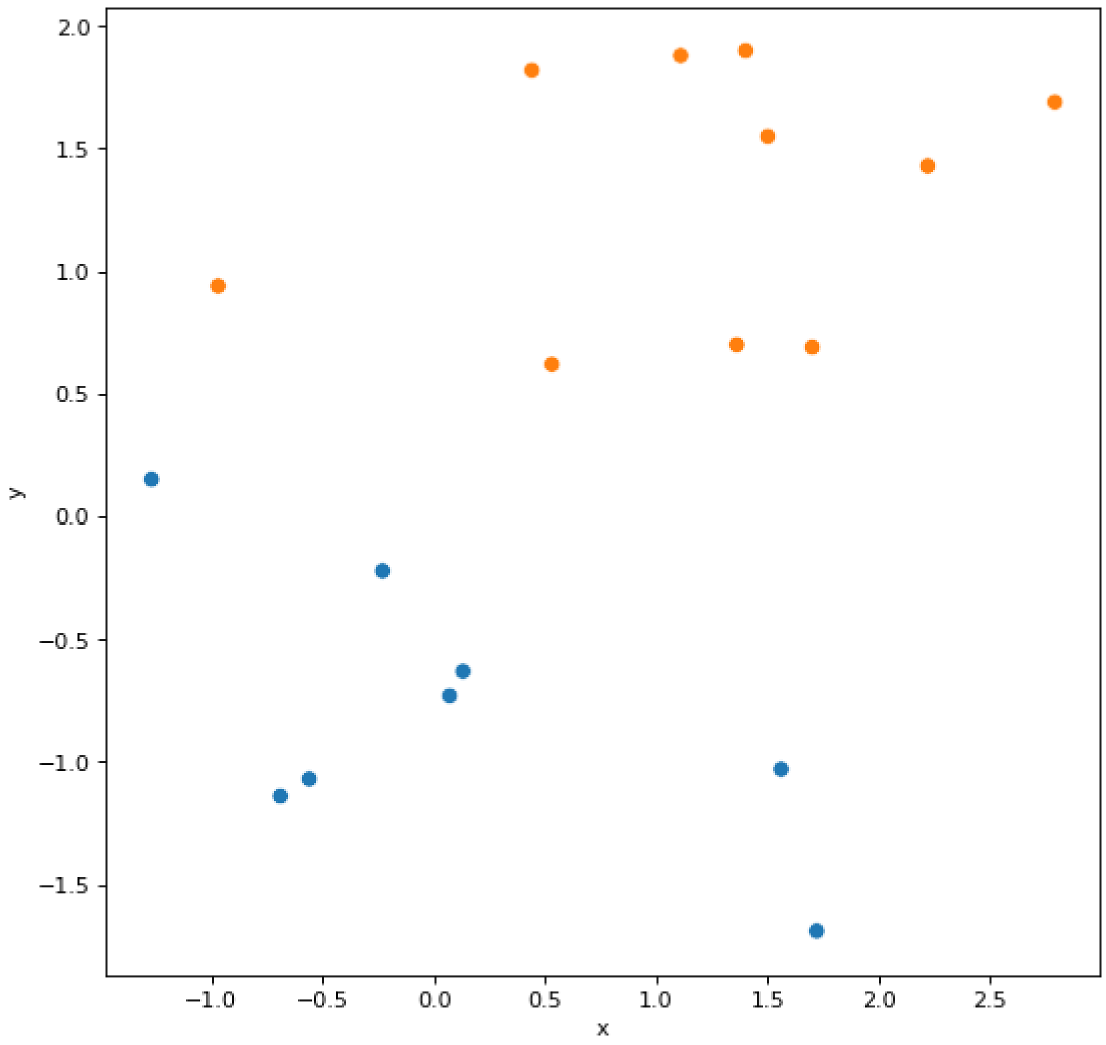
[4]:
```
ex1 = pd.read_csv("ex1.csv")
ex1
```

[4]:
```
       x     y  z
0  -0.56 -1.07  A
1  -0.23 -0.22  A
2   1.56 -1.03  A
3   0.07 -0.73  A
4   0.13 -0.63  A
5   1.72 -1.69  A
6  -1.27  0.15  A
7  -0.69 -1.14  A
8   2.22  1.43  B
9   1.36  0.70  B
10  1.40  1.90  B
11  1.11  1.88  B
12  0.44  1.82  B
13  2.79  1.69  B
14  1.50  1.55  B
15 -0.97  0.94  B
16  1.70  0.69  B
```

```
17  0.53  0.62  B
```

We can see the that data is composed of two classes in two dimensions, and it is clear that these two classes are perfectly linearly separable - i.e. we can draw a straight line that divides the classes.

[5]:
```
sns.scatterplot(x='x', y='y', hue='z', data=ex1, legend=False)
plt.show()
```



With a linear SVM model our goal is to identify the line (or a separating hyperplane in higher dimensions) with the largest possible margin. Like the other models we've already seen, we fit the SVM by constructing our feature matrix and outcome vector and then calling the `fit` method for our model object.

