MACHINE LEARNING FOR DATA MINING
LECTURE 4: SVM

SUPPORT VECTOR MACHINE (SVM) CLASSIFICATION

Siamak Sarmady (UUT, UU)
Sebastian Thrun, Katie Malone (Google)

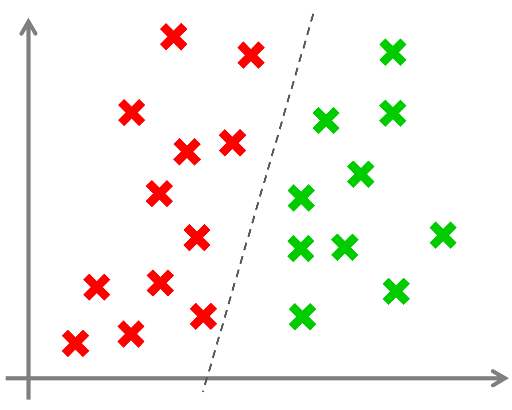
Support Vector Machine (SVM)

- SVM is a very popular classification methods. The method was proposed by Vladimir N. Vapnik in 1963.
- It is a great and powerful algorithm but it could be slow for some applications.
- By default it is a binary classifier. However it is possible to make a multiclass classifier using it.

Concept

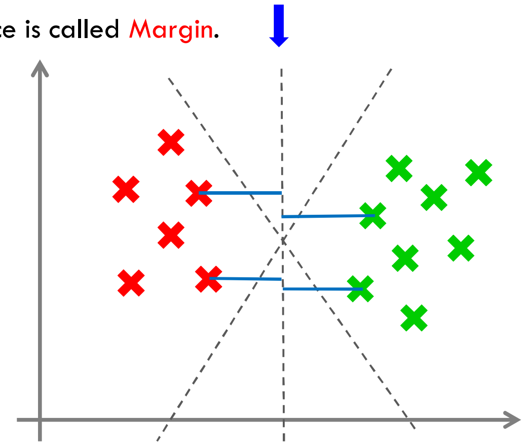
Fitting a line

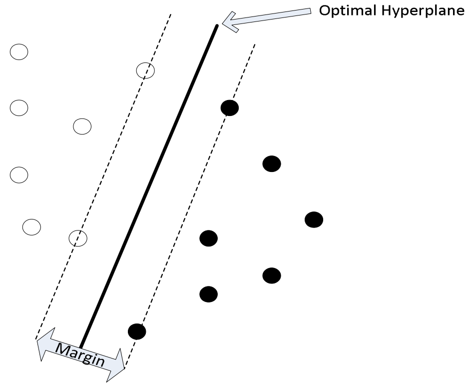
- SVM is a statistical method that finds a suitable line or plane (a hyperplane) that separates data of two classes.



Fitting a separating line

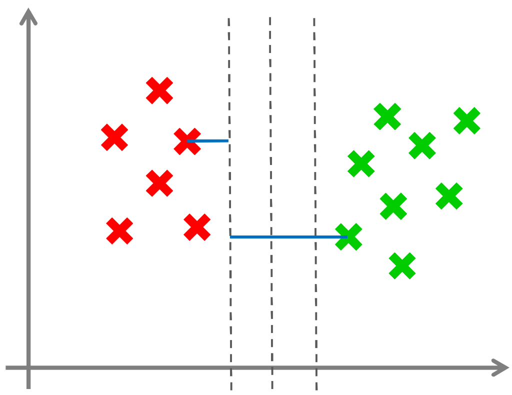
- Which line do you think better separates the data?
- Obviously all 3 lines in the example separate the classes…
- But the marked line is better because it maximizes the distance to the nearest points (concurrently for both classes)
- The distance is called Margin.





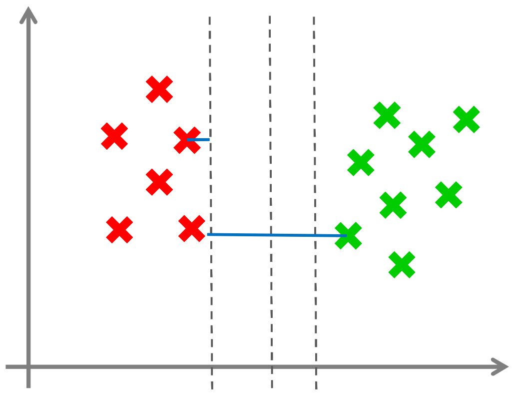
Fitting a separating line

- Which line do you think maximizes the margin better?
- The left one maximizes the distance to the datas in the right side but not to both classes…
- So the line in the middle is better and more robust



