Machine Learning

| SVC Error Function

Prof. Dr. Mostafa Elhosseini

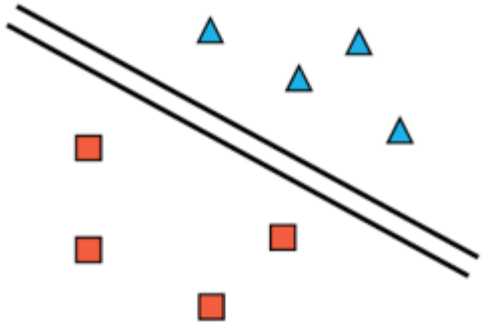
https://youtube.com/ElhosseiniAcademy

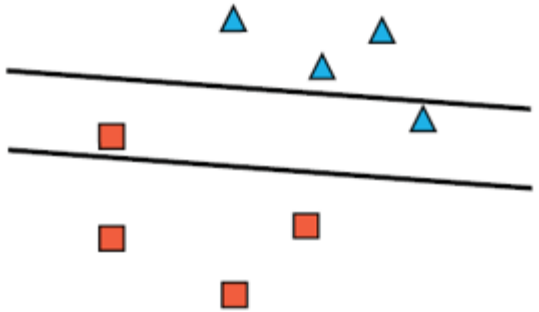
Agenda

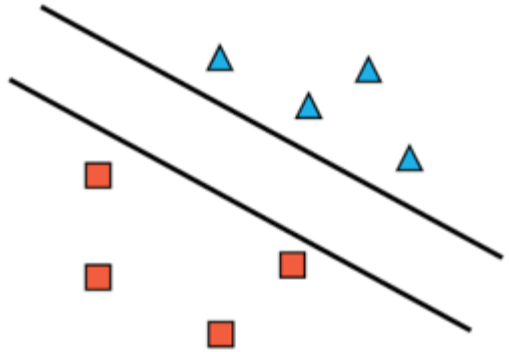
- Classification error function
- Distance error function
- C parameter
- Coding SVC

Error Function

- What do we want the model to achieve?
  - Each of the two lines should classify the points as best as possible.
  - The two lines should be as far away from each other as possible.

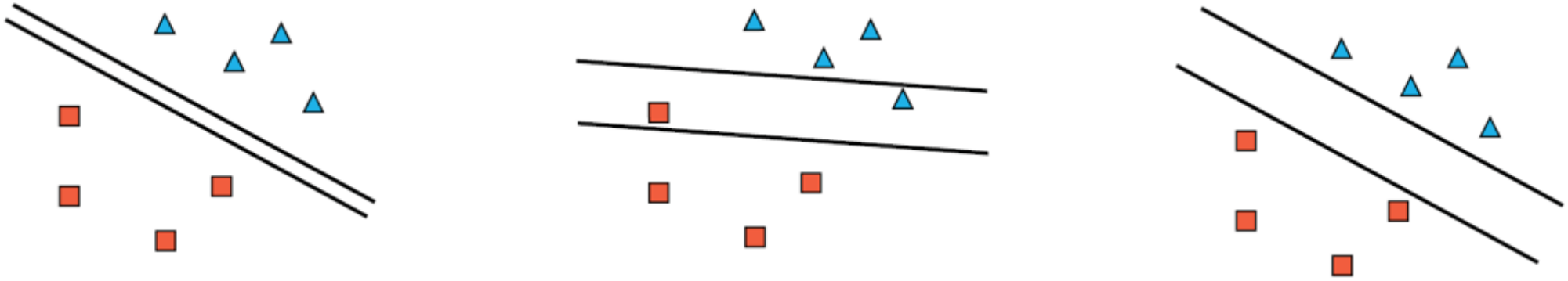






Error Function

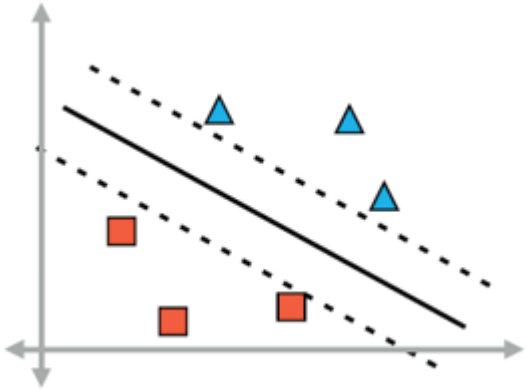
- It tries to maximize two things at the same time: the classification of the points and the distance between the lines.



