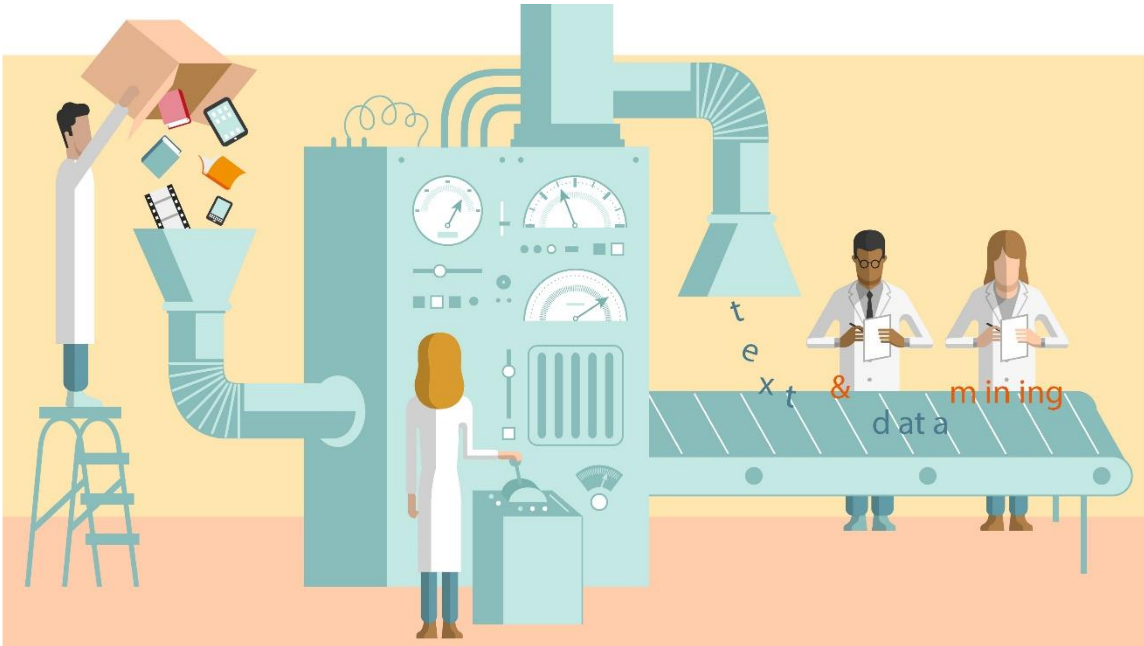


Report

HW3

Yasaman Mirmohammad | Data Mining | Fall_2018

Task 1:Association Rules

Step1:Apriori aloorithm:

Apriori is an algorithm used to identify frequent item sets (in our case, item pairs). It does so using a "bottom up" approach, first identifying individual items that satisfy a minimum occurence threshold. It then extends the item set, adding one item at a time and checking if the resulting item set still satisfies the specified threshold. The algorithm stops when there are no more items to add that meet the minimum occurrence requirement

Step2:**Association Rules Mining**

Once the item sets have been generated using apriori, we can start mining association rules.

One common application of these rules is in the domain of recommender systems, where customers who purchased item A are recommended item B.

Here are 3 key metrics to consider when evaluating association rules:

Here are 3 key metrics to consider when evaluating association rules:

support
This is the percentage of orders that contains the item set. In the example above, there are 5 orders in total and {apple,egg} occurs in 3 of them, so:

support{apple,egg} = 3/5 or 60%

The minimum support threshold required by apriori can be set based on knowledge of your domain. In this grocery dataset for example, since there could be thousands of distinct items and an order can contain only a small fraction of these items, setting the support threshold to 0.01% may be reasonable.

confidence
Given two items, A and B, confidence measures the percentage of times that item B is purchased, given that item A was purchased. This is expressed as:

confidence{A->B} = support{A,B} / support{A}

Confidence values range from 0 to 1, where 0 indicates that B is never purchased when A is purchased, and 1 indicates that B is always purchased whenever A is purchased. Note that the confidence measure is directional. This means that we can also compute the percentage of times that item A is purchased, given that item B was purchased:

confidence{B->A} = support{A,B} / support{B}

In our example, the percentage of times that egg is purchased, given that apple was purchased is:

confidence{apple->egg} = support{apple,egg} / support{apple}

= (3/5) / (4/5)

= 0.75 or 75%

A confidence value of 0.75 implies that out of all orders that contain apple, 75% of them also contain egg. Now, we look at the confidence measure in the opposite direction (ie: egg->apple):

confidence{egg->apple} = support{apple,egg} / support{egg}

= (3/5) / (3/5)

= 1 or 100%

Here we see that all of the orders that contain egg also contain apple. But, does this mean that there is a relationship between these two items, or are they occurring together in the same orders simply by chance? To answer this question, we look at another measure which takes into account the popularity of *both* items.

lift

Given two items, A and B, lift indicates whether there is a relationship between A and B, or whether the two items are occuring together in the same orders simply by chance (ie: at random). Unlike the confidence metric whose value may vary depending on direction (eg: confidence{A->B} may be different from confidence{B->A}), lift has no direction. This means that the lift{A,B} is always equal to the lift{B,A}:

lift{A,B} = lift{B,A} = support{A,B} / (support{A} * support{B})

Questions, set1:

(a) What are the frequent 1-itemsets?
(b) What are the frequent 2-itemsets?
(c) What are the frequent 2-itemsets with support greater or equal to 7? B,E
(d) What are the association rules generated by the A-priori algorithm with a confidence of 1?

('C',) => ('E',) : 1.0

('C',) => ('A',) : 1.0

('C',) => ('A', 'E') : 1.0

('A', 'C') => ('E',) : 1.0

('E', 'C') => ('A',) : 1.0

(e) What is the confidence of the association rule {B}⇒{E} generated by the A-priori algorithm? 0.875

Answer:

#Frequent Itemset :

support('D',): 0.5

support('C',): 0.4

support('A',): 0.6

support('E',): 0.9

support('B',): 0.8

support('E', 'C'): 0.4

support('A', 'C'): 0.4

support('B', 'E'): 0.7

support('D', 'E'): 0.4

support('A', 'E'): 0.5

support('A', 'B'): 0.4

support('A', 'E', 'C'): 0.4

#rules :

confidence('C',) => ('E',) : 1.0

confidence('A',) => ('C',) : 0.667

confidence('C',) => ('A',) : 1.0

confidence('B',) => ('E',) : 0.875

confidence('E',) => ('B',) : 0.778

confidence('D',) => ('E',) : 0.8

confidence('A',) => ('E',) : 0.833

confidence('E',) => ('A',) : 0.556

confidence('A',) => ('B',) : 0.667

confidence('B',) => ('A',) : 0.5

confidence('A',) => ('E', 'C') : 0.667

confidence('C',) => ('A', 'E') : 1.0

confidence('A', 'E') => ('C',) : 0.8

confidence('A', 'C') => ('E',) : 1.0

confidence('E', 'C') => ('A',) : 1.0

Questions,Set2:

(a) What are the frequent itemsets?
(b) How many association rules can you generate from the frequent itemsets with confidence bigger than 0.65?
(c) What are the association rules that you can generate from the frequent itemsets with confidence bigger than 0.8?

confidence('D',) => ('A',) : 1.0

confidence('A',) => ('C',) : 1.0

confidence('B',) => ('E',) : 1.0

confidence('E',) => ('B',) : 1.0

confidence('D',) => ('C',) : 1.0

confidence('D',) => ('A', 'C') : 1.0

confidence('A', 'D') => ('C',) : 1.0

confidence('D', 'C') => ('A',) : 1.0

confidence('A', 'E') => ('C',) : 1.0

confidence('A', 'B') => ('C',) : 1.0

confidence('B', 'C') => ('E',) : 1.0

confidence('E', 'C') => ('B',) : 1.0

confidence('A', 'B') => ('E',) : 1.0

confidence('A', 'E') => ('B',) : 1.0

confidence('A', 'E') => ('C', 'B') : 1.0

confidence('A', 'B') => ('E', 'C') : 1.0

confidence('A', 'E', 'C') => ('B',) : 1.0

confidence('A', 'B', 'E') => ('C',) : 1.0

confidence('A', 'B', 'C') => ('E',) : 1.0

(d) What is the confidence of the association rule $\{E\}\Rightarrow\{C\}$? 0.667

(e) What is the support value of the association rule $\{B\}\Rightarrow\{C\}$?0.5

Answer:

#Frequent Itemset :

support('D',): 0.25

support('C',): 0.75

support('A',): 0.5

support('E',): 0.75

support('B',): 0.75

support('A', 'D'): 0.25

support('E', 'C'): 0.5

support('B', 'C'): 0.5

support('A', 'C'): 0.5

support('B', 'E'): 0.75

support('D', 'C'): 0.25

support('A', 'E'): 0.25

support('A', 'B'): 0.25

support('A', 'D', 'C'): 0.25

support('A', 'E', 'C'): 0.25

support('A', 'B', 'C'): 0.25

support('B', 'E', 'C'): 0.5

support('A', 'B', 'E'): 0.25

support('E', 'C', 'A', 'B'): 0.25

#rules :