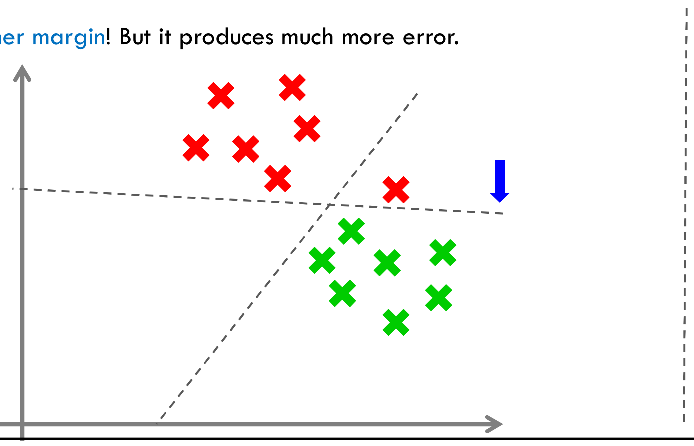
Robust Line

- If we had chosen the left side line, a small noise on the data near to the line would cause it to be missclassified as the right hand side class…
- So the purpose is to maximize the margin and therefore robustness.



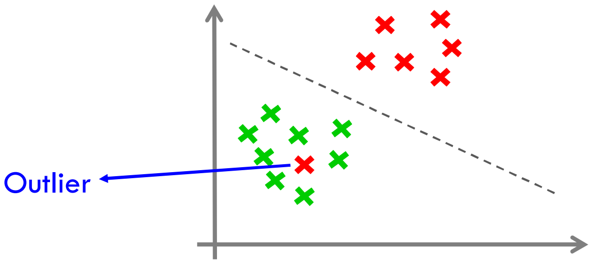
Robustness vs. Error Level

- Which line do you think better separates the data?
- This question is tricky… The SVM will prefer the marked line because eventhough the other one maximizes the margin but it produces classification error…
- Minimizing error has priority over maximizing the margin… SVM first minmizes the classification error then maximizes the margin
- The right most line has even higher margin! But it produces much more error.



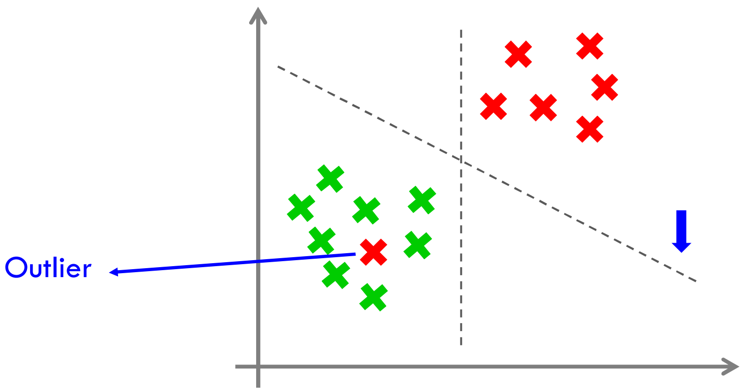
Outliers

- Sometimes it is not possible (or easy) to find a decision surface that totally separates classes …
- So what do you want SVM to do?
  - Give up if it cannot find a perfect decision surface
  - Give a random result (based on the probability)
  - Do the best it can… and specify a decision surface that describes the data best
- Normally the SVM algorithms tolerate individual outliers … otherwise probability of overfitting would exist



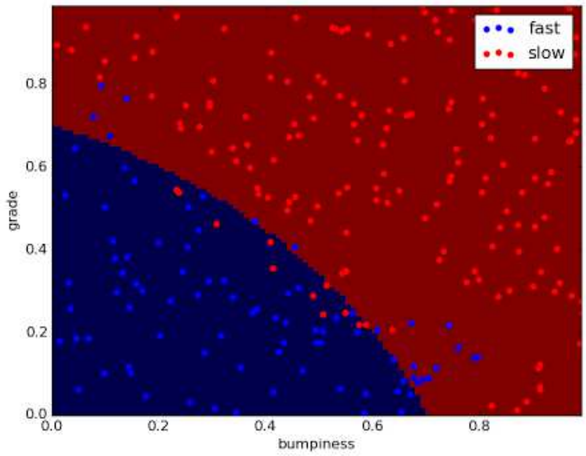
Maximizing Margin and Ignoring Outliers

- Which line do you think is better?
- The line which is marked provides better margin to the nearest points …
- As we will see, SVM has parameters that will determine how willing it is to ignore outliers.