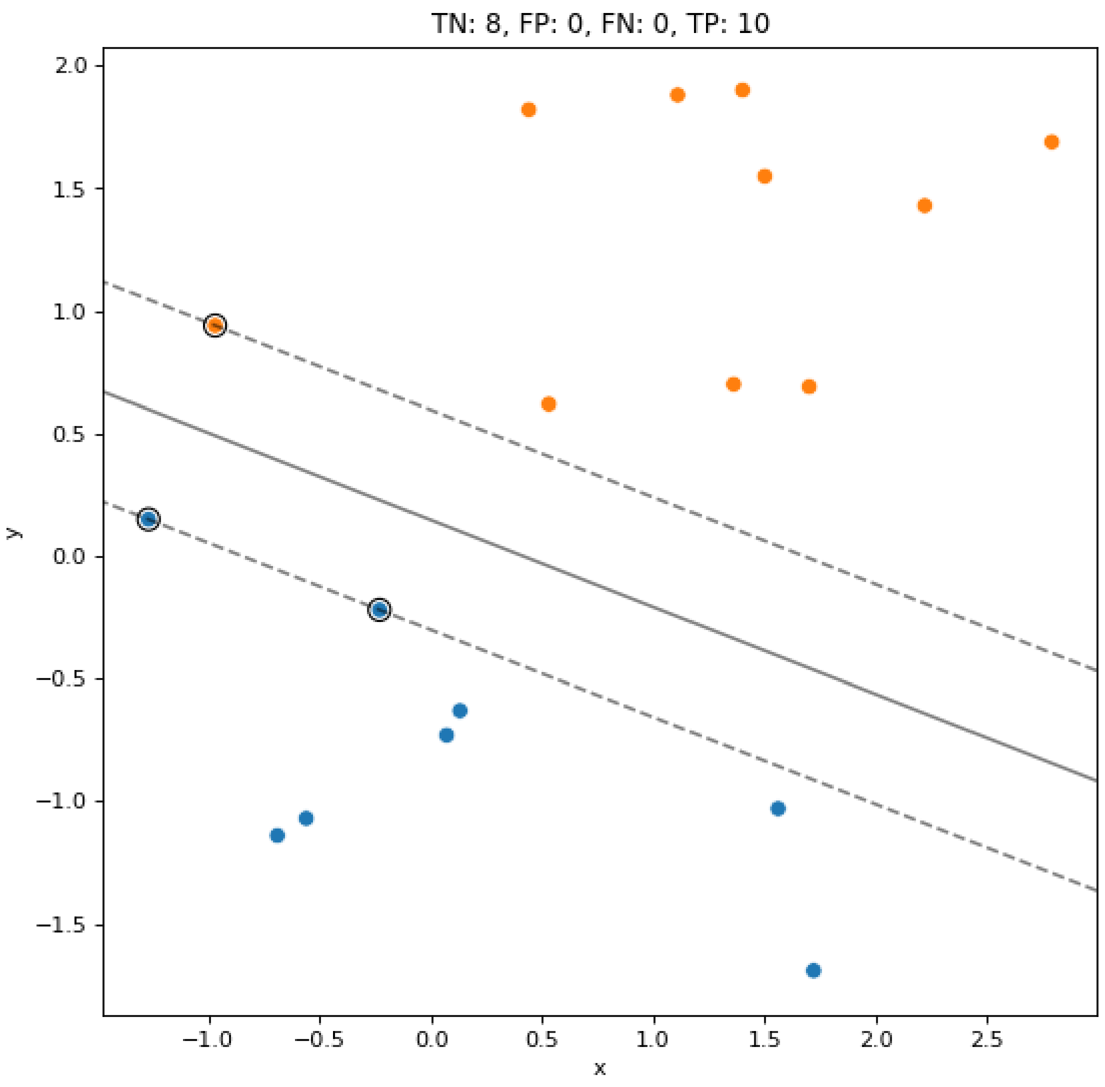
[6]:
```
X_ex1 = ex1.drop('z', axis=1)
y_ex1 = ex1.z
```

When fitting the model we will only worry about two parameters for now: `kernel` which determines what if any kernel is used when calculating the dot product, and `C` which is the penalty for misclassification. These parameters default to `kernel='rbf'` for a radial basis function and `C=1`, we will change these to a `linear` kernel and a arbitrary large penalty respectively.

[7]:
```
m = SVC(kernel='linear', C=10).fit(X_ex1, y_ex1)
```

Once fit we can visualize the decision boundary and the margins using the `plot_margin` function we defined above.

[8]:
```
plot_margin(m, ex1)
```



The solid line is the separating hyperplane (line) and the dashed lines are the margins.

