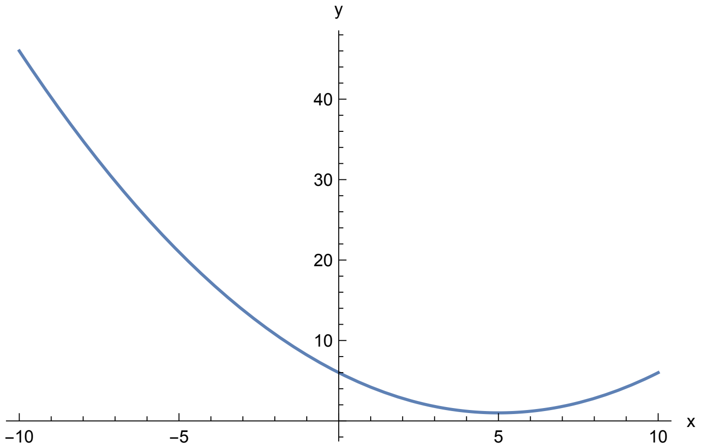
3rd example support vector machine

In this example we revisit the previous example and modify it slightly by moving around the parabola and ellipse.

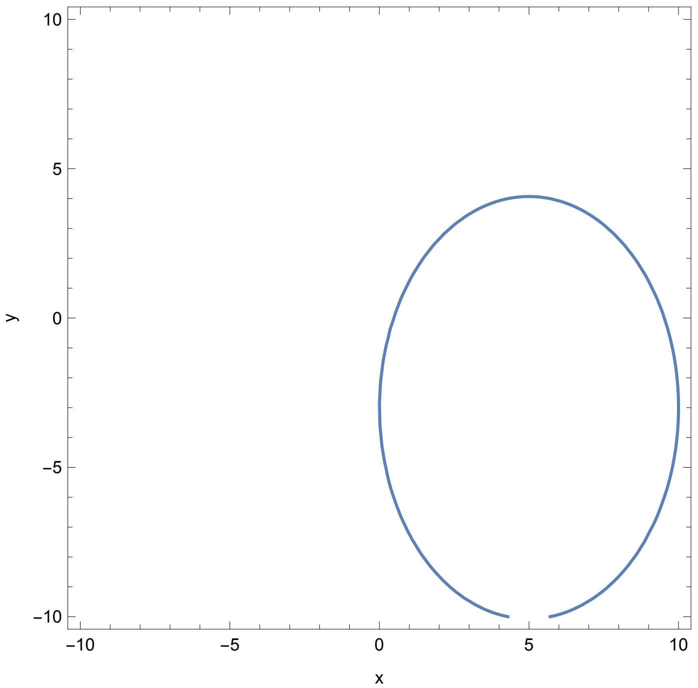
In[76]:=
```
f2b[x_] := 0.2 (x - 5) ^2 + 1
f3b[x_] := (x[[1]] - 5) ^2 + 0.5 (x[[2]] + 3) ^2
```

In[78]:=
```
Plot[f2b[x], {x, -10, 10}, AxesLabel → {"x", "y"}]
ContourPlot[f3b[{x, y}] == 25, {x, -10, 10}, {y, -10, 10}, FrameLabel → {"x", "y"}]
```

Out[78]=



Out[79]=



First of all we need to generate some random data points and our training data set (as before):

```
alldata[n_] := RandomReal[{-10, 10}, {n, 2}]
```

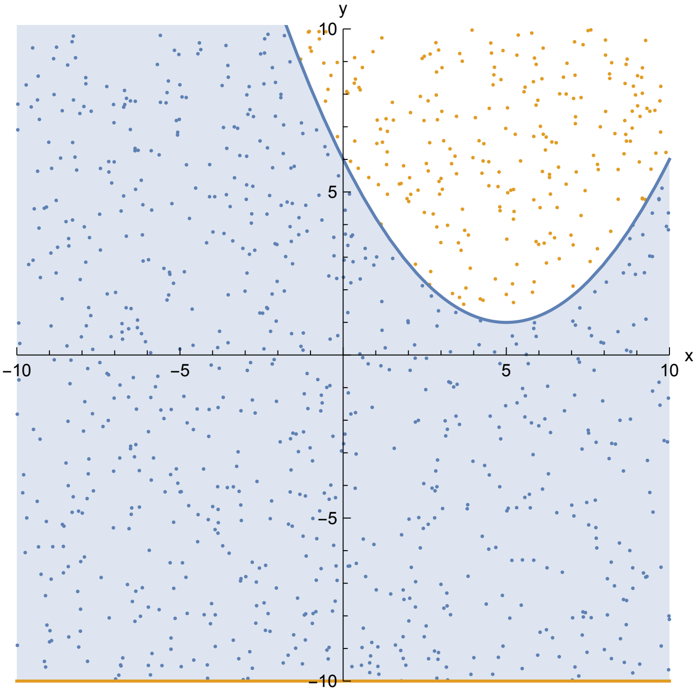
In[80]:=
```
ts2b[datax_] := Module[{data = datax}, data2 = {};
  Do[If[(f2b[data[[i, 1]]] - data[[i, 2]]) ≥ 0, data2 = Append[data2, {data[[i]],
       1}], data2 = Append[data2, {data[[i]], -1}]], {i, 1, Length[data]}];
  data2]
ts3b[datax_] := Module[{data = datax}, data2 = {};
  Do[If[(f3b[data[[i]]]) <= 25, data2 = Append[data2, {data[[i]], 1}],
    data2 = Append[data2, {data[[i]], -1}]], {i, 1, Length[data]}];
  data2]
```