- The error function should penalize any model that doesn't achieve these things
- Error = Classification Error + Distance Error

Classification error function

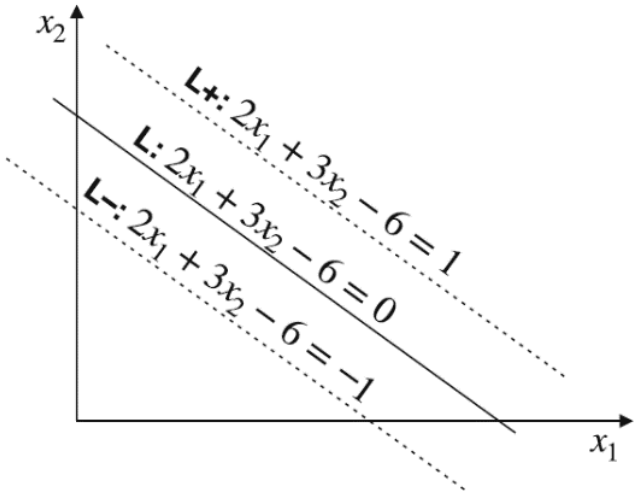
- Trying to classify the points correctly
- Because the classifier is formed by two lines, we think of them as two separate discrete perceptrons.



- We then calculate the total error of this classifier as the sum of the two perceptron errors

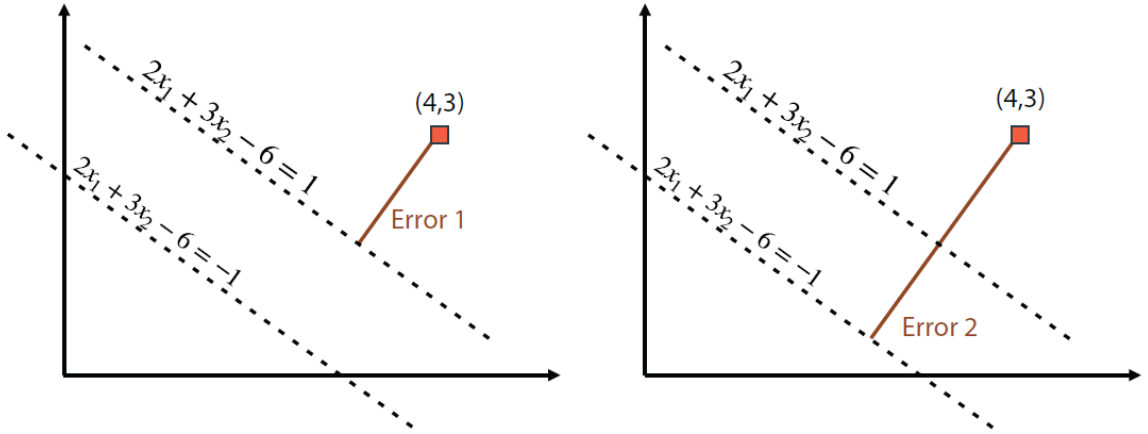
Classification error function

- The SVM uses two parallel lines, and luckily, parallel lines have similar equations; they have the same weights but a different bias.
  - $L+: \ w_1x_1 + w_2x_2 + b = 1$
  - $L-: \ w_1x_1 + w_2x_2 + b = -1$
- Example
  - $L: \ 2x_1 + 3x_2 - 6 = 0$
  - $L+: \ 2x_1 + 3x_2 - 6 = 1$
  - $L-: \ 2x_1 + 3x_2 - 6 = -1$



Classification error function

- Our classifier now consists of the lines $L+$ and $L-$. We can think of $L+$ and $L-$ as two independent perceptron classifiers, and each of them has the same goal of classifying the points correctly.



Error = Error 1 + Error 2

  - Notice that in an SVM, both lines have to classify the points well.

Classification error function

- The error function for the discrete perceptron with prediction $\hat{y} = step(w_1 x_1 + w_2 x_2 + b)$ at the point (p, q) is given by the following:
  - 0 if the point is correctly classified, and
  - $|\boldsymbol{w_1 x_1 + w_2 x_2 + b}|$ if the point is incorrectly classified

Classification error function

Example

- Consider the point (4,3) with a label of 0. This point is incorrectly classified by both of the perceptrons.