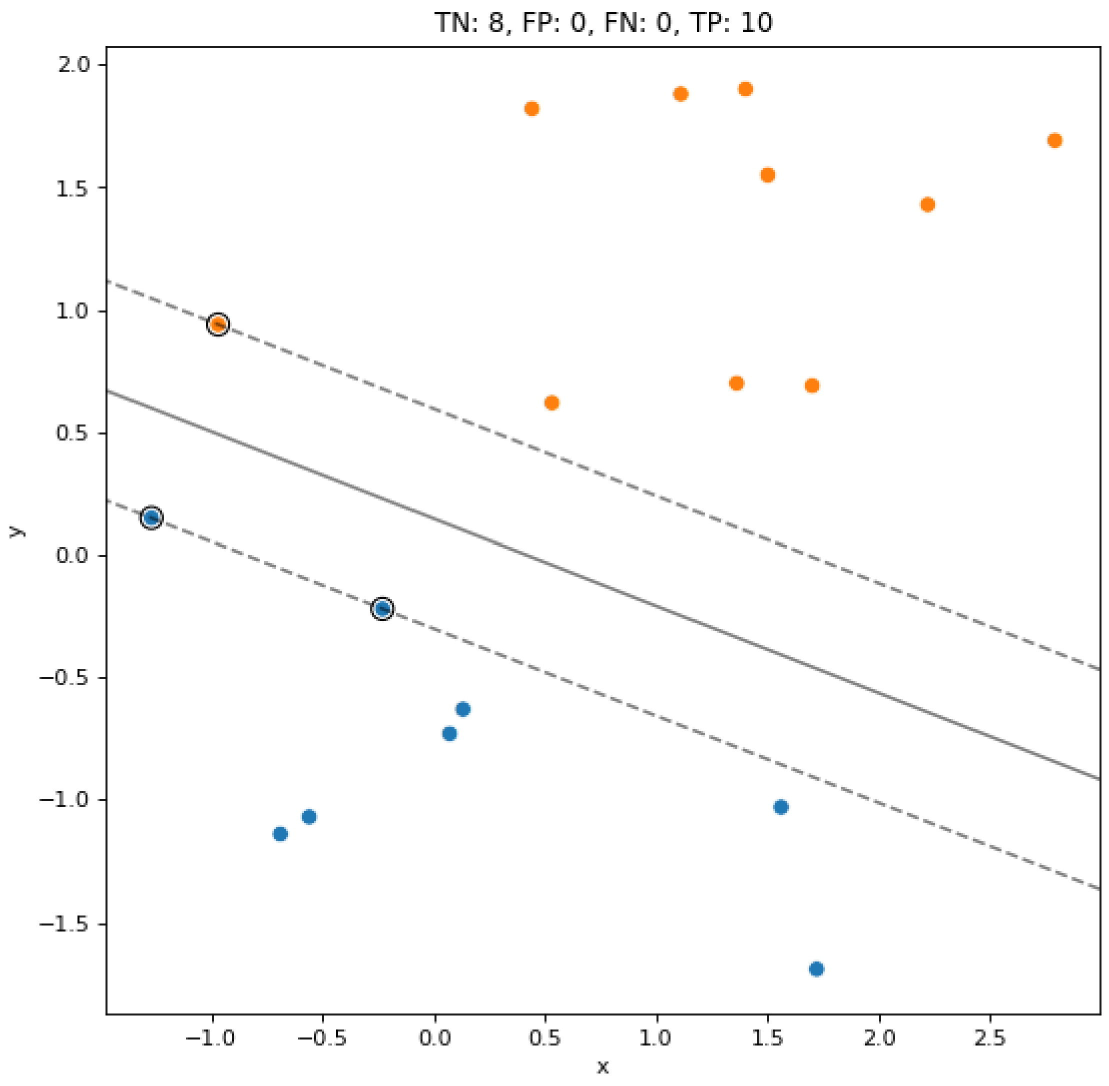
3.1.1 Exercise 1

How many support vectors are there for this model?

There are three support vectors in this model, the two blue points at approximately (-0.25,-0.25) and (-1.5,0.25), and the orange point at approximately (-1.25,0.9).

We can explore how the penalty value affects this fit by setting up a simple function and interactive widget.

[9]:
```
C = 10
m = SVC(kernel='linear', C=C).fit(X_ex1, y_ex1)
plot_margin(m, ex1)
```



3.1.2 Exercise 2

How does the boundary line and the margins change as you change the value of `C`?

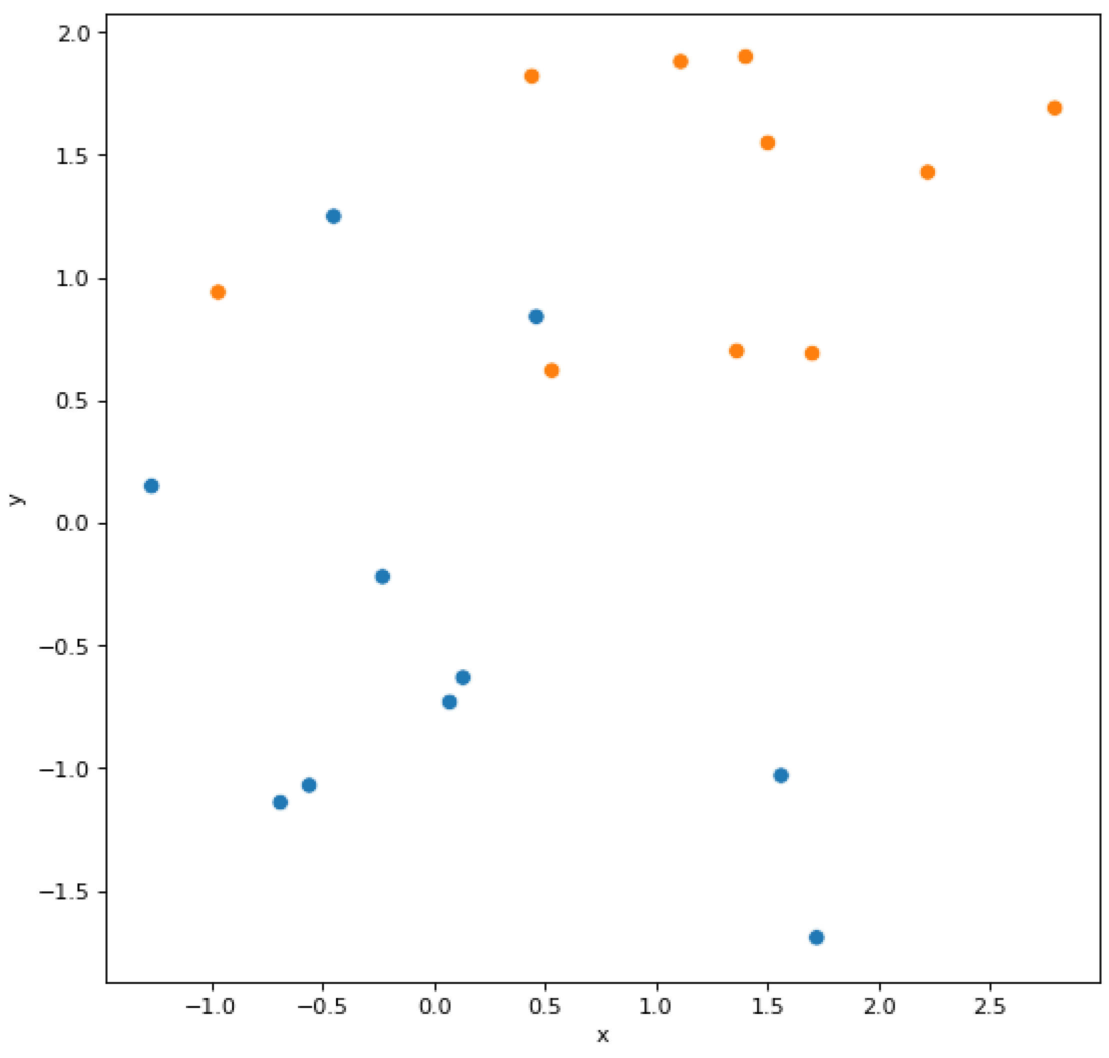
As we increase C, the margin becomes smaller and our model allows more points to lie inside the street. For small values of C, the bias is large but the variance is small.