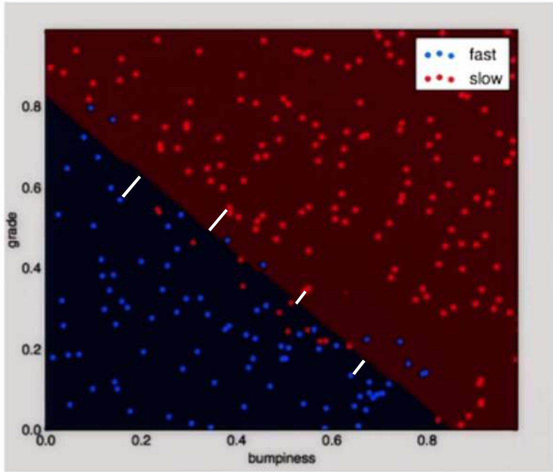


Using Scikit Learn To classify

- With previous data that we used in Naive Base classification



Naïve Base

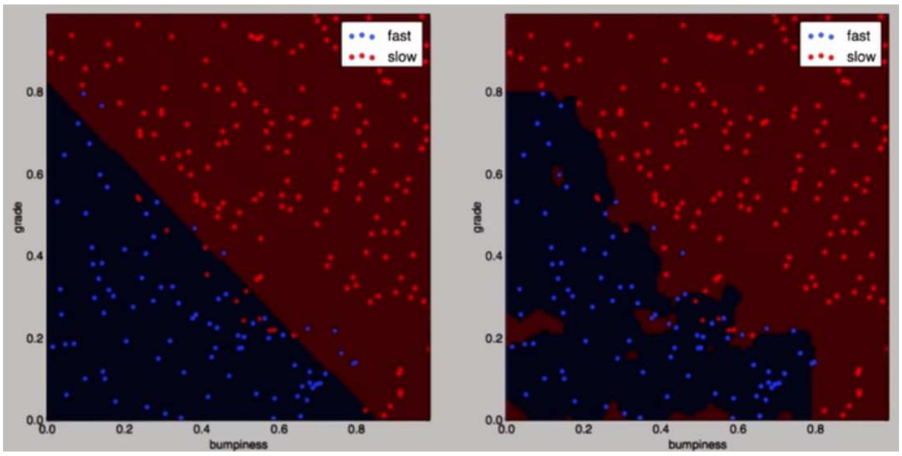


SVM (Linear)

Non-linear Classification and Kernel Trick

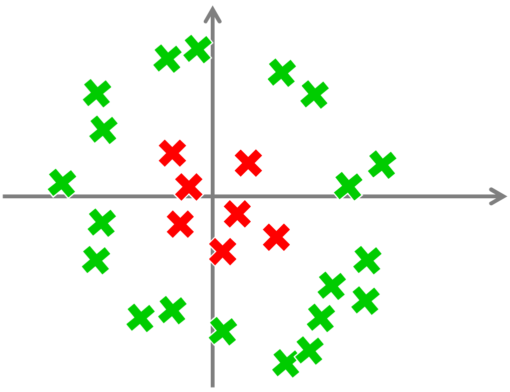
Non-linear Classification using SVM

- Both of the two shown classifications have been done using SVM!
- SVM is built to use lines (hyperplanes) for classification. So how can it build non-linear class boundaries?



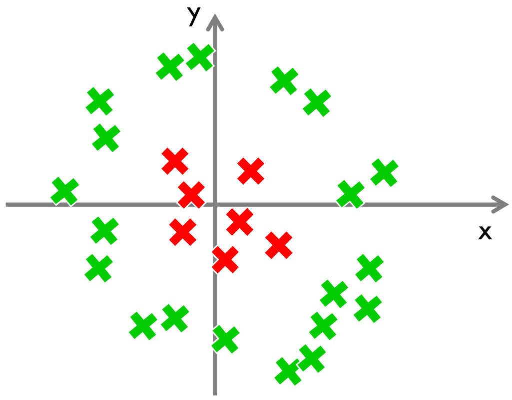
Classifying with Non-liner Boundaries

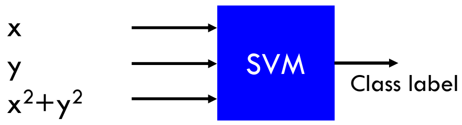
- Do you think SVM can separate the two classes with a line/hyperplane?
- Based on what we learned, there is no line or hyperplane that can do the separation.
- But we will learn a trick that will make it possible to separate these data using a line or hyperplane…