In[82]:=
```
gb[datax_] := Module[{data = datax}, g = {};
  b = {};
  Do[If[data[[i, 2]] == 1, g = Append[g, data[[i, 1]]],
    b = Append[b, data[[i, 1]]]], {i, 1, Length[data]}];
  {g, b}]
```

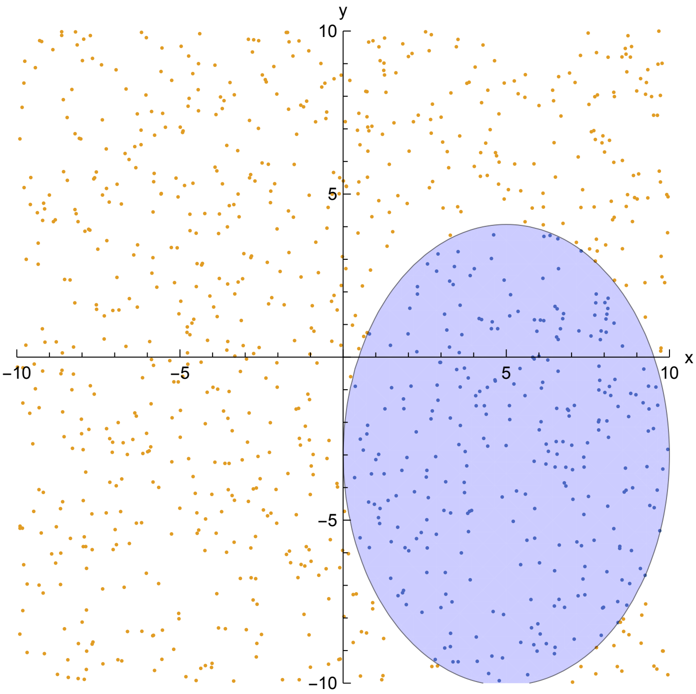
Let’s look at our data:

In[83]:=
```
Show[ListPlot[gb[ts2b[alldata[1000]]],
  PlotRange → {{-10, 10}, {-10, 10}}, AspectRatio → 1, AxesLabel → {"x", "y"}],
 Plot[{f2b[x], -10}, {x, -10, 10}, Filling → {1 → {2}}]]
Show[ListPlot[gb[ts3b[alldata[1000]]], PlotRange → {{-10, 10}, {-10, 10}},
  AspectRatio → 1, AxesLabel → {"x", "y"}],
 ContourPlot[f3b[{x, y}], {x, -10, 10}, {y, -10, 10}, Contours → {25},
  ContourShading → {Directive[Blue, Opacity[0.2]], None}]]
```

Out[83]=



Out[84]=

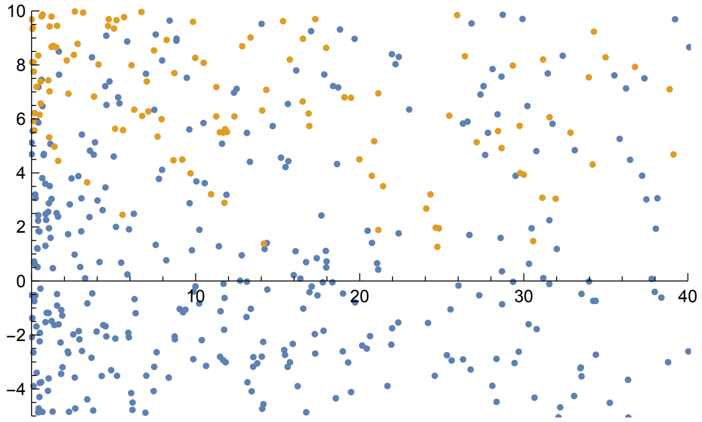


Let's look at the kernel map from before

In[96]:=
```
kmc[x_] := {x[[1]]^2, x[[2]]};
```

In[126]:=
```
d3 = alldata[1000];
d3a = gb[ts2b[d3]][[1]];
d3b = gb[ts2b[d3]][[2]];
```