- $L+$: $\hat{y} = step(2x_1 + 3x_2 - 7)$
- $L-$: $\hat{y} = step(2x_1 + 3x_2 - 5)$

- Classification error: $|2 \times 4 + 3 \times 3 - 7| + |2 \times 4 + 3 \times 3 - 5| = 10 + 12 = 22$

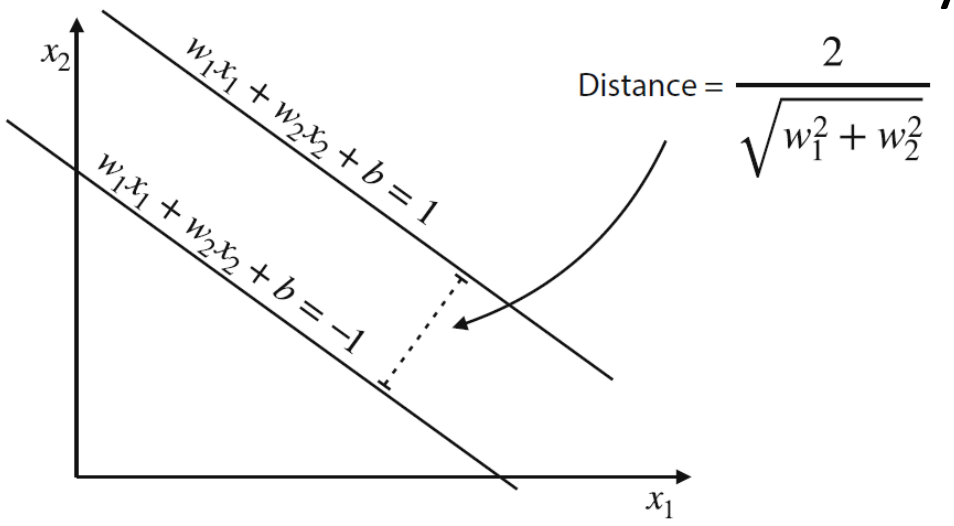
Distance error function

- Trying to separate our two lines as far apart as possible
- Simple error function that is large when the two lines are close and small when they are far
- This error function is called the distance error function

- if our lines have equations $w_1x_1 + w_2x_2 + b = 1$ and $w_1x_1 + w_2x_2 + b = -1$, then the error function is $\boldsymbol{{w_1}^2 + {w_2}^2}$
- Why? — See Appendix 01

Distance error function

- The perpendicular distance between the two lines is precisely: $\frac{2}{\sqrt{{w_1}^2+{w_2}^2}}$

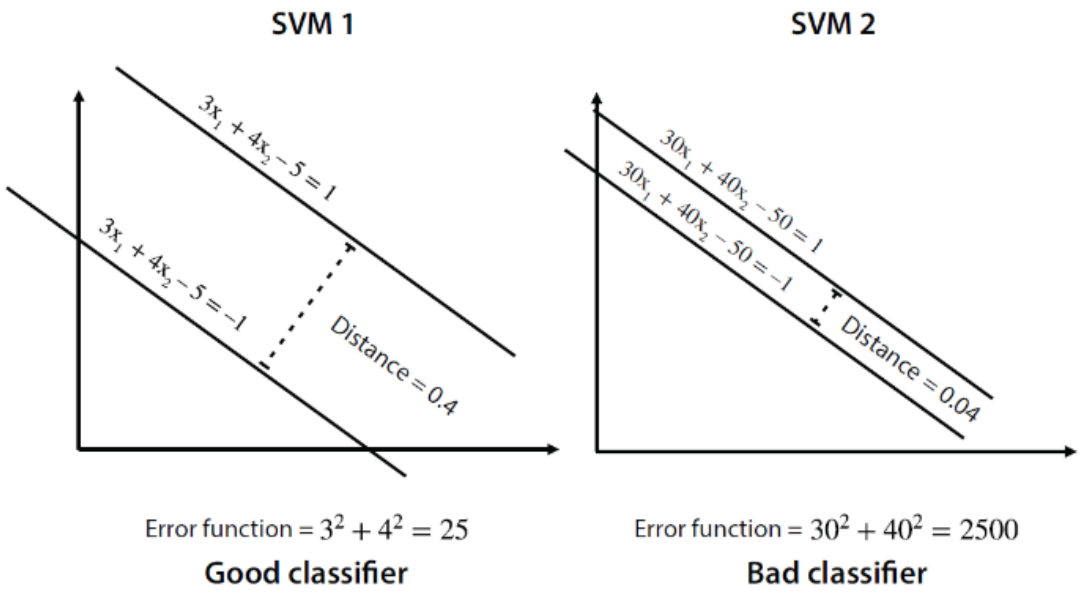


- Knowing this, notice the following:
  - When ${w_1}^2 + {w_2}^2$ is large, d is small
  - When ${w_1}^2 + {w_2}^2$ is small, d is large
- ${w_1}^2 + {w_2}^2$ is a good error function, as it gives us large values for the bad classifiers and small values for the good classifiers