SVM Trick

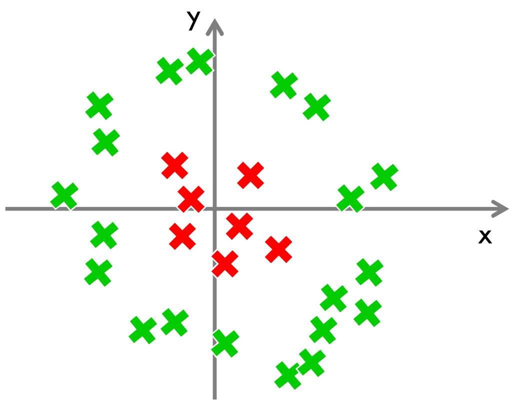
- So far we learned that SVM is a box that we feed x and y features of each data and it predicts the class…
- What if we provide a third feature x^2+y^2 (which is basically computable from x and y i.e. a transform of them, and therefore not a new feature)? Does SVM work now?

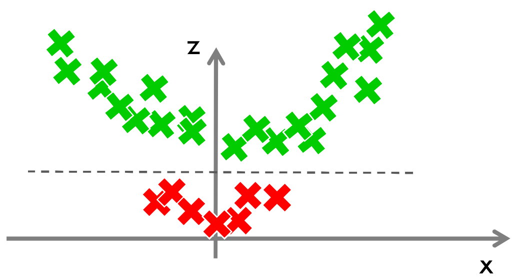


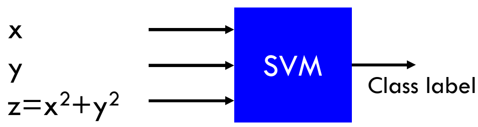


SVM Trick

- Let's call the new feature z. Now we have 3 features and therefore a 3 dimensional space.
- Let's draw the X-Z plane. Notice that z values cannot be negative … values near origin have small z …
- Can we now draw a hyperplane to separate the lines?

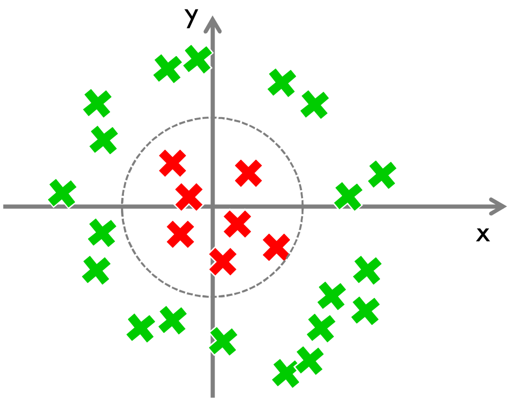


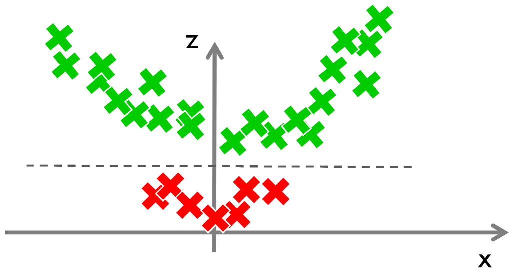




SVM Trick

- The line (or hyperplane) in the right hand side graph is equal to a circle (map to a circle) in the first graph (a plane cutting a cone in the middle)
- So by adding the new feature we could use the SVM to classify the data using a hyper plane.

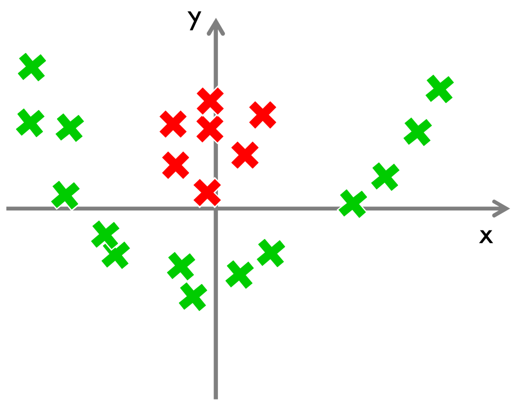




SVM Trick

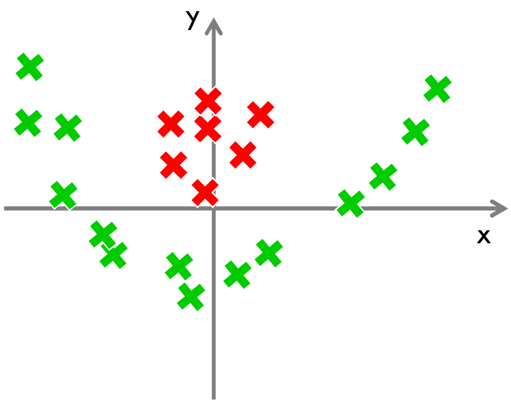
- How about the following graph? Can it be transformed into another form to make classifying it using a line or hyper plane possible?
- Which of the additional features might help?

