support vector machines

logistic regression:

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)}, \theta))] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by

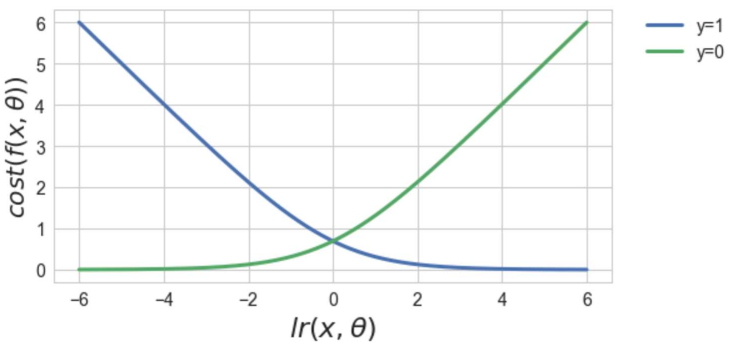
$-log(f(x^{(i)}, \theta))$.

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by

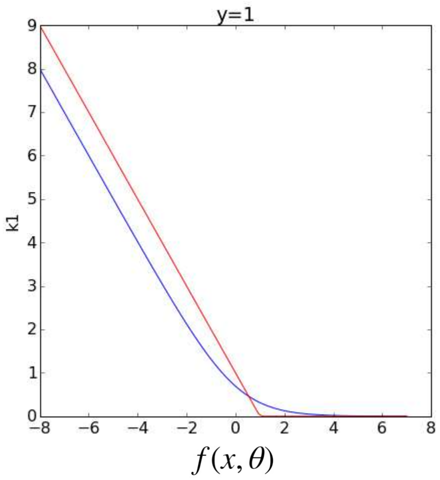
$-log(1 - f(x^{(i)}, \theta))$.

support vector machines

logistic regression:



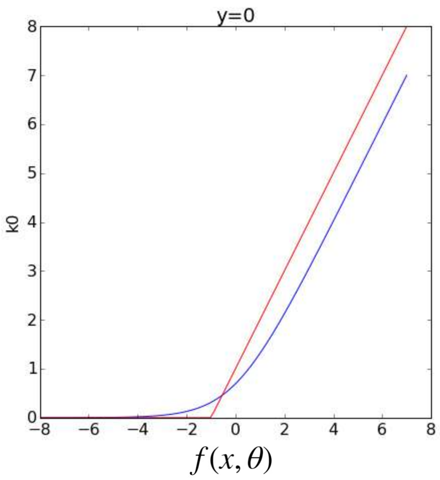
support vector machines



- replace cost function by piecewise linear function
- if *y* = 1 then the contribution to the cost is

$$k_1(f(x,\theta)) = max(0, 1 - f(x,\theta))$$

support vector machines



- replace cost function by piecewise linear function
- if *y* = 1 then the contribution to the cost is

$$k_1(f(x,\theta)) = max(0, 1 - f(x,\theta))$$

- if *y* = 0 then the contribution to the cost is

$$k_0(f(x,\theta)) = max(0, 1 + f(x,\theta))$$

support vector machines

Fit a linear model

$$f(x, \theta) = \theta' x$$

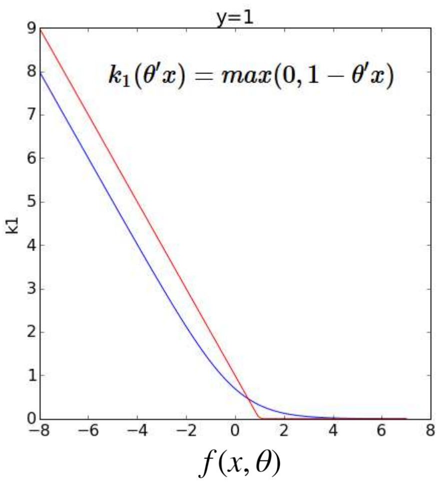
such that

$$J(\theta) = \left[C \sum_{i=1}^{n} y^{(i)} k_1(\theta' x^{(i)}) + (1 - y^{(i)}) k_0(\theta' x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

with $k_1(\theta' x) = max(0, 1 - \theta' x)$ and $k_0(\theta' x) = max(0, 1 + \theta' x)$

is minimized.