Distance error function

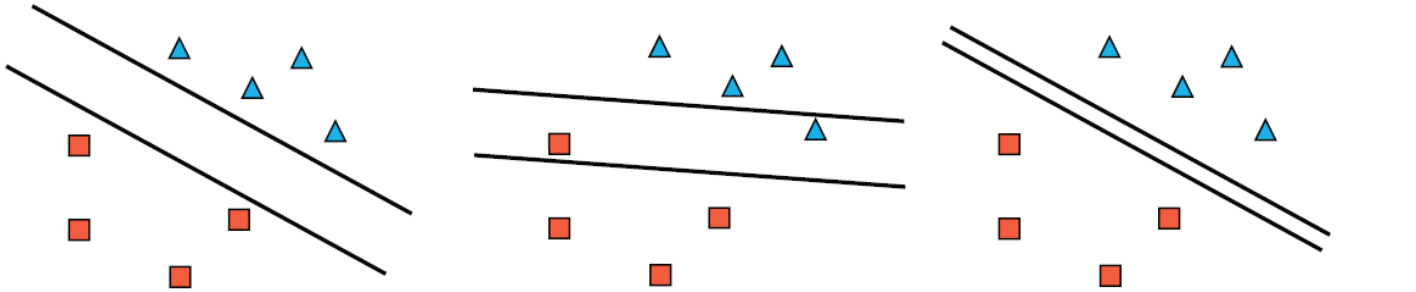
Example

- SVM 1:
  - $L+: \ 3x_1 + 4x_2 + 5 = 1$
  - $L-: \ 3x_1 + 4x_2 + 5 = -1$
  - Distance error function $= 3^2 + 4^2 = 25$
- SVM 2:
  - $L+: \ 30x_1 + 40x_2 + 50 = 1$
  - $L-: \ 30x_1 + 40x_2 + 50 = -1$
  - Distance error function $= 30^2 + 40^2 = 2500$
- SVM 1 a much better classifier



Error formula

- Error = Classification Error + Distance Error
- A good SVM that minimizes this error function must then try to make as few classification errors as possible, while simultaneously trying to keep the lines as far apart as possible.



Good classifier

Bad classifier
(makes errors)

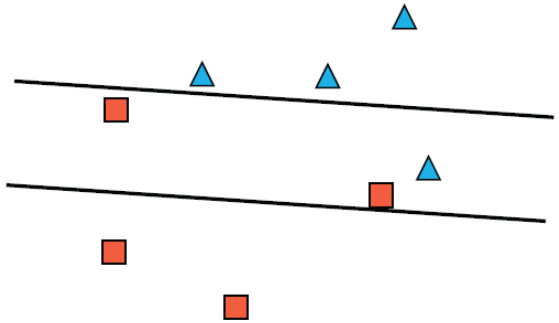
Bad classifier
(lines are too close together)

Error formula

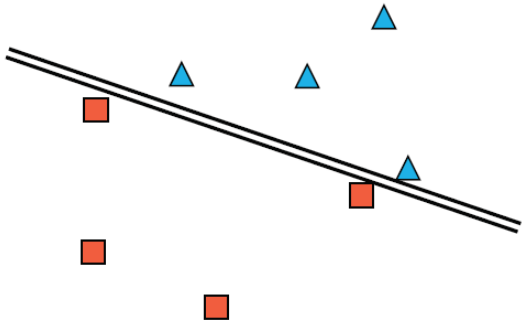
- Do we want our SVM to focus more on classification or distance?
- The C parameter is used in cases where we want to train an SVM that pays more attention to classification than to distance (or the other way around).
- We want to make sure the classifier makes as few errors as possible, while keeping the lines as far apart as possible.

Error formula

- But what if we have to sacrifice one for the benefit of the other?
- Which one should we prefer?



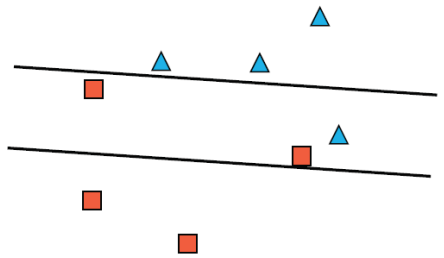
Lines are far apart (good).
Makes errors (bad).