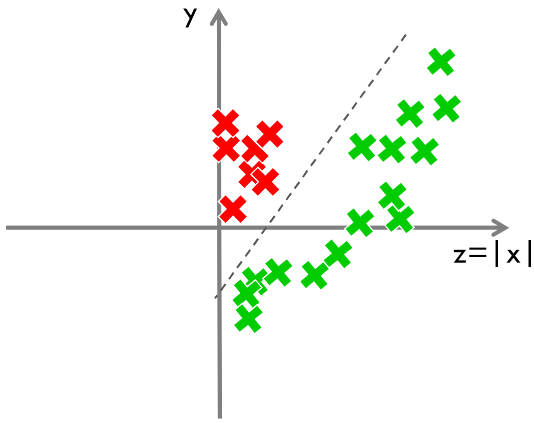
  - $z = x^2 + y^2$
  - $z = | x |$
  - $z = | y |$



SVM Trick

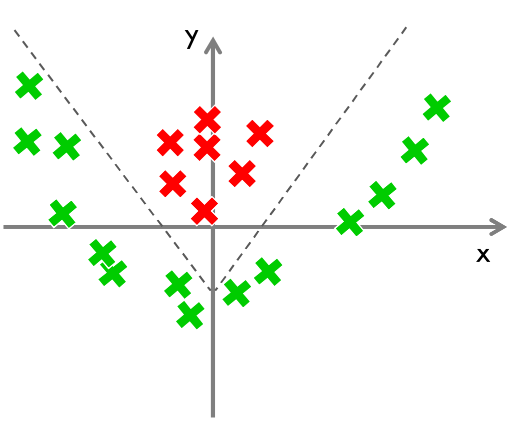
- The answer is: z= | x |
- Note that this time we replaced a feature with another feature (that is computable from the initial features). The transform will move the points in the negative side of x to its positive side.

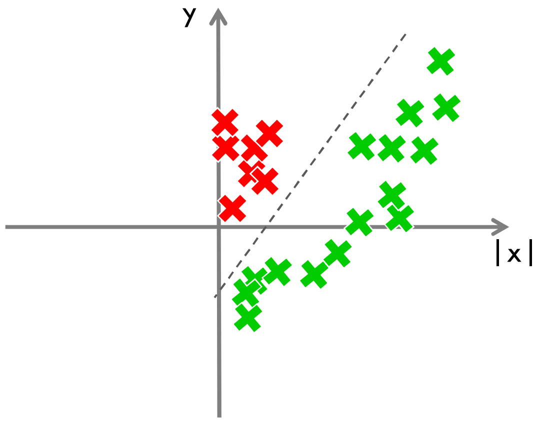




SVM Trick

- If you ask how would the line be in the original space, then it will be represented with two lines (i.e. a non-linear decision surface).





Kernel Functions

- So, the question is that, should we add new features to make the problem classifiable using SVM?
- The interesting thing about support vector machine is that it can use a function called Kernel function to do that for us…
- These are functions that take a low dimensional space (like 2 features x and y) and map it to a very high dimensional space…

(x,y) → (like $x_1,x_2,x_3,x_4, \ldots, x_n$)

Not separable   Separable

- We then classify the data in the new space and if needed bring back the results to original space.
- The plane which separates the classes in the high dimensional space, will actually look non-linear in the original space.