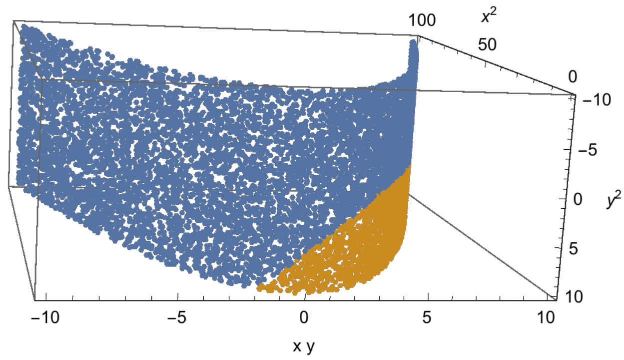
In[129]:=
```
ListPlot[{Table[kmc[d3a[[i]]], {i, 1, Length[d3a]}],
   Table[kmc[d3b[[i]]], {i, 1, Length[d3b]}]},
 PlotRange → {{0, 40}, {-5, 10}}, PlotStyle → PointSize[0.01]]
```

Out[129]=


This map does not separate our data nicely anymore. We can look at the data in 3D with a different kernel map.

In[106]:=
```
kmb2[x_] := {x[[1]]^2, x[[1]], x[[2]]};
```

In[110]:=
```
ListPointPlot3D[{Table[kmb2[d3a[[i]]], {i, 1, Length[d3a]}],
   Table[kmb2[d3b[[i]]], {i, 1, Length[d3b]}]}, AxesLabel → {"x²", " x y", " y²"}]
```

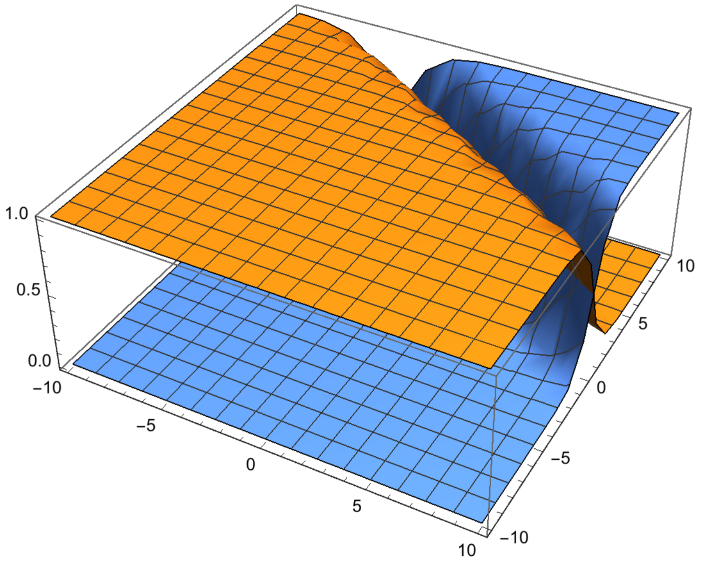
Out[110]=


In[134]:=
```
c = Classify[<|1 → d3a, -1 → d3b|>,
   Method → {"SupportVectorMachine", "KernelType" → "Linear"}]
```

Out[134]= ClassifierFunction[Input type: NumericalVector (length: 2) Classes: −1, 1]

In[136]:=
```
Plot3D[{
   c[{x, y}, "Probability" → 1],
   c[{x, y}, "Probability" → -1]
  },
  {x, -10, 10}, {y, -10, 10},
  Exclusions → None]
```

Out[136]=



Doesn’t look too good, let’s quantify how bad this is:

test data set:

In[201]:=
```
t3 = alldata[10 000];
t3a = gb[ts2b[t3]][[1]];
t3b = gb[ts2b[t3]][[2]];
```

In[186]:=
```
N[Count[c[t3a], 1] / Length[t3a]]
N[Count[c[t3a], -1] / Length[t3a]]
N[Count[c[t3b], 1] / Length[t3b]]
N[Count[c[t3b], -1] / Length[t3b]]
```

Out[186]= 0.9714

Out[187]= 0.0285997

Out[188]= 0.25169

Out[189]= 0.74831

Now, let’s do the same thing with the kernel map:

In[158]:=
```
c2 = Classify[<|1 → Table[kmb2[d3a[[i]]], {i, 1, Length[d3a]}],
    -1 → Table[kmb2[d3b[[i]]], {i, 1, Length[d3b]}]|>,
  Method → {"SupportVectorMachine", "KernelType" → "Linear"}]
```

Out[158]=
```
ClassifierFunction[ Input type: NumericalVector (length: 3)
                    Classes: -1, 1 ]
```

In[190]:=
```
N[Count[c2[Table[kmb2[t3a[[i]]], {i, 1, Length[t3a]}]], 1] / Length[t3a]]
N[Count[c2[Table[kmb2[t3a[[i]]], {i, 1, Length[t3a]}]], -1] / Length[t3a]]
N[Count[c2[Table[kmb2[t3b[[i]]], {i, 1, Length[t3b]}]], 1] / Length[t3b]]
N[Count[c2[Table[kmb2[t3b[[i]]], {i, 1, Length[t3b]}]], -1] / Length[t3b]]
```

Out[190]= 0.995171

Out[191]= 0.00482853

Out[192]= 0.0436817

Out[193]= 0.956318

Still, the points in the parabola are classified better but not yet perfectly. Empirically (check!), this is related to the amount of data in the different training sets. There are not enough points in set b.

In[208]:=
```
Length[d3a]
Length[d3b]
```

Out[208]= 812

Out[209]= 188

Let's do the same modification for the ellipsoid

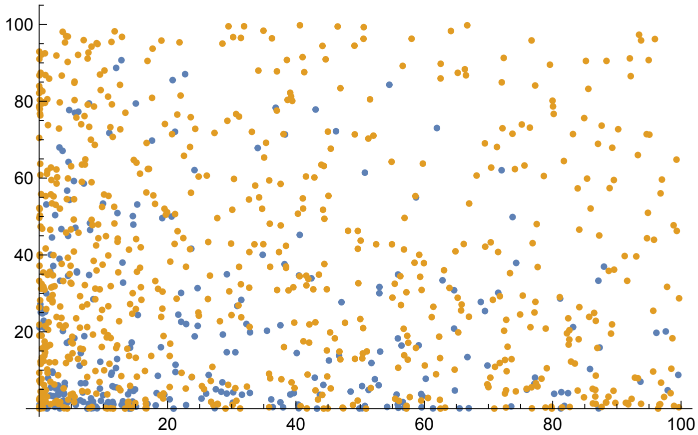
Let's look at the kernel map from before

In[210]:=
```
kmb[x_] := {x[[1]]^2, x[[2]]^2};
```

In[211]:=
```
d3 = alldata[1000];
d3a = gb[ts3b[d3]][[1]];
d3b = gb[ts3b[d3]][[2]];
```

In[215]:=
```
ListPlot[{Table[kmb[d3a[[i]]], {i, 1, Length[d3a]}],
  Table[kmb[d3b[[i]]], {i, 1, Length[d3b]}]}, PlotStyle → PointSize[0.01]]
```

Out[215]=



Let's modify the map

In[218]:=
```
kmb2[x_] := {x[[1]]^2, x[[1]], x[[1]] x[[2]], x[[2]], x[[2]]^2};
```

In[219]:=
```
c4 = Classify[<|1 → Table[kmb2[d3a[[i]]], {i, 1, Length[d3a]}],
    -1 → Table[kmb2[d3b[[i]]], {i, 1, Length[d3b]}]|>,
  Method → {"SupportVectorMachine", "KernelType" → "Linear"}]
```

Out[219]=
```
ClassifierFunction[ Input type: NumericalVector (length: 5)
                    Classes: -1, 1 ]
```

In[224]:=
```
t3 = alldata[10 000];
t3a = gb[ts3b[t3]][[1]];
t3b = gb[ts3b[t3]][[2]];
```

In[227]:=
```
N[Count[c4[Table[kmb2[t3a[[i]]], {i, 1, Length[t3a]}]], 1] / Length[t3a]]
N[Count[c4[Table[kmb2[t3a[[i]]], {i, 1, Length[t3a]}]], -1] / Length[t3a]]
N[Count[c4[Table[kmb2[t3b[[i]]], {i, 1, Length[t3b]}]], 1] / Length[t3b]]
N[Count[c4[Table[kmb2[t3b[[i]]], {i, 1, Length[t3b]}]], -1] / Length[t3b]]
```

Out[227]= 0.9594

Out[228]= 0.0405999

Out[229]= 0.0228461

Out[230]= 0.977154

And we are doing pretty well now...