confidence('A',) => ('D',) : 0.5

confidence('D',) => ('A',) : 1.0

confidence('E',) => ('C',) : 0.667

confidence('C',) => ('E',) : 0.667

confidence('B',) => ('C',) : 0.667

confidence('C',) => ('B',) : 0.667

confidence('A',) => ('C',) : 1.0

confidence('C',) => ('A',) : 0.667

confidence('B',) => ('E',) : 1.0

confidence('E',) => ('B',) : 1.0

confidence('D',) => ('C',) : 1.0

confidence('C',) => ('D',) : 0.333

confidence('A',) => ('E',) : 0.5

confidence('E',) => ('A',) : 0.333

confidence('A',) => ('B',) : 0.5

confidence('B',) => ('A',) : 0.333

confidence('A',) => ('D', 'C') : 0.5

confidence('D',) => ('A', 'C') : 1.0

confidence('C',) => ('A', 'D') : 0.333

confidence('A', 'D') => ('C',) : 1.0

confidence('A', 'C') => ('D',) : 0.5

confidence('D', 'C') => ('A',) : 1.0

confidence('A',) => ('E', 'C') : 0.5

confidence('E',) => ('A', 'C') : 0.333

confidence('C',) => ('A', 'E') : 0.333

confidence('A', 'E') => ('C',) : 1.0

confidence('A', 'C') => ('E',) : 0.5

confidence('E', 'C') => ('A',) : 0.5

confidence('A',) => ('B', 'C') : 0.5

confidence('B',) => ('A', 'C') : 0.333

confidence('C',) => ('A', 'B') : 0.333

confidence('A', 'B') => ('C',) : 1.0

confidence('A', 'C') => ('B',) : 0.5

confidence('B', 'C') => ('A',) : 0.5

confidence('B',) => ('E', 'C') : 0.667

confidence('E',) => ('B', 'C') : 0.667

confidence('C',) => ('B', 'E') : 0.667

confidence('B', 'E') => ('C',) : 0.667

confidence('B', 'C') => ('E',) : 1.0

confidence('E', 'C') => ('B',) : 1.0

confidence('A',) => ('B', 'E') : 0.5

confidence('B',) => ('A', 'E') : 0.333

confidence('E',) => ('A', 'B') : 0.333

confidence('A', 'B') => ('E',) : 1.0

confidence('A', 'E') => ('B',) : 1.0

confidence('B', 'E') => ('A',) : 0.333

confidence('E',) => ('A', 'B', 'C') : 0.333

confidence('C',) => ('A', 'B', 'E') : 0.333

confidence('A',) => ('B', 'E', 'C') : 0.5

confidence('B',) => ('A', 'E', 'C') : 0.333

confidence('E', 'C') => ('A', 'B') : 0.5

confidence('A', 'E') => ('C', 'B') : 1.0

confidence('B', 'E') => ('C', 'A') : 0.333

confidence('A', 'C') => ('E', 'B') : 0.5

confidence('B', 'C') => ('E', 'A') : 0.5

confidence('A', 'B') => ('E', 'C') : 1.0

confidence('A', 'E', 'C') => ('B',) : 1.0

confidence('B', 'E', 'C') => ('A',) : 0.5

confidence('A', 'B', 'E') => ('C',) : 1.0

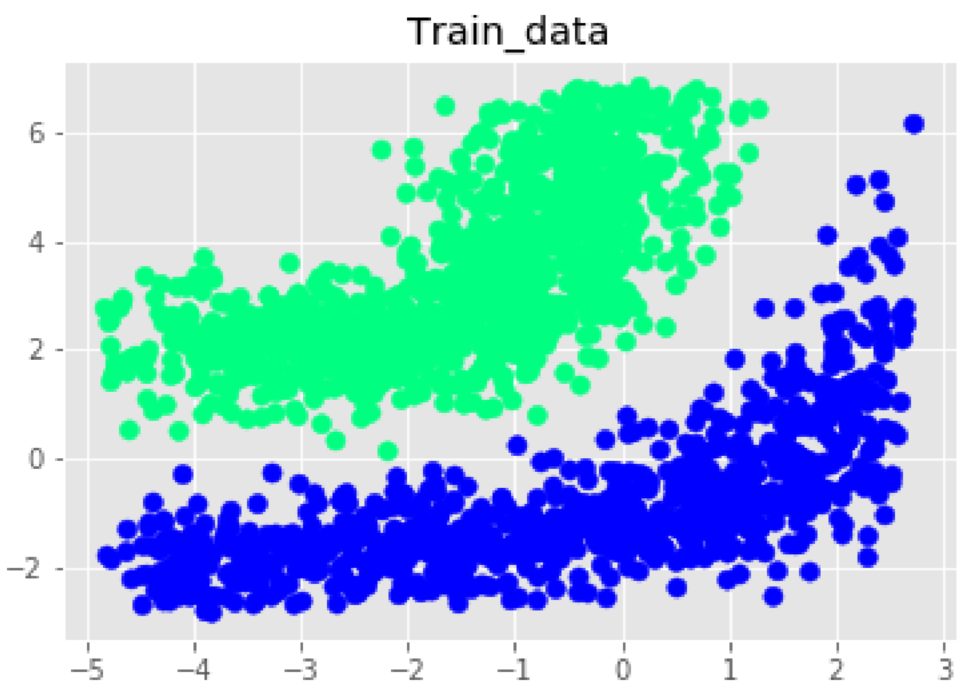
confidence('A', 'B', 'C') => ('E',) : 1.0

Task 2: SVM Classifier

Step1: train_test_validation split:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=1)
```

Step2: Plot train set



Step3: linear SVM

This dataset is not linearly seperable , so we have to use softmax margin(linearSVC).

In case of non-linearly separable data, the simple SVM algorithm cannot be used. Rather, a modified version of SVM, called Kernel SVM, is used.

Basically, the kernel SVM projects the non-linearly separable data lower dimensions to linearly separable data in higher dimensions in such a way that data points belonging to different classes are allocated to different dimensions. Again, there is complex mathematics involved in this, but you do not have to worry about it in order to use SVM. Rather we can simply use Python's Scikit-Learn library that to implement and use the kernel SVM.

The most basic way to use a SVC is with a linear kernel, which means the decision boundary is a straight line (or hyperplane in higher dimensions). Linear kernels are rarely used in practice, however I wanted to show it here since it is the most basic version of SVC. As can been seen below, it is not very good at classifying (which can be seen by all the blue X's in the red region) because the data is not linear.

(source:https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/)

Step4: effect of c parameter

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable

Result:

```
C: 0.01

val acc 0.9913194444444444

test acc 0.9777777777777777

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 0.9973684210526316
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9842767295597484
test recall: 0.9619289340101523
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.2041684574315704, 0.18633899812498247, 0.29814239699997
197]
validation_error [0.2041684574315704, 0.18633899812498247, 0.2981423969
9997197]
test_error [0.2041684574315704, 0.18633899812498247, 0.2981423969999719
7]
*****************    ************************    ******************
**********************   **************************
C: 0.21000000000000002
val acc 0.9895833333333334
test acc 0.9791666666666666
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9894736842105263
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.9715762273901809
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1954764156937815, 0.2041241452319315, 0.288675134594812
87]
validation_error [0.1954764156937815, 0.2041241452319315, 0.28867513459
481287]
test_error [0.1954764156937815, 0.2041241452319315, 0.28867513459481287
]
```

```
*****************    ***********************    ******************
**********************   **************************
C: 0.41000000000000003
val acc 0.9895833333333334
test acc 0.9819444444444444
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9894736842105263
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.9766233766233766
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1909821042237691, 0.2041241452319315, 0.268741924943284
97]
validation_error [0.1909821042237691, 0.2041241452319315, 0.26874192494
328497]
test_error [0.1909821042237691, 0.2041241452319315, 0.26874192494328497
]
*****************    ***********************    ******************
**********************   **************************
C: 0.6100000000000001
val acc 0.9895833333333334
test acc 0.9819444444444444
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9894736842105263
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.9766233766233766
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1954764156937815, 0.2041241452319315, 0.268741924943284
97]
validation_error [0.1954764156937815, 0.2041241452319315, 0.26874192494
328497]
test_error [0.1954764156937815, 0.2041241452319315, 0.26874192494328497
]
*****************    ***********************    ******************
**********************   **************************
C: 0.81
val acc 0.9895833333333334
test acc 0.9819444444444444
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
```