Kernel Functions

- There are several Kernel functions available…
  - Linear
  - RBF
  - Polynomial
  - Sigmoid
  - Pre-computed
- Some libraries will use linear or RBF by default…(a well tuned RBF is always better than linear for example)

Some rules of thumb:

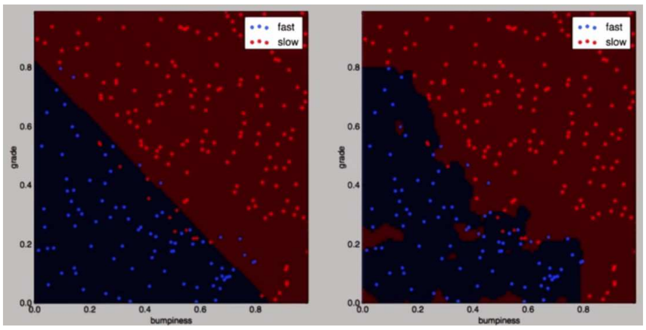
- Use linear kernel when number of features is larger than number of observations.
- Use gaussian kernel when number of observations is larger than number of features.
- If number of observations is larger than 50,000 speed could be an issue when using gaussian kernel; hence, one might want to use linear kernel.

Adjusting Parameters

Kernel and SVM Parameters

- In machine learning, the phrase "Parameters" refers to the settings we use when creating a classifier. Each type of classifier will obviously have different parameters.
- In SVM classification we have three important parameters, namely Kernel type, C and Gama (different Kernels might have additional parameters, e.g. Polynomial Kernel needs a parameter called order).
- That's what causes the SVM to classify below data differently… (Gamma = 1000 in RBF, Otherwise for very small Gamma they would look almost similar)

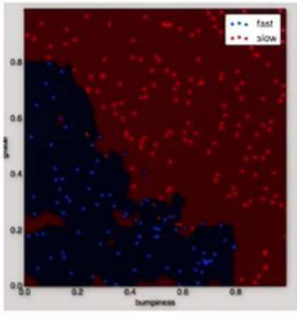
Linear Kernel RBF Kernel



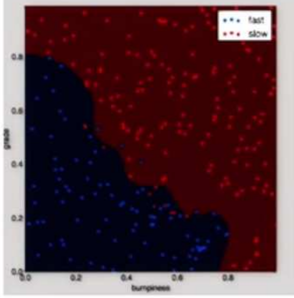
Kernel and SVM Parameters

- Guess which one is classified with a linear kernel. Which one with RBF with high Gamma, and which one with RBF with low Gamma?
- The linear Kernel will only give you a linear boundary (in the original space).

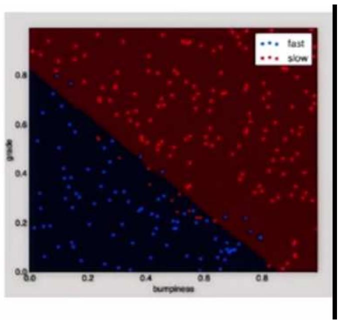
RBF with high Gamma



RBF Kernel with low Gamma

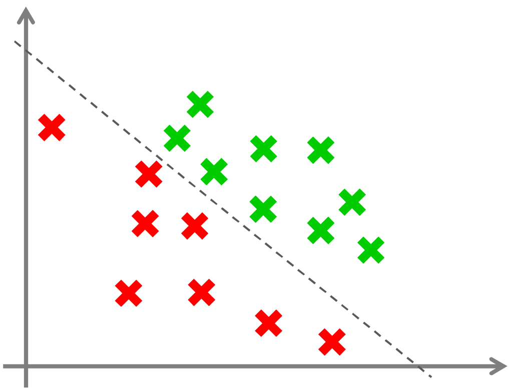


Linear



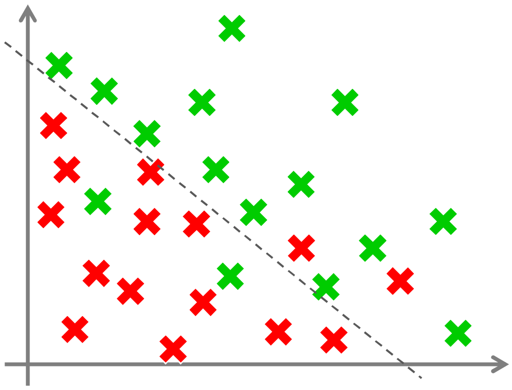
Over fitting

- Even though the unsmooth line correctly classifies the existing (training) data, it is erratic!
- It does not generalize (expand to the whole data) well.
- A straight line could classify the same data and at the same time provide a better generalization.
- All three parameters of SVM can affect over fitting (Kernel, C and Gamma).