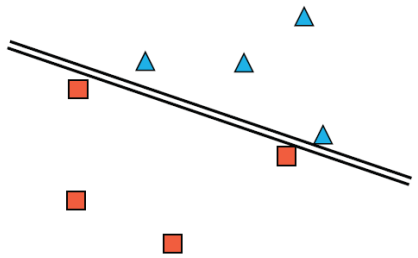
Lines are too close (bad).
Doesn't make errors (good).

Error formula

- It turns out that the answer for this depends on the problem we are solving. Sometimes we want a classifier that makes as few errors as possible, even if the lines are too close, and sometimes we want a classifier that keeps the lines apart, even if it makes a few errors. How do we control this?
- We use a parameter which we call the C parameter.



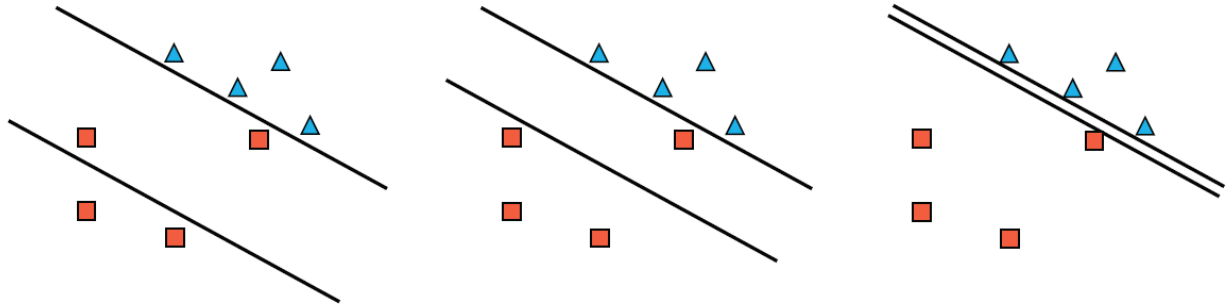
Lines are far apart (good).
Makes errors (bad).



Lines are too close (bad).
Doesn't make errors (good).

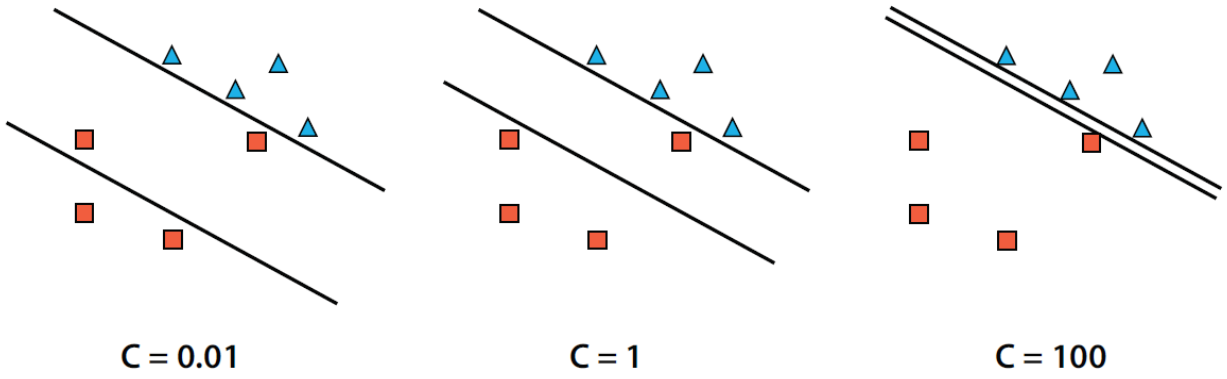
C parameter

- Error formula = $C \cdot$ (Classification Error) + (Distance Error)
- If C is large, then the error formula is dominated by the classification error, so our classifier focuses more on classifying the points correctly. If C is small, then the formula is dominated by the distance error, so our classifier focuses more on keeping the lines far apart.