```
test prec: 0.9894736842105263
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.9766233766233766
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1954764156937815, 0.2041241452319315, 0.268741924943284
97]
validation_error [0.1954764156937815, 0.2041241452319315, 0.26874192494
328497]
test_error [0.1954764156937815, 0.2041241452319315, 0.26874192494328497
]
*****************    ************************    ******************
*********************   **************************
C: 1.01
val acc 0.9895833333333334
test acc 0.9791666666666666
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9868421052631579
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.974025974025974
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1909821042237691, 0.2041241452319315, 0.288675134594812
87]
validation_error [0.1909821042237691, 0.2041241452319315, 0.28867513459
481287]
test_error [0.1909821042237691, 0.2041241452319315, 0.28867513459481287
]
*****************    ************************    ******************
*********************   **************************
C: 1.2100000000000002
val acc 0.9895833333333334
test acc 0.9791666666666666
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9868421052631579
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.974025974025974
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1909821042237691, 0.2041241452319315, 0.288675134594812
87]
```

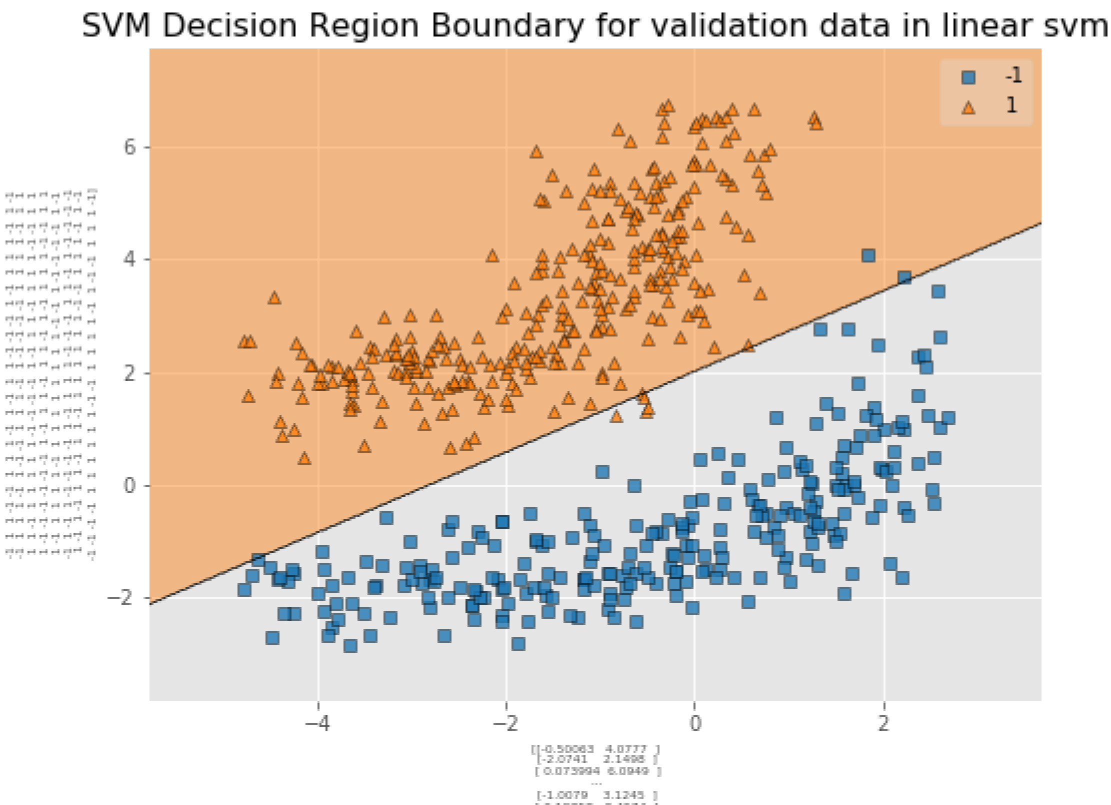
```
validation_error [0.1909821042237691, 0.2041241452319315, 0.28867513459
481287]
test_error [0.1909821042237691, 0.2041241452319315, 0.28867513459481287
]
*****************    ************************    ******************
**********************   **************************
C: 1.4100000000000001
val acc 0.9878472222222222
test acc 0.9791666666666666
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.9840255591054313
test prec: 0.9868421052631579
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935483870967742
test recall: 0.974025974025974
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.1909821042237691, 0.22047927592204922, 0.28867513459481
287]
validation_error [0.1909821042237691, 0.22047927592204922, 0.2886751345
9481287]
test_error [0.1909821042237691, 0.22047927592204922, 0.2886751345948128
7]
```

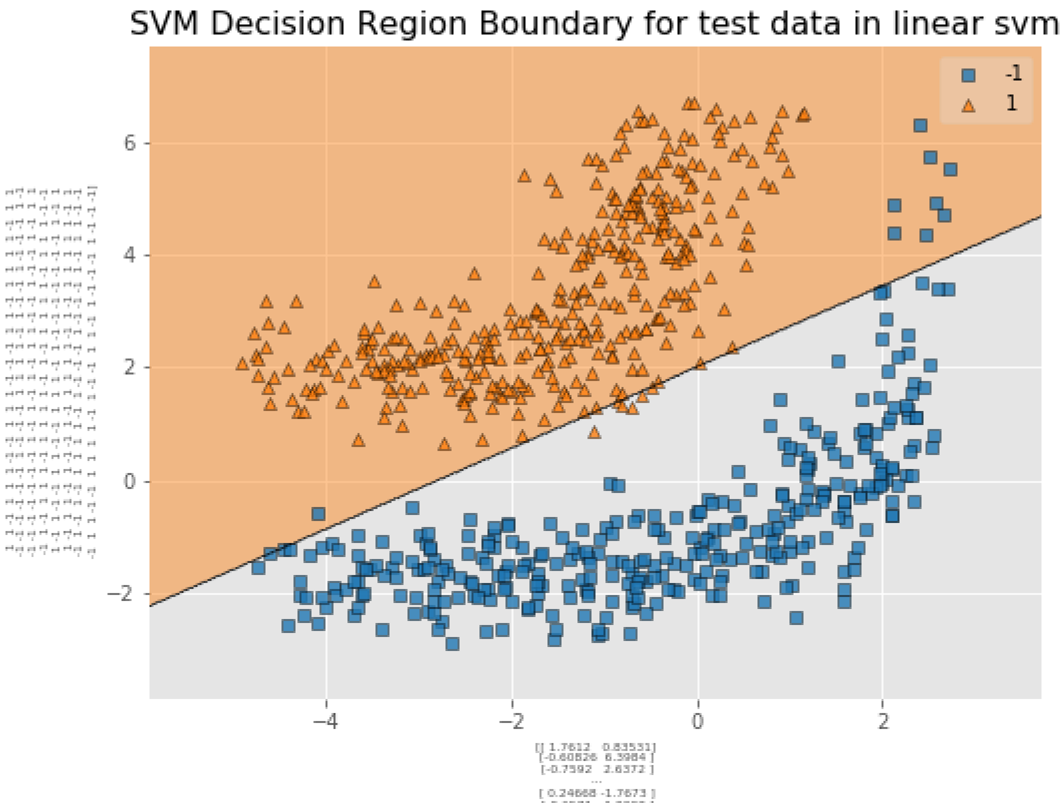
```
Confusion matrix:
[[330  10]
 [  5 375]]
****************************************************************************
             precision    recall  f1-score   support

         -1       0.99      0.97      0.98       340
          1       0.97      0.99      0.98       380