3.2 2.2 Example 2 - Non-separable data

We will not complicate our previous example somewhat by adding two additional points from the blue `A` class to our data. This is available in the `ex2.csv` file.

[10]:
```
ex2 = pd.read_csv("ex2.csv")
```

[11]:
```
sns.scatterplot(x='x', y='y', hue='z', data=ex2, legend=False)
plt.show()
```



3.2.1 Exercise 3

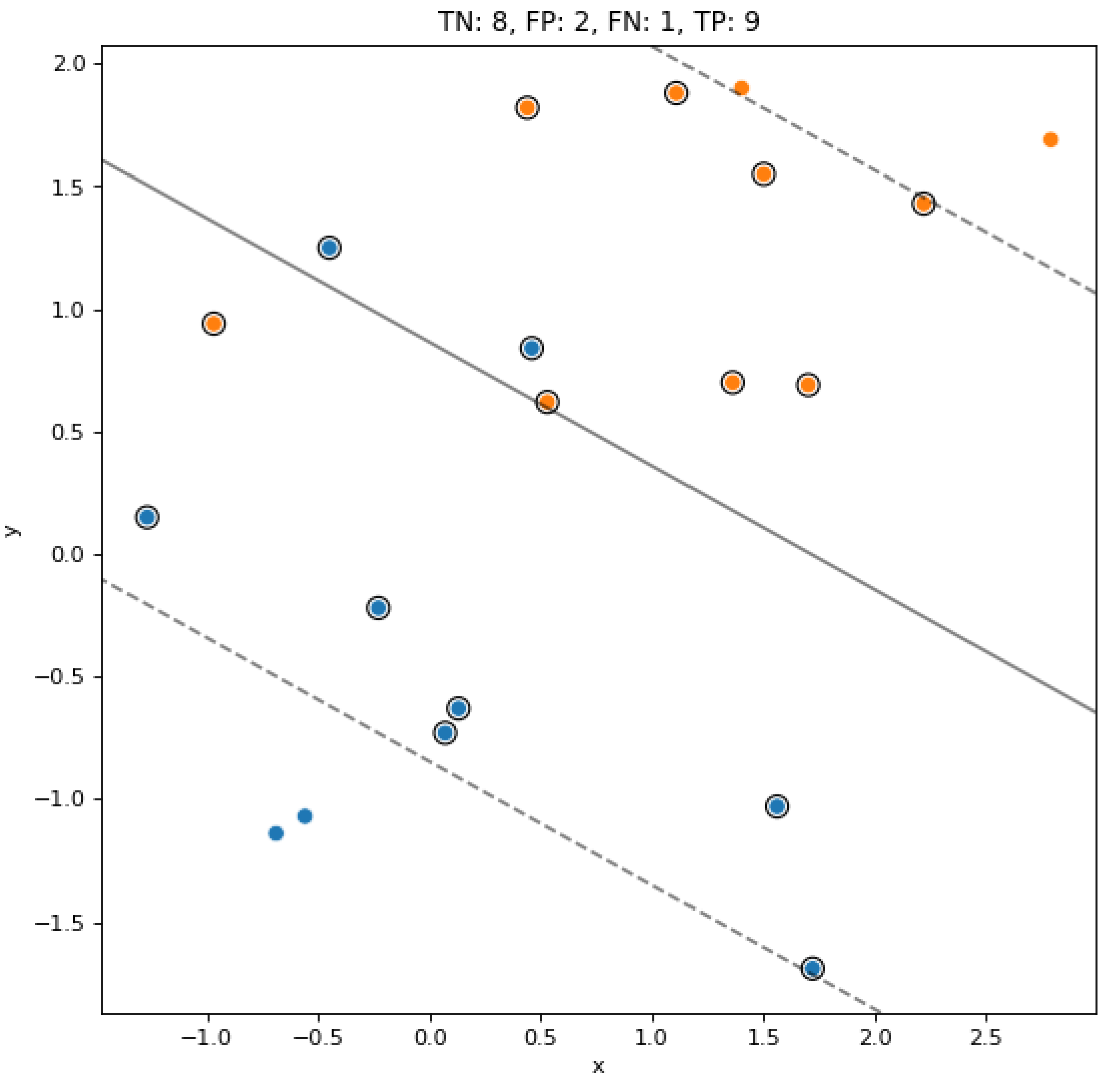
Are these two classes linearly separable?

These classes are not linearly seperable, we require a non-linear decision boundary to separate the classes.

We can again fit a `SVC` model to these data using the same code we used with example 1.

[12]:
```
X_ex2 = ex2.drop('z', axis=1)
y_ex2 = ex2.z
```

[13]:
```
C = 0.05
m = SVC(kernel='linear', C=C).fit(X_ex2, y_ex2)
plot_margin(m, ex2)
```



3.2.2 Exercise 4

How does the "fit" of this model differ compared to the "fit" for example 1. *Hint* - make your comparison for equivalent values of `C`.

When looking at a value of `C=10` for both models, the margin and correspondingly the number of support vectors is larger in this model. This gives a fit that has high bias and lower variance for the same value of C.

3.2.3 Exercise 5

How do the boundary line and margins change as you change the value of `C`?

As you increase the value of C, the margin becomes smaller, however we require much larger values of C for the margin to become small as in the previous examples. For example a value of C=1000000 leaves the margin relatively large compared to a similar value in exercise 1.

Small values of C increase the size of the boundary as in the previous examples.