C parameter

- As you can see, reducing C makes the street larger, but it also leads to more margin violations. In other words, reducing C results in more instances supporting the street, so there's less risk of overfitting. But if you reduce it too much, then the model ends up underfitting

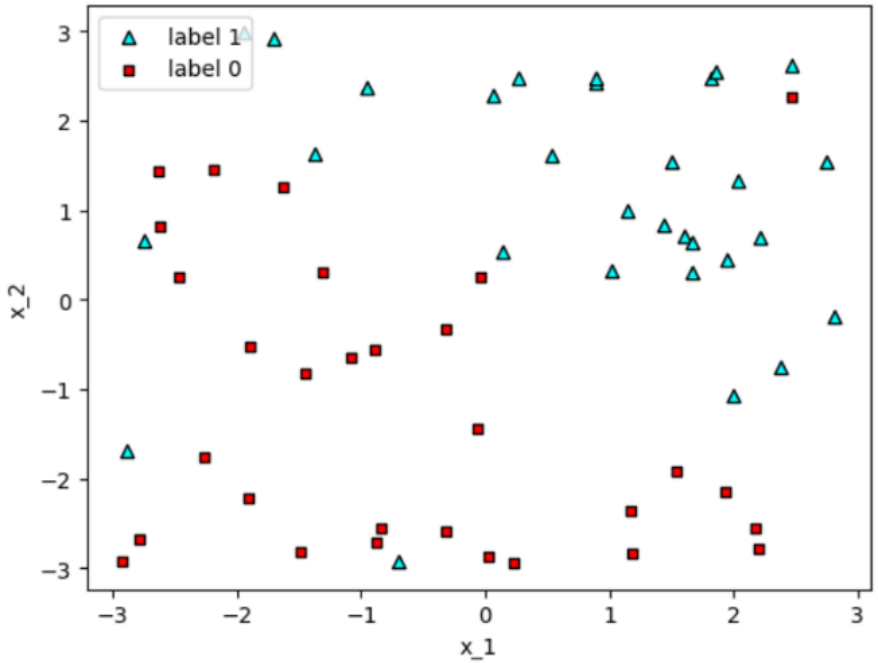


Coding SVC

- Example 01 - Grokking

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import utils
from sklearn.svm import SVC
```

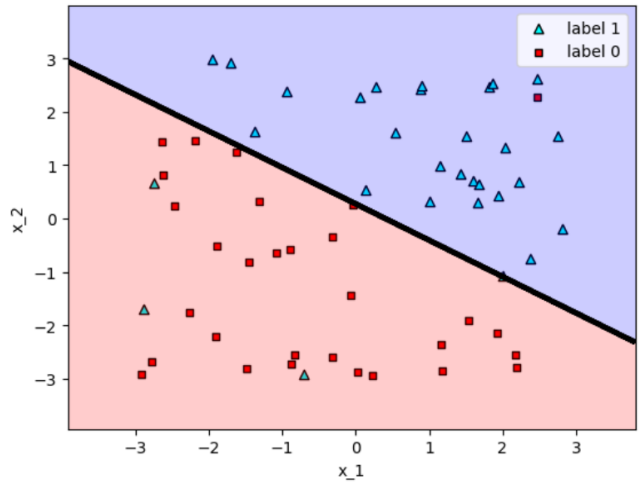
```
# Loading the linear dataset

linear_data = pd.read_csv('linear.csv')
features = np.array(linear_data[['x_1', 'x_2']])
labels = np.array(linear_data['y'])
utils.plot_points(features, labels)
```



Coding SVC

```
svm_linear = SVC(kernel='linear')
svm_linear.fit(features, labels)
print("Accuracy:", svm_linear.score(features, labels))
utils.plot_model(features, labels, svm_linear)
```

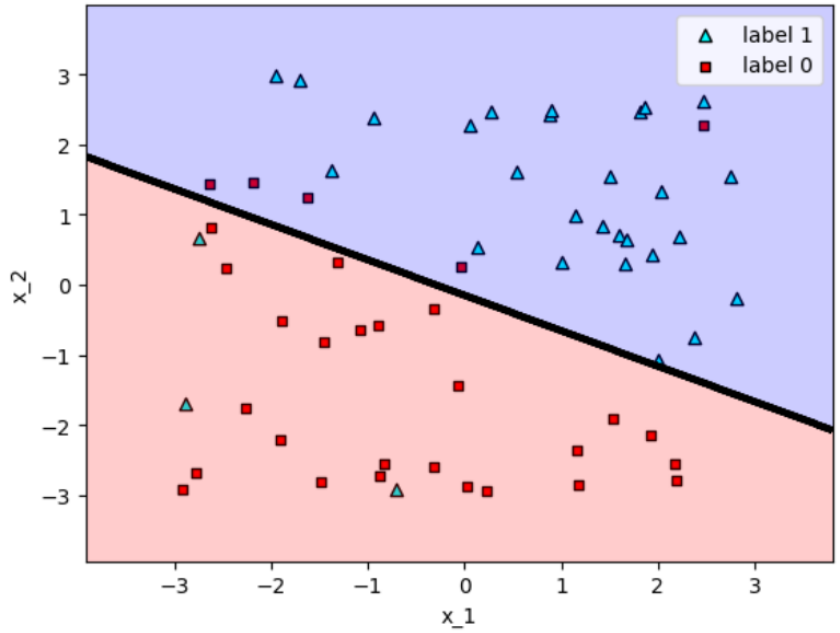
```
Accuracy: 0.9333333333333333
```



Coding SVC