avg / total       0.98      0.98      0.98       720
```

C=1

```
val acc 0.9895833333333334
test acc 0.9791666666666666
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 0.987220447284345
test prec: 0.9868421052631579
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 0.9935691318327974
test recall: 0.974025974025974
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
train_error: [0.1909821042237691, 0.2041241452319315, 0.288675134594
81287]
validation_error [0.1909821042237691, 0.2041241452319315, 0.28867513
459481287]
test_error [0.1909821042237691, 0.2041241452319315, 0.28867513459481
287]
```





part2: nonlinear SVM

Where SVM becomes extremely powerful is when it is combined with *kernels*. We have seen a version of kernels before, in the basis function regressions of [In Depth: Linear Regression](). There we projected our data into higher-dimensional space defined by polynomials and Gaussian basis functions, and thereby were able to fit for nonlinear relationships with a linear classifier.

It is clear that no linear discrimination will *ever* be able to separate this data. But we can draw a lesson from the basis function regressions in [In Depth: Linear Regression](), and think about how we might project the data into a higher dimension such that a linear separator *would* be sufficient. For example, one simple projection we could use would be to compute a *radial basis function*centered on the middle clump:

We can visualize this extra data dimension using a three-dimensional plot—if you are running this notebook live, you will be able to use the sliders to rotate the plot:

CLASSIFY USING A RBF KERNEL

Radial Basis Function is a commonly used kernel in SVC:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

where $||x-x'||2||x-x'||2$ is the squared Euclidean distance between two data points xx and $x'x'$. If this doesn't make sense, Sebastian's book has a full description. However, for this tutorial, it is only important to know that an SVC classifier using an RBF kernel has two parameters: `gamma` and `C`.

Gamma

`gamma` is a parameter of the RBF kernel and can be thought of as the 'spread' of the kernel and therefore the decision region. When `gamma` is low, the 'curve' of the decision boundary is very low and thus the decision region is very broad. When `gamma` is high, the 'curve' of the decision boundary is high, which creates islands of decision-boundaries around data points. We will see this very clearly below.

C

`C` is a parameter of the SVC learner and is the penalty for misclassifying a data point. When `C` is small, the classifier is okay with misclassified data points (high bias, low variance). When `C` is large, the classifier is heavily penalized for misclassified data and therefore bends over backwards avoid any misclassified data points (low bias, high variance).

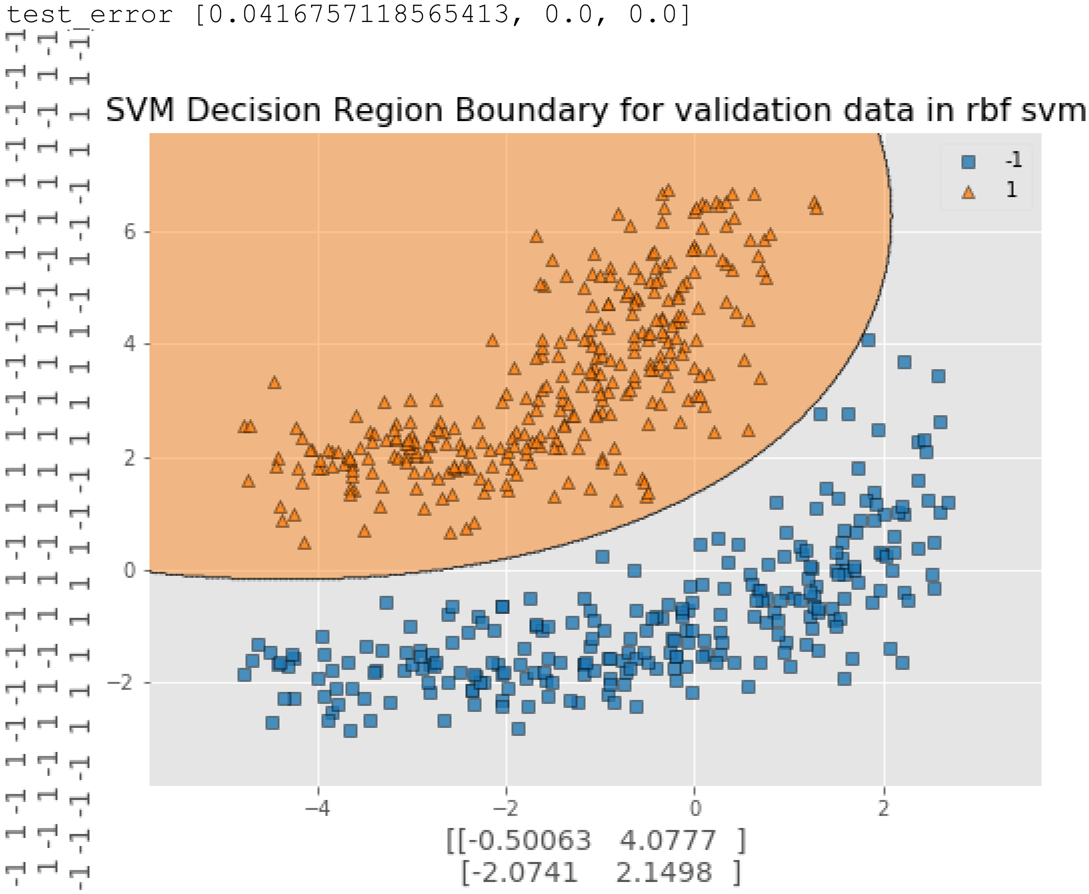
```
C: 0.01
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 0.21000000000000002
```

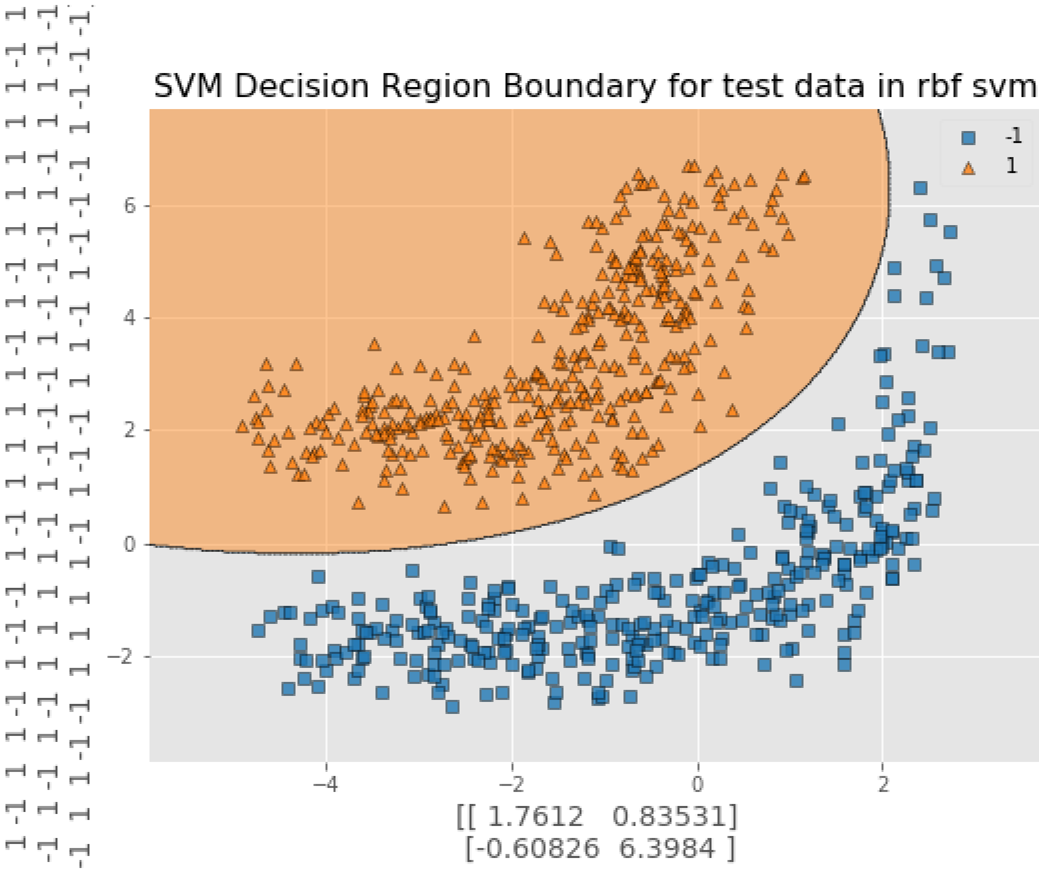
```
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 0.41000000000000003
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 0.6100000000000001
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 0.81
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 1.01
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 1.2100000000000002
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
*****************    ************************    ******************
**********************   **************************
C: 1.4100000000000001
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
```