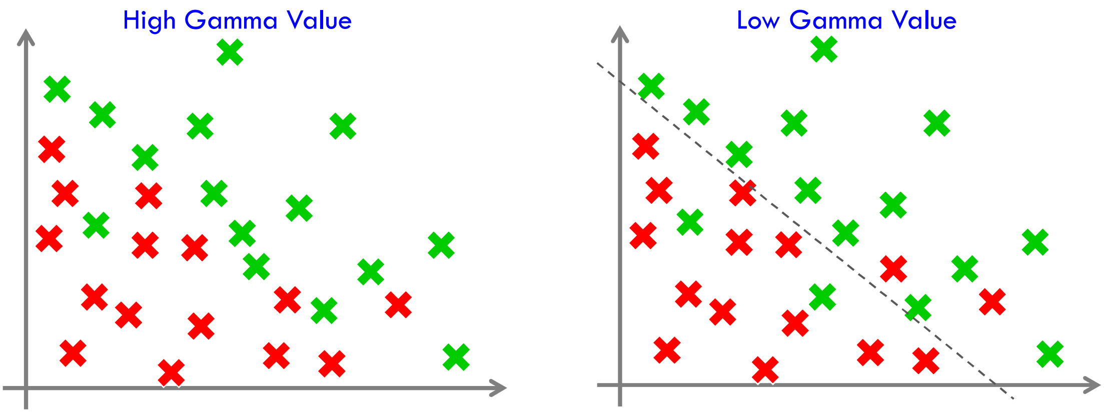
C Parameter in SVM

- C Parameter controls the trade off between having a Smooth boundary and having lower error (i.e. classifying points correctly)
- If you increase the C parameter, you will get a more accurate boundary (and therefore more correct answers for training stage), but you may risk having an over-fitted classifier.
- This parameter will make big difference, specially in RBF kernel.



ɣ (gamma) Parameter in SVM

- Gamma parameter in SVM how far from the boundary will SVM look into the data for the calculation and adjustment of the boundary.
- Higher Gamma value will cause the nearer values to have higher impact and therefore more intricate fit to the near points. Therefore, very high values might cause over fitting again.
- Small values will result in a more straight, less detailed and less jagged fit.



Adjusting the Parameters

- You will normally play with the parameters and find a setting which works better.
- For example in below tables, the results are checked for different Kernels, C and Gamma values.

Gamma	Train Error	Test Error	Training Time (s)
0.145	0.1719	0.1669	58.25
0.250	0.1668	0.1735	75.46
0.500	0.1636	0.1695	77.11
1.000	0.1631	0.1632	83.05
1.500	0.1598	0.1737	69.39
2.000	0.1602	0.1671	71.75
2.500	0.1604	0.1633	75.61
3.000	0.1616	0.1581	80.03
3.500	0.1602	0.1637	67.14
4.000	0.1592	0.1615	68.91
4.500	0.1587	0.1674	77.01
5.000	0.1591	0.1602	58.25

C	Training Error	Test Error	Training Time (s)
1	0.16342	0.16454	28.3
5	0.16116	0.16644	50.8
7.5	0.15884	0.16798	63.7
9	0.15976	0.16776	71.3
10	0.16160	0.15812	80.0
12.5	0.15866	0.17088	78.9
15	0.15870	0.16726	95.7
20	0.15766	0.16128	99.7
30	0.16021	0.16124	115

Kernel	Parameters	Train Error	Test Error	Train Time
Gaussian RBF Kernel	Gamma = 3, C = 10	0.16160	0.15812	80.0
Polynomial	Degree = 1	0.21780	0.21736	125.6
Polynomial	Degree = 2	0.17086	0.17466	391.1
Polynomial	Degree = 3	0.16536	0.16436	3247.0
Linear	-	0.21420	0.21136	60.5

Conclusion

SVM Usability

- They perform very well in complicated domains.
- SVM might not perform well in very large data sets, because the training algorithm has a complexity of worse than $O(n^3)$. So it becomes too slow with large data.
- If there are too many features, SVM might again become too slow.
- They also don't work very well when there is too much noise in the data (Naïve Bayes might work better).
- So selection of the method depends on the size and type of your data.