```
# C = 0.01
svm_c_001 = SVC(kernel='linear', C=0.01)
svm_c_001.fit(features, labels)
print("C = 0.01")
print("Accuracy:", svm_c_001.score(features, labels))
utils.plot_model(features, labels, svm_c_001)

# C = 100
svm_c_100 = SVC(kernel='linear', C=100)
svm_c_100.fit(features, labels)
print("C = 100")
print("Accuracy:", svm_c_100.score(features, labels))
utils.plot_model(features, labels, svm_c_100)
```

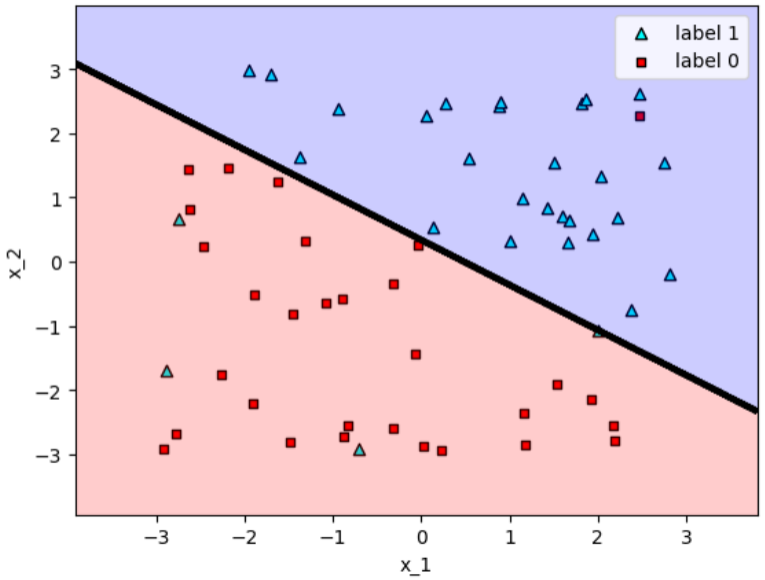
```
C = 0.01
Accuracy: 0.8666666666666667
```



Coding SVC

```
# C = 0.01
svm_c_001 = SVC(kernel='linear', C=0.01)
svm_c_001.fit(features, labels)
print("C = 0.01")
print("Accuracy:", svm_c_001.score(features, labels))
utils.plot_model(features, labels, svm_c_001)

# C = 100
svm_c_100 = SVC(kernel='linear', C=100)
svm_c_100.fit(features, labels)
print("C = 100")
print("Accuracy:", svm_c_100.score(features, labels))
utils.plot_model(features, labels, svm_c_100)
```

```
C = 100
Accuracy: 0.9166666666666666
```



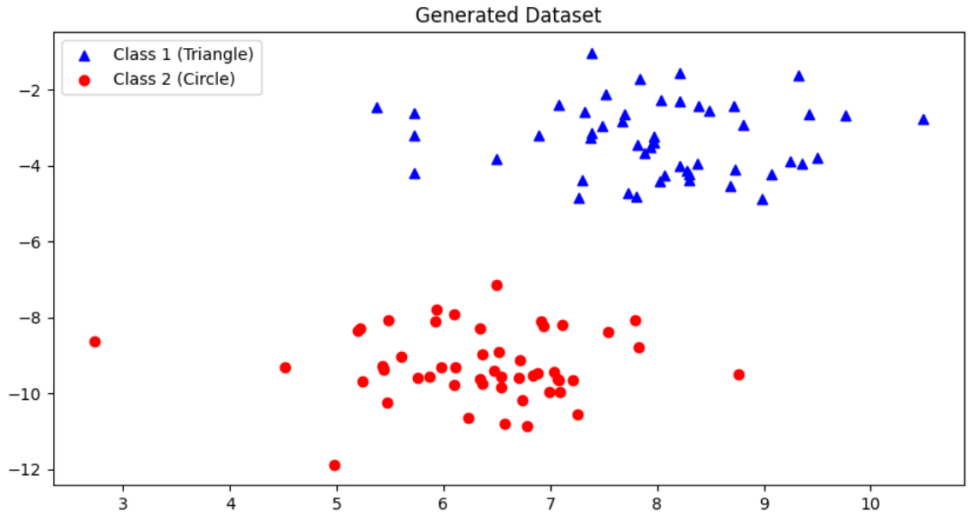
Coding SVC

- Building an SVM to separate a linear dataset