```
Confusion matrix:
[[340   0]
 [  0 380]]
**********************************************************************
              precision    recall  f1-score   support

         -1       1.00      1.00      1.00       340
          1       1.00      1.00      1.00       380

avg / total       1.00      1.00      1.00       720
```

```
C: 1
val acc 1.0
test acc 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val prec: 1.0
test prec: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val recall: 1.0
test recall: 1.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
train_error: [0.0416757118565413, 0.0, 0.0]
validation_error [0.0416757118565413, 0.0, 0.0]
test_error [0.0416757118565413, 0.0, 0.0]
```





Method: Using GridSearch

```
from sklearn.grid_search import GridSearchCV
```

```
param_grid = {'C':[1,10,100,1000],'gamma':[1,0.1,0.001,0.0001], 'kernel':['linear','rbf']}
grid = GridSearchCV(SVC(),param_grid,refit = True, verbose=2)
```

```
grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
[CV] C=1, gamma=1, kernel=linear .....................................
[CV] ............................ C=1, gamma=1, kernel=linear -   0.0s
[CV] C=1, gamma=1, kernel=linear .....................................
[CV] ............................ C=1, gamma=1, kernel=linear -   0.0s
[CV] C=1, gamma=1, kernel=linear .....................................
[CV] ............................ C=1, gamma=1, kernel=linear -   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............................... C=1, gamma=1, kernel=rbf -   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............................... C=1, gamma=1, kernel=rbf -   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............................... C=1, gamma=1, kernel=rbf -   0.0s
[CV] C=1, gamma=0.1, kernel=linear ...................................
[CV] .......................... C=1, gamma=0.1, kernel=linear -   0.0s
[CV] C=1, gamma=0.1, kernel=linear ...................................
[CV] .......................... C=1, gamma=0.1, kernel=linear -   0.0s
[CV] C=1, gamma=0.1, kernel=linear ...................................
[CV] .......................... C=1, gamma=0.1, kernel=linear -   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] ............................. C=1, gamma=0.1, kernel=rbf -   0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s

[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] ............................. C=1, gamma=0.1, kernel=rbf -   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] ............................. C=1, gamma=0.1, kernel=rbf -   0.0s
[CV] C=1, gamma=0.001, kernel=linear .................................
[CV] ........................ C=1, gamma=0.001, kernel=linear -   0.0s
[CV] C=1, gamma=0.001, kernel=linear .................................
[CV] ........................ C=1, gamma=0.001, kernel=linear -   0.0s
[CV] C=1, gamma=0.001, kernel=linear .................................
[CV] ........................ C=1, gamma=0.001, kernel=linear -   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........................... C=1, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........................... C=1, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........................... C=1, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1, gamma=0.0001, kernel=linear ................................
[CV] ....................... C=1, gamma=0.0001, kernel=linear -   0.0s
[CV] C=1, gamma=0.0001, kernel=linear ................................
```

```
[CV] ....................... C=1, gamma=0.0001, kernel=linear -   0.0s
[CV] C=1, gamma=0.0001, kernel=linear ................................
[CV] ....................... C=1, gamma=0.0001, kernel=linear -   0.0s
[CV] C=1, gamma=0.0001, kernel=rbf ...................................
[CV] .......................... C=1, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=1, gamma=0.0001, kernel=rbf ...................................
[CV] .......................... C=1, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=1, gamma=0.0001, kernel=rbf ...................................
[CV] .......................... C=1, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=10, gamma=1, kernel=linear ....................................
[CV] ........................... C=10, gamma=1, kernel=linear -   0.0s
[CV] C=10, gamma=1, kernel=linear ....................................
[CV] ........................... C=10, gamma=1, kernel=linear -   0.0s
[CV] C=10, gamma=1, kernel=linear ....................................
[CV] ........................... C=10, gamma=1, kernel=linear -   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] .............................. C=10, gamma=1, kernel=rbf -   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] .............................. C=10, gamma=1, kernel=rbf -   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] .............................. C=10, gamma=1, kernel=rbf -   0.0s
[CV] C=10, gamma=0.1, kernel=linear ..................................
[CV] ......................... C=10, gamma=0.1, kernel=linear -   0.0s
[CV] C=10, gamma=0.1, kernel=linear ..................................
[CV] ......................... C=10, gamma=0.1, kernel=linear -   0.0s
[CV] C=10, gamma=0.1, kernel=linear ..................................
[CV] ......................... C=10, gamma=0.1, kernel=linear -   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ............................ C=10, gamma=0.1, kernel=rbf -   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ............................ C=10, gamma=0.1, kernel=rbf -   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ............................ C=10, gamma=0.1, kernel=rbf -   0.0s
[CV] C=10, gamma=0.001, kernel=linear ................................
[CV] ....................... C=10, gamma=0.001, kernel=linear -   0.0s
[CV] C=10, gamma=0.001, kernel=linear ................................
[CV] ....................... C=10, gamma=0.001, kernel=linear -   0.0s
[CV] C=10, gamma=0.001, kernel=linear ................................
[CV] ....................... C=10, gamma=0.001, kernel=linear -   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] .......................... C=10, gamma=0.001, kernel=rbf -   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] .......................... C=10, gamma=0.001, kernel=rbf -   0.0s
```