3.3 2.3 Example 3 - A non-linear kernel

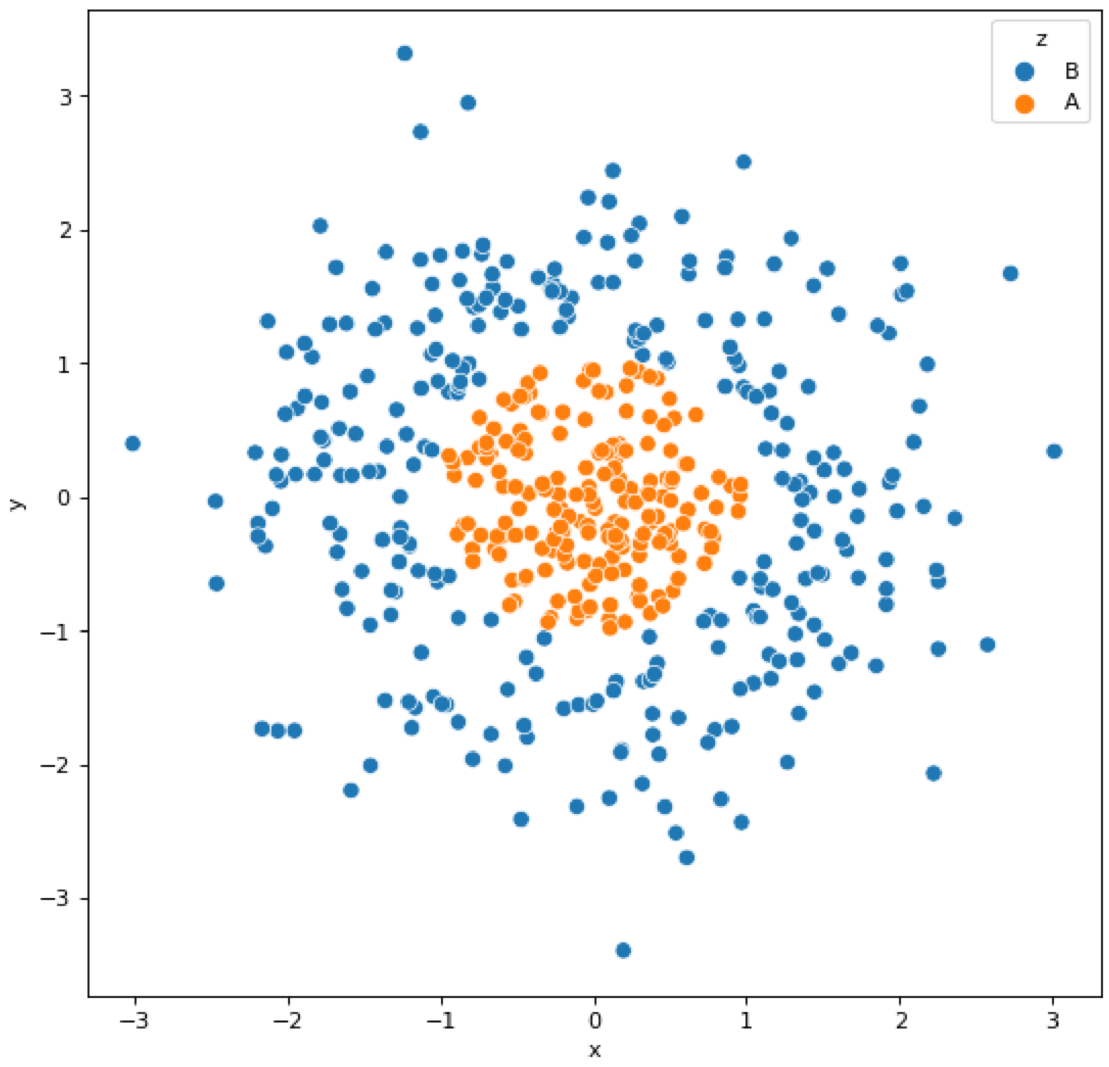
Next we will look at a new data set that would seem to also fall in the non-separable category.

[14]:
```
ex3 = pd.read_csv("ex3.csv")
X_ex3 = ex3.drop('z', axis=1)
y_ex3 = ex3.z
ex3
```

[14]:
```
          x      y  z
0   -0.577 -2.012  B
1    0.355  0.395  A
2    0.983  2.502  B
3   -1.059  1.063  B
4   -0.243 -0.271  A
..     …      … ..
470  0.022 -1.527  B
471 -0.943  0.306  A
472 -1.782  0.445  B
473  0.820  0.146  A
474 -1.269 -0.488  B

[475 rows x 3 columns]
```

[15]:
```
sns.scatterplot(x='x', y='y', hue='z', data=ex3, legend=True)
plt.show()
```



Clearly for these data there is no possible separating line that could come close to classifying these points. However, due to the kernel trick it is possible (and efficient) to project our data into a higher dimensional space where it may be possible to then define a separating hyperplane. For these data we will consider a simple polynomial kernel with degree 2. This kernel is defined as