```
import pandas as pd
from sklearn.svm import SVC
from plot import plot_dataset, plot_decision_boundary
```

```
df = pd.read_csv('binary_classification_dataset.csv')
X = df[['Feature1', 'Feature2']].values
y = df['Label'].values
```

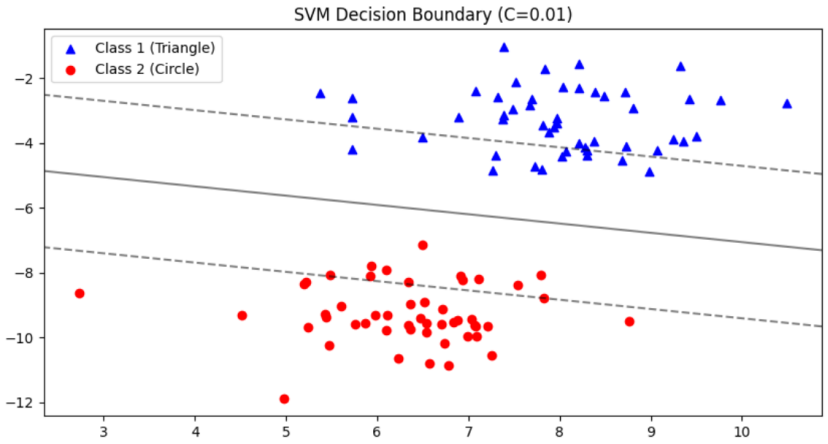
```
plot_dataset(X, y)
```



Coding SVC

- Building an SVM to separate a linear dataset

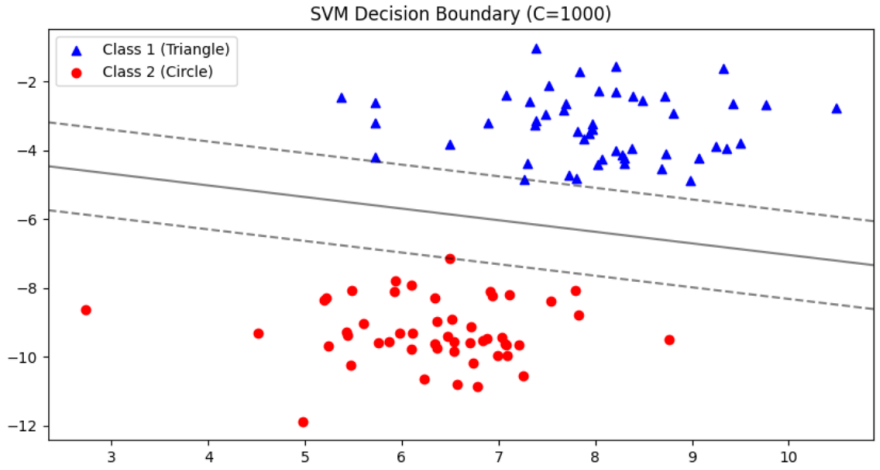
```
clf_small_C = SVC(kernel='linear', C=0.01)
clf_small_C.fit(X, y)
plot_decision_boundary(clf_small_C, X, y, title="SVM Decision Boundary (C=0.01)")
```



Coding SVC

- Building an SVM to separate a linear dataset

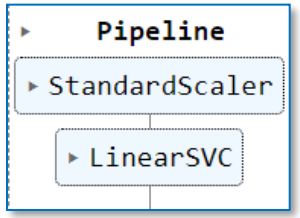
```
clf_large_C = SVC(kernel='linear', C=1000)
clf_large_C.fit(X, y)
plot_decision_boundary(clf_large_C, X, y, title="SVM Decision Boundary (C=1000)")
```



Coding SVC

- IRIS Dataset

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2)  # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
                        LinearSVC(C=1, dual=True, random_state=42))
svm_clf.fit(X, y)
```




```
X_new = [[5.5, 1.7], [5.0, 1.5]]
svm_clf.predict(X_new)
```

```
array([ True, False])
```

```
svm_clf.decision_function(X_new)
```

```
array([ 0.66163411, -0.22036063])
```

Kernel Function

- Support vector Classifier is pretty cool, because
  - Handle outliers
  - Allow overlapping classification — because it allows misclassification

- But, what if we had tons of overlap?

- Next Lecture