SVM using Scikit Learn

Using SkLearn – Program 1 (Data from Array)

```
import numpy as np

features_train = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1],[3,3]])
labels_train   = np.array([1, 1, 1, 2, 2, 2])
features_test  = np.array([[-2, -2], [-2, -3], [2, 3], [1, 2]])
labels_test    = np.array([1, 1, 1, 2])

from sklearn import svm

#clf = svm.SVC()
#clf = svm.SVC(kernel='linear',C=1.0, gamma=0.1)
#clf = svm.SVC(kernel='poly', degree=3,C=1.0, gamma=0.1)
clf = svm.SVC(kernel='rbf',C=1.0, gamma=0.1)
clf.fit(features_train, labels_train)
pred = clf.predict(features_test)

print "Test labels:      ", labels_test
print "Predicted labels: ", pred

from sklearn.metrics import accuracy_score
print "Accuracy:         ", accuracy_score(pred, labels_test)

print "\nPredicted label for ", [-0.8, -1] ," is ", (clf.predict([[-0.8, -1]]))
```

Using SkLearn – Program 2 (Data from file)

```
import numpy as np

def buildData(filename, featureCols, testRatio):
    f = open(filename)
    data = np.loadtxt(fname = f, delimiter = ',')
    np.random.shuffle(data)    # randomize the order

    X = data[:, :featureCols]  # select columns 0:featureCols-1
    y = data[:, featureCols]   # select column  featureCols

    n_points = y.size
    print "Imported " + str(n_points) + " lines."

    ### split into train/test sets
    split = int((1-testRatio) * n_points)
    X_train = X[0:split,:]
    X_test  = X[split:,:]
    y_train = y[0:split]
    y_test  = y[split:]

    return X_train, y_train, X_test, y_test
```

Using SkLearn – Program 2 (Data from file)

```
def buildClassifier(features_train, labels_train):
    from sklearn import svm

    #clf = svm.SVC(kernel='linear',C=1.0, gamma=0.1)
    #clf = svm.SVC(kernel='poly', degree=3,C=1.0, gamma=0.1)
    clf = svm.SVC(kernel='rbf',C=1.0, gamma=0.1)
    clf.fit(features_train, labels_train)
    return clf

def checkAccuracy(clf, features, labels):
    from sklearn.metrics import accuracy_score

    pred = clf.predict(features)
    accuracy = accuracy_score(pred, labels)
    return accuracy
```

Using SkLearn – Program 2 (Data from file)

```
features_train, labels_train, features_test, labels_test = buildData("car-eval-data-1.csv", 6, 0.5)
clf           = buildClassifier(features_train, labels_train)
trainAccuracy = checkAccuracy(clf, features_train, labels_train)
testAccuracy  = checkAccuracy(clf, features_test, labels_test)
print "Training Items: "     + str(labels_train.size) + ", Test Items: " + str(labels_test.size)
print "Training Accuracy: "  + str(trainAccuracy)
print "Test Accuracy: "      + str(testAccuracy)

i = 0
while i < labels_test.size:
  pred = clf.predict(features_test[i])
  marker = " "
  if labels_test[i] <> pred:
    marker = " *"
  print "F(" + str(i) + ") : " + str(features_test[i]) + " label= " + str(labels_test[i]) + " \\   pred= "
+ str(pred) + marker;
  i = i + 1
```

Using SkLearn - Output

```
Imported 1728 lines.
Training Items: 1209, Test Items: 519
Training Accuracy: 0.954507857734
Test Accuracy: 0.947976878613

F(0) : [ 1.  2.  2.  2.  2.  2.] label= 0.0 pred= [ 0.]
F(1) : [ 1.  2.  2.  2.  2.  0.] label= 0.0 pred= [ 0.]
F(2) : [ 1.  2.  2.  2.  2.  1.] label= 0.0 pred= [ 0.]
F(3) : [ 1.  2.  2.  2.  1.  2.] label= 0.0 pred= [ 0.]
...
F(14) : [ 1.  2.  2.  4.  1.  1.] label= 0.0 pred= [ 1.] *
F(15) : [ 1.  2.  2.  4.  0.  2.] label= 1.0 pred= [ 1.]
F(16) : [ 1.  2.  2.  4.  0.  0.] label= 0.0 pred= [ 0.]
F(17) : [ 1.  2.  2.  4.  0.  1.] label= 0.0 pred= [ 0.]
F(18) : [ 1.  2.  2.  5.  2.  2.] label= 1.0 pred= [ 1.]
F(19) : [ 1.  2.  2.  5.  2.  0.] label= 0.0 pred= [ 0.]
...
F(23) : [ 1.  2.  2.  5.  1.  1.] label= 0.0 pred= [ 1.] *
F(24) : [ 1.  2.  2.  5.  0.  2.] label= 0.0 pred= [ 1.] *
F(25) : [ 1.  2.  2.  5.  0.  0.] label= 0.0 pred= [ 0.]
F(26) : [ 1.  2.  2.  5.  0.  1.] label= 0.0 pred= [ 0.]
F(27) : [ 1.  2.  3.  2.  2.  2.] label= 0.0 pred= [ 0.]
```