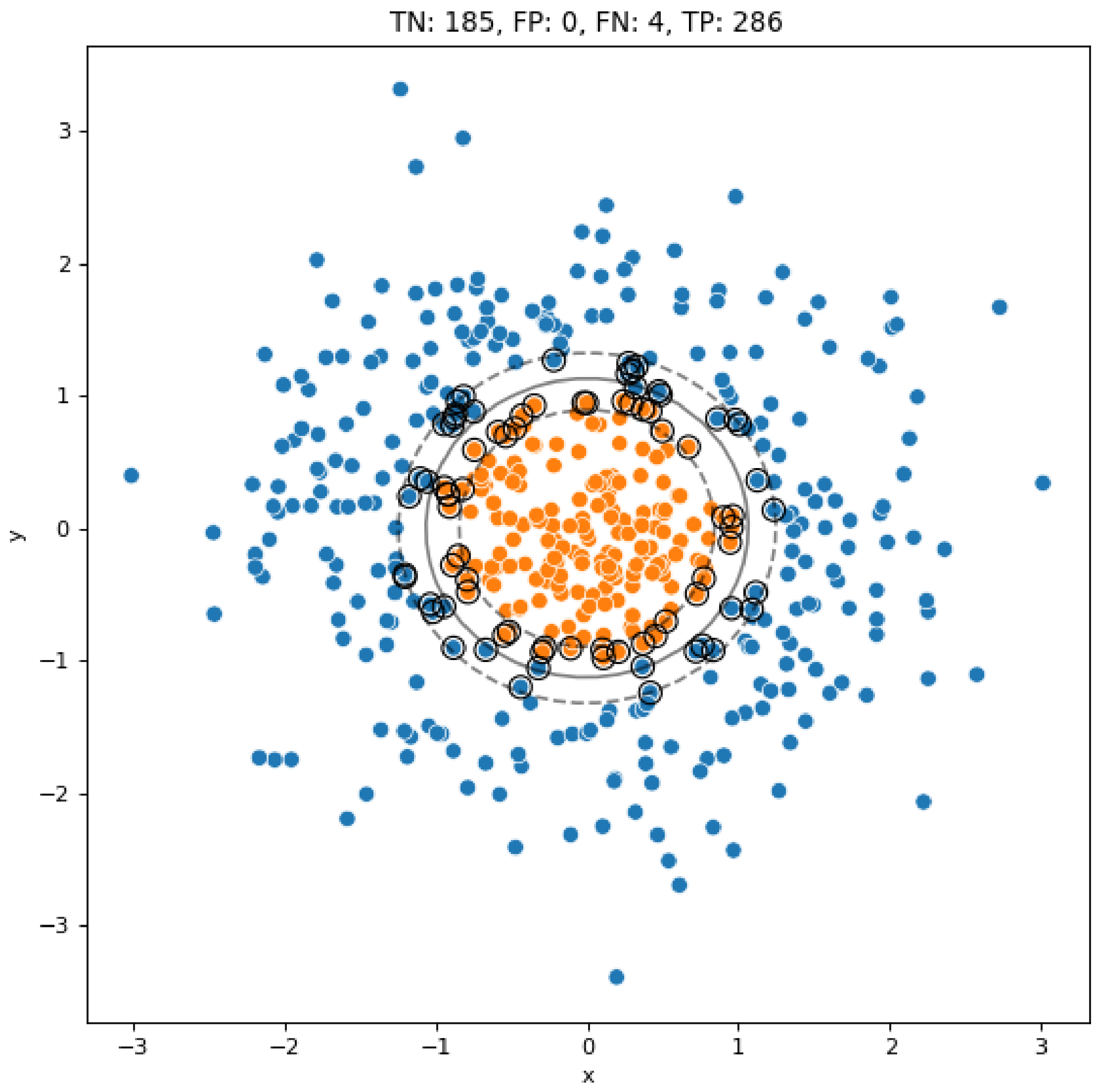
$$K(x, x') = (\gamma\langle x, x'\rangle + r)^d$$

where γ is a scaling parameter, d is the degree, and r related to the intercept and is called `coef0` by sklearn. More details on the various kernels that can be used with the `SVC` model are available here.

The effect of using such a kernel is immediately obvious, as you can see in the interactive figure below.

[31]:
```
C = 1          # [1,2,5,7,10,20,50,100]
kernel = 'poly' # ['poly', 'linear']
```

```
m = SVC(kernel=kernel, degree=2, C=C, gamma='scale').fit(X_ex3, y_ex3)
plot_margin(m, ex3)
```



3.3.1 Exercise 6

Compare the fit of the model using the polynomial and linear kernel. Describe the shape of the boundaries and the margins.

The linear model is much worse, we would expect this as a non-linear decision boundary here is clearly prefered. We can see that the polynomial model has 290 True Positives and 185 True

Negatives and is perfectly predicting all the A's and B's in the training set. It is using a nonlinear decision boundary and nonlinear margin to fit the training data

While the linear model predicts all of the data points to be B as the hyperplane is not displayed on the graph and we can only see the margin. Giving 290 True Positives and 185 False Positives.

3.3.2 Exercise 7

How do the boundary line and margins change as you change the value of `C`?

Different values of C for the linear model make no difference to the predictions. Smaller values of C for the polynomial model lead to a larger margin with more support vectors, while a larger value of C leads to a smaller margin therefore lower bias and higher variance.

To better understand what is happening when we use a kernel with a SVM we can think about the kernel as projecting our data into a higher dimensional space - with this particular kernel we can think of it as adding a third dimension, proportional to $x^2 + y^2$ to our data and then asking for the SVM model to find a separating *plane* in this 3-d space. We can crudely visualize this by creating this new dimension and plotting our data as a 3d scatter plot.

[32]:
```
def plot_proj(elev=10, azim=45):
    colors = {'A':'#D37042', 'B':'#3D5DA1'}

    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.view_init(elev=elev, azim=azim)
    ax.scatter(ex3.x, ex3.y, (ex3.x**2 + ex3.y**2), c=ex3.z.apply(lambda x:␣
 ↪colors[x]))

    ax.set_xlabel('x')
    ax.set_ylabel('y ')
    ax.set_zlabel('z ')

    plt.show()
```

You can adjust the values of `elev` and `azim` to adjust the "camera" position for the plot to get a better view of

[37]:
```
%matplotlib notebook
plot_proj(elev = 10, azim = 45)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.3.3 Exercise 8

As you adjust your view of the data via the elevation and azimuth, imagine a plane that could be used to separate these data into the two classes. If we were to project that plane back down to just `x` and `y`, what would its shape be?

The intersection of the separating plane and the surface would be a circle when projected down to the x-y axis. However, the whole plane projected down would span the whole xy axis.

3.4 2.4 Example 4 - Other Kernels

Next we will consider an even more complicated separation task where one class is split into two separate clusters by the second class. The data ara available as `ex4.csv`.

[48]:
```
ex4 = pd.read_csv("ex4.csv")
X_ex4 = ex4.drop('z', axis=1)
y_ex4 = ex4.z
ex4
```

[48]:
```
        x     y  z
0    0.90 -1.21  A
1   -1.65 -1.16  B
2    1.16 -1.35  A
3    0.76  0.39  A
4    1.36 -1.41  A
..    ...   ... ..
395 -1.13  0.77  A
396 -1.12 -1.86  B
397 -2.01 -1.01  B
398 -1.97 -2.43  B
399 -0.44  0.21  A

[400 rows x 3 columns]
```

[49]:
```
plt.figure(figsize=(8, 8))
sns.scatterplot(x='x', y='y', hue='z', data=ex4, legend=True)
plt.show()
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

Below we set up a similar interactive tool for experimenting with different penalties and kernel functions for these data. Note that the degree value is only used by polynomial kernel and is ignore by the linear and rbf kernels.