```
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] .......................... C=10, gamma=0.001, kernel=rbf -   0.0s
[CV] C=10, gamma=0.0001, kernel=linear ...............................
[CV] ...................... C=10, gamma=0.0001, kernel=linear -   0.0s
[CV] C=10, gamma=0.0001, kernel=linear ...............................
[CV] ...................... C=10, gamma=0.0001, kernel=linear -   0.0s
[CV] C=10, gamma=0.0001, kernel=linear ...............................
[CV] ...................... C=10, gamma=0.0001, kernel=linear -   0.0s
[CV] C=10, gamma=0.0001, kernel=rbf ..................................
[CV] ......................... C=10, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=10, gamma=0.0001, kernel=rbf ..................................
[CV] ......................... C=10, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=10, gamma=0.0001, kernel=rbf ..................................
[CV] ......................... C=10, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=100, gamma=1, kernel=linear ...................................
[CV] .......................... C=100, gamma=1, kernel=linear -   0.0s
[CV] C=100, gamma=1, kernel=linear ...................................
[CV] .......................... C=100, gamma=1, kernel=linear -   0.0s
[CV] C=100, gamma=1, kernel=linear ...................................
[CV] .......................... C=100, gamma=1, kernel=linear -   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] ............................. C=100, gamma=1, kernel=rbf -   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] ............................. C=100, gamma=1, kernel=rbf -   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] ............................. C=100, gamma=1, kernel=rbf -   0.0s
[CV] C=100, gamma=0.1, kernel=linear .................................
[CV] ........................ C=100, gamma=0.1, kernel=linear -   0.0s
[CV] C=100, gamma=0.1, kernel=linear .................................
[CV] ........................ C=100, gamma=0.1, kernel=linear -   0.0s
[CV] C=100, gamma=0.1, kernel=linear .................................
[CV] ........................ C=100, gamma=0.1, kernel=linear -   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........................... C=100, gamma=0.1, kernel=rbf -   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........................... C=100, gamma=0.1, kernel=rbf -   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........................... C=100, gamma=0.1, kernel=rbf -   0.0s
[CV] C=100, gamma=0.001, kernel=linear ...............................
[CV] ...................... C=100, gamma=0.001, kernel=linear -   0.0s
[CV] C=100, gamma=0.001, kernel=linear ...............................
[CV] ...................... C=100, gamma=0.001, kernel=linear -   0.0s
[CV] C=100, gamma=0.001, kernel=linear ...............................
```

```
[CV] ...................... C=100, gamma=0.001, kernel=linear -   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ......................... C=100, gamma=0.001, kernel=rbf -   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ......................... C=100, gamma=0.001, kernel=rbf -   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ......................... C=100, gamma=0.001, kernel=rbf -   0.0s
[CV] C=100, gamma=0.0001, kernel=linear ..............................
[CV] ..................... C=100, gamma=0.0001, kernel=linear -   0.0s
[CV] C=100, gamma=0.0001, kernel=linear ..............................
[CV] ..................... C=100, gamma=0.0001, kernel=linear -   0.0s
[CV] C=100, gamma=0.0001, kernel=linear ..............................
[CV] ..................... C=100, gamma=0.0001, kernel=linear -   0.0s
[CV] C=100, gamma=0.0001, kernel=rbf .................................
[CV] ........................ C=100, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=100, gamma=0.0001, kernel=rbf .................................
[CV] ........................ C=100, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=100, gamma=0.0001, kernel=rbf .................................
[CV] ........................ C=100, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=1000, gamma=1, kernel=linear ..................................
[CV] ......................... C=1000, gamma=1, kernel=linear -   0.4s
[CV] C=1000, gamma=1, kernel=linear ..................................
[CV] ......................... C=1000, gamma=1, kernel=linear -   0.6s
[CV] C=1000, gamma=1, kernel=linear ..................................
[CV] ......................... C=1000, gamma=1, kernel=linear -   0.4s
[CV] C=1000, gamma=1, kernel=rbf .....................................
[CV] ............................ C=1000, gamma=1, kernel=rbf -   0.0s
[CV] C=1000, gamma=1, kernel=rbf .....................................
[CV] ............................ C=1000, gamma=1, kernel=rbf -   0.0s
[CV] C=1000, gamma=1, kernel=rbf .....................................
[CV] ............................ C=1000, gamma=1, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.1, kernel=linear ................................
[CV] ....................... C=1000, gamma=0.1, kernel=linear -   0.3s
[CV] C=1000, gamma=0.1, kernel=linear ................................
[CV] ....................... C=1000, gamma=0.1, kernel=linear -   0.4s
[CV] C=1000, gamma=0.1, kernel=linear ................................
[CV] ....................... C=1000, gamma=0.1, kernel=linear -   0.4s
[CV] C=1000, gamma=0.1, kernel=rbf ...................................
[CV] .......................... C=1000, gamma=0.1, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.1, kernel=rbf ...................................
[CV] .......................... C=1000, gamma=0.1, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.1, kernel=rbf ...................................
[CV] .......................... C=1000, gamma=0.1, kernel=rbf -   0.0s
```