[71]:
```python
C = 1        # [1,5,10,50,100],
degree = 4   # [2,3,4],
kernel = 'rbf' # ['poly', 'rbf', 'linear'])

for c in [1,5,10,50,100]:
    for deg in [2,3,4]:
        for k in ['poly', 'rbf', 'linear']:
            m = SVC(kernel=k, degree=deg, C=c, gamma='scale').fit(X_ex4, y_ex4)
            print("TN: {0}, FP: {1}, FN: {2}, TP: {3}".format(
                        *confusion_matrix(
                            ex4['z'],
                            m.predict(ex4.drop('z', axis=1))
                        ).flatten()))
```

```
TN: 192, FP: 8, FN: 19, TP: 181
TN: 192, FP: 8, FN: 14, TP: 186
TN: 168, FP: 32, FN: 102, TP: 98
TN: 10, FP: 190, FN: 1, TP: 199
TN: 192, FP: 8, FN: 14, TP: 186
TN: 168, FP: 32, FN: 102, TP: 98
TN: 197, FP: 3, FN: 23, TP: 177
TN: 192, FP: 8, FN: 14, TP: 186
TN: 168, FP: 32, FN: 102, TP: 98
TN: 191, FP: 9, FN: 13, TP: 187
TN: 192, FP: 8, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 10, FP: 190, FN: 1, TP: 199
TN: 192, FP: 8, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 193, FP: 7, FN: 18, TP: 182
TN: 192, FP: 8, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 191, FP: 9, FN: 13, TP: 187
TN: 191, FP: 9, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 10, FP: 190, FN: 1, TP: 199
TN: 191, FP: 9, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 193, FP: 7, FN: 17, TP: 183
TN: 191, FP: 9, FN: 12, TP: 188
TN: 168, FP: 32, FN: 102, TP: 98
TN: 191, FP: 9, FN: 13, TP: 187
TN: 190, FP: 10, FN: 9, TP: 191
TN: 168, FP: 32, FN: 102, TP: 98
```

```
TN: 10, FP: 190, FN: 1, TP: 199
TN: 190, FP: 10, FN: 9, TP: 191
TN: 168, FP: 32, FN: 102, TP: 98
TN: 193, FP: 7, FN: 16, TP: 184
TN: 190, FP: 10, FN: 9, TP: 191
TN: 168, FP: 32, FN: 102, TP: 98
TN: 191, FP: 9, FN: 13, TP: 187
TN: 189, FP: 11, FN: 10, TP: 190
TN: 168, FP: 32, FN: 102, TP: 98
TN: 10, FP: 190, FN: 1, TP: 199
TN: 189, FP: 11, FN: 10, TP: 190
TN: 168, FP: 32, FN: 102, TP: 98
TN: 193, FP: 7, FN: 14, TP: 186
TN: 189, FP: 11, FN: 10, TP: 190
TN: 168, FP: 32, FN: 102, TP: 98
```

[137]:
```
plt.figure(figsize=(8, 8))
m = SVC(kernel='poly', degree=4, C=100, gamma='scale').fit(X_ex4, y_ex4)
plot_margin(m, ex4)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.4.1 Exercise 9

What combination of parameters appears to produce the best fit? Is it easy to tell this by visual inspection alone?

Linear is clearly very poor for all values of C.

The polynomial model of degree 2 looks like it would generalise best to test data from the polynomialsn and has the best TP and TN of the polynomials. We would also, visually, prefer a lower value of C that results in a larger margin since there are some outliers in our data.

The rbf models appear to work well with the training data, however, they appear to be overfitting the data more than the polynomial models. We would prefer

In our For loop, we prefer `C=100`,`deg=4` and `kernel=poly` as the best model. It is hard to verify this only visually

---

3.4.2 Exercise 10

How do the support vectors change as the kernel, penalty, and degree are changed?

For larger values of C we have smaller margins therefore less support vectors.

A higher degree results in a less linear decision boundary and varies in the number of support vectors. Some degrees provide a better balance of over and underfitting.

The linear models result in a large number of support vectors and wide margins due to a nonlinear decision boundary being required to well fit the data. The polynomial and rbf models give a much smaller number of support vectors. While the rbf kernel fits the data tightly for high degrees and results in the outside data points being picked up as support vectors, unlike the other models.

3.5 2.5 Model Assessment

So far we have only inspected the various models by eye to get a sense of how well they fit our data. Since we are undertaking a classification task here we would like to be able to leverage the metrics and scoring tools we have already learned around logistic regression and related tools. The issue is that while we could generate a simple confusion matrix for our models' predictions this is somewhat limiting.

Let us consider the `rbf` model for example 4, we can fit this model and then explore what options we have.

[75]:
```
m = SVC(kernel='rbf', C=1, gamma='scale').fit(X_ex4, y_ex4)
```

From this model we can report the accuracy, which is defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

using the `score` method. Similarly, if we want the confusion matrix or any other metric that can be calculated using the appropriate function and the `predict` method.

[76]:
```
m.score(X_ex4, y_ex4)
```

[76]:
```
0.945
```

[77]:
```
sklearn.metrics.confusion_matrix(y_ex4, m.predict(X_ex4))
```

[77]:
```
array([[192,   8],
       [ 14, 186]])
```

By default, SVM models do not support the construction of anything like a ROC curve since the predictions are not probabilistic - i.e. labels are assigned based on which side of the separator a point falls. As such, `SVC` models do not implement `predict_proba` by default.

[78]:
```
m.predict_proba(X_ex4)
```

```
        ␣
↪---------------------------------------------------------------------------
```

```
        AttributeError                            Traceback (most recent call␣
↪last)

        <ipython-input-78-ed7b134d651b> in <module>
    ----> 1 m.predict_proba(X_ex4)


        /opt/conda/lib/python3.7/site-packages/sklearn/svm/_base.py in␣
↪predict_proba(self)
        655             datasets.
        656             """
    --> 657             self._check_proba()
        658             return self._predict_proba
        659


        /opt/conda/lib/python3.7/site-packages/sklearn/svm/_base.py in␣
↪_check_proba(self)
        622         def _check_proba(self):
        623             if not self.probability:
    --> 624                 raise AttributeError("predict_proba is not available␣
↪when "
        625                                      " probability=False")
        626             if self._impl not in ('c_svc', 'nu_svc'):


        AttributeError: predict_proba is not available when  probability=False
```

As the error message suggests, we can obtain probabilistic predictions for each observation by setting `probability=True` before fitting our model. Details on what is involved in this process are provided in the sklearn user guide here. Generally, if probabilistic predictions are needed then an alternative modelling method is likely to be superior.

With the available scoing metrics we can make use of the usual cross valiation tools such as `cross_val_score` which can be useful for the purpose of comparing different models,

[89]:
```python
rbf = sklearn.model_selection.cross_val_score(
    SVC(kernel='rbf', C=1, gamma='scale'),
    X_ex4, y_ex4,
    cv=KFold(5, shuffle=True, random_state=1234)
)

print(rbf)
print(rbf.mean())
```

```
[0.925  0.975  0.9125 0.975  0.925 ]
0.9425000000000001
```

```
[80]: poly = sklearn.model_selection.cross_val_score(
          SVC(kernel='poly', degree=2, C=1, gamma='scale'),
          X_ex4, y_ex4,
          cv=KFold(5, shuffle=True, random_state=1234)
      )

      print(poly)
      print(poly.mean())
```

```
[0.9375 0.975  0.8875 0.95   0.925 ]
0.9349999999999999
```

Which indicates that the rbf appears to slightly outperform the degree 2 polynomial, but only for `C=1`.

3.5.1 Exercise 11

If you adjust `C` are you able to find a better performing version of either the `rbf` or `poly` SVM models?

```
[133]: for i in np.logspace(-15, 3, num=10):
           rbf = sklearn.model_selection.cross_val_score(
               SVC(kernel='rbf', C=i, gamma='scale'),
               X_ex4, y_ex4,
               cv=KFold(5, shuffle=True, random_state=1234)
           )

           #print(rbf)
           print(i, rbf.mean())


       for i in np.logspace(-15, 3, num=10):
           poly = sklearn.model_selection.cross_val_score(
               SVC(kernel='poly', degree=2, C=i, gamma='scale'),
               X_ex4, y_ex4,
               cv=KFold(5, shuffle=True, random_state=1234)
           )

           #print(poly)
           print(i, poly.mean())
```

```
1e-15 0.45499999999999996
1e-13 0.45499999999999996
1e-11 0.45499999999999996
1e-09 0.45499999999999996
1e-07 0.45499999999999996
```

```
1e-05 0.45499999999999996
0.001 0.45499999999999996
0.1 0.9400000000000001
10.0 0.9349999999999999
1000.0 0.9349999999999999
1e-15 0.45499999999999996
1e-13 0.45499999999999996
1e-11 0.45499999999999996
1e-09 0.45499999999999996
1e-07 0.45499999999999996
1e-05 0.45499999999999996
0.001 0.45499999999999996
0.1 0.9275
10.0 0.945
1000.0 0.9450000000000001
```

Therefore, we see that the score is maximized as c approaches zero i.e with the biggest margin

---

3.5.2 Exercise 12

Construct a full cross validated grid search over the parameter values: `C = [1,5,10,50,100]`,`degree = [2,3,4]`, and `kernel = ['poly', 'rbf', 'linear']`. Which SVM model performs best? Use `plot_margin` to show the resulting seperator and support vectors.

[132]:
```
# # C1 = [1,5,10,50,100]
# # degree1 = [2,3,4]
# # kernel1 = ['poly', 'rbf', 'linear']

param_grid = {'C' : [1,5,10,50,100],  'degree' : [2, 3, 4], 'kernel' :␣
↪['linear']}

# # # l_gs2 = GridSearchCV(
# # #     make_pipeline(
# # #         SVM()
# # #     ),
# # #     param_grid=param_grid1,
# # #     cv=KFold(5, shuffle=True, random_state=1),
# # #     scoring="neg_root_mean_squared_error"
# # # ).fit(X_ex4, y_ex4)

# # grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# # # fitting the model for grid search
# # grid.fit(X_ex4, y_ex4)

# # l_gs = GridSearchCV(
# #     make_pipeline(
# #         SVM()
# #     )
# # )

param_grid1 = {'C' : [1,5,10,50,100],  'degree' : [2, 3, 4], 'kernel' :␣
 ↪['linear','poly','rbf']}
grid_search1 = GridSearchCV(SVC(), param_grid, cv=5)
grid_search1.fit(X_ex4, y_ex4)

print(grid_search1.best_estimator_)
#print(sklearn.model_selection.cross_val_score(grid_search1, X_ex4, y_ex4))
```

```
SVC(C=1, degree=2, kernel='linear')
```

---

4 2.6. Additional Reading

The sklearn user guide for SVM models is particularly good compared to some of the others we have see so far. I suggets browsing through it as a good overview of the modeling method in addition to what has been covered in lecture. Specifically, I would also like to highly recommend section 1.4.5 Tips on Practical Use has excellent guidance on usage of these models.

4.1 3. Competing the worksheet

At this point you have hopefully been able to complete all the preceeding exercises. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF and turn it in on gradescope under the `mlp-week08` assignment.

```
[ ]: !jupyter nbconvert --to pdf mlp-week08.ipynb
```

Created in Deepnote