```
[CV] C=1000, gamma=0.001, kernel=linear ..............................
[CV] ..................... C=1000, gamma=0.001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.001, kernel=linear ..............................
[CV] ..................... C=1000, gamma=0.001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.001, kernel=linear ..............................
[CV] ..................... C=1000, gamma=0.001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.001, kernel=rbf .................................
[CV] ........................ C=1000, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.001, kernel=rbf .................................
[CV] ........................ C=1000, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.001, kernel=rbf .................................
[CV] ........................ C=1000, gamma=0.001, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.0001, kernel=linear .............................
[CV] .................... C=1000, gamma=0.0001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.0001, kernel=linear .............................
[CV] .................... C=1000, gamma=0.0001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.0001, kernel=linear .............................
[CV] .................... C=1000, gamma=0.0001, kernel=linear -   0.4s
[CV] C=1000, gamma=0.0001, kernel=rbf ................................
[CV] ....................... C=1000, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.0001, kernel=rbf ................................
[CV] ....................... C=1000, gamma=0.0001, kernel=rbf -   0.0s
[CV] C=1000, gamma=0.0001, kernel=rbf ................................
[CV] ....................... C=1000, gamma=0.0001, kernel=rbf -   0.0s
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:    8.4s finished
```

Out[422]:

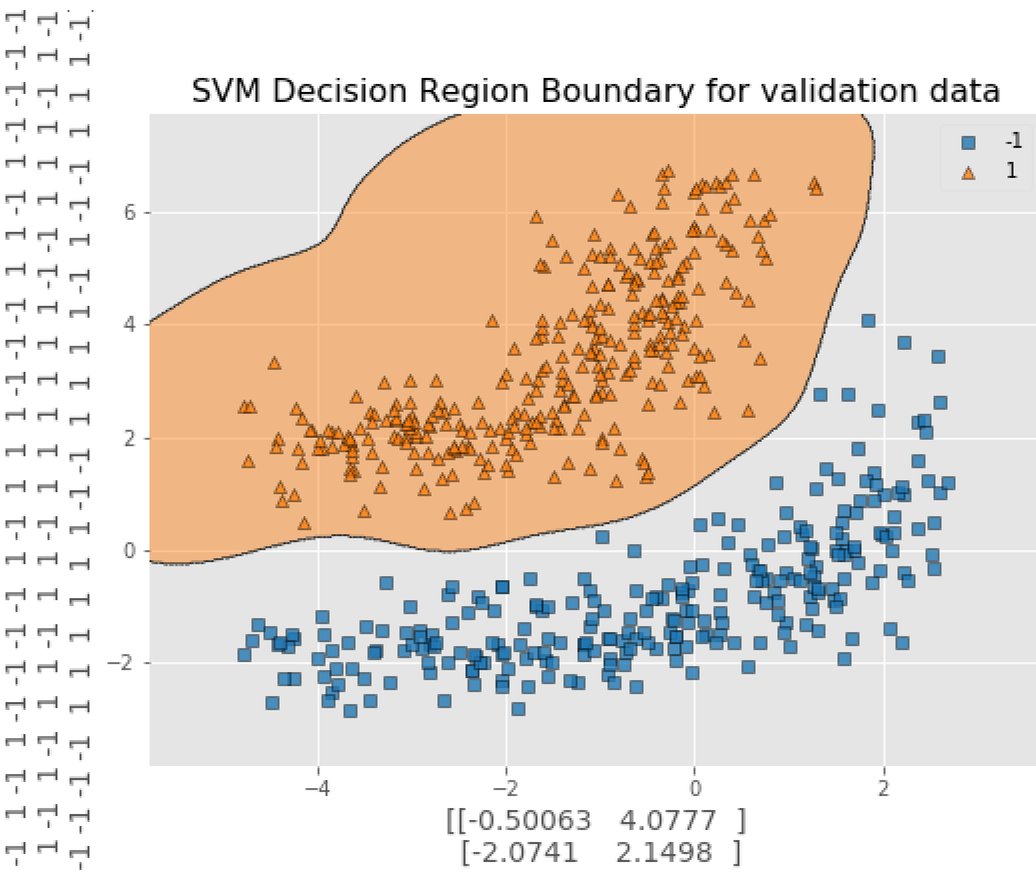
```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.
0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'C': [1, 10, 100, 1000], 'gamma': [1, 0.1, 0.001, 0.
0001], 'kernel': ['linear', 'rbf']},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=2)
```

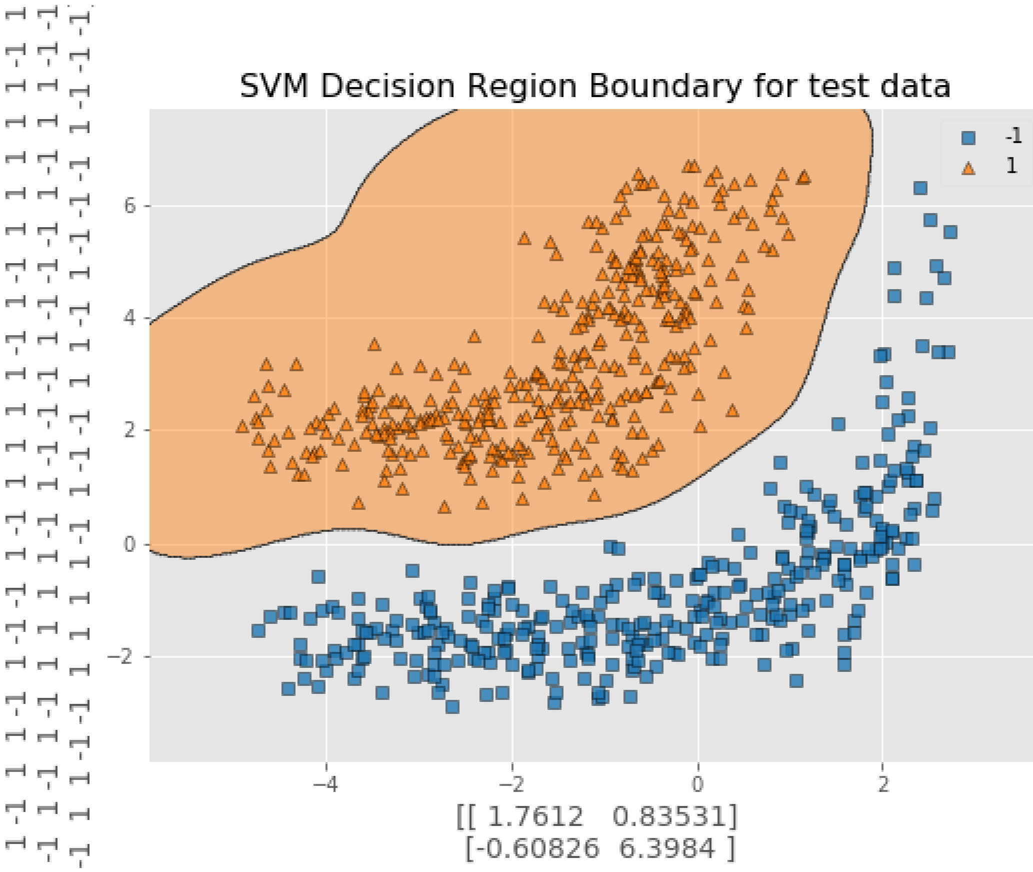
```
grid.best_params_
```

```
{'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

```
              precision    recall  f1-score   support

         -1       1.00      1.00      1.00       263
          1       1.00      1.00      1.00       313

avg / total       1.00      1.00      1.00       576

[[263   0]
 [  0 313]]
********************************************************
val_error: 0.0
```

```
              precision    recall  f1-score   support

         -1       1.00      1.00      1.00       340
          1       1.00      1.00      1.00       380

avg / total       1.00      1.00      1.00       720

[[340   0]
 [  0 380]]
********************************************************
test_error: 0.0
```


SVM Decision Region Boundary for validation data
-1
1
[[-0.50063   4.0777]
[-2.0741   2.1498]



[[1.7612   0.83531]
 [-0.60826 6.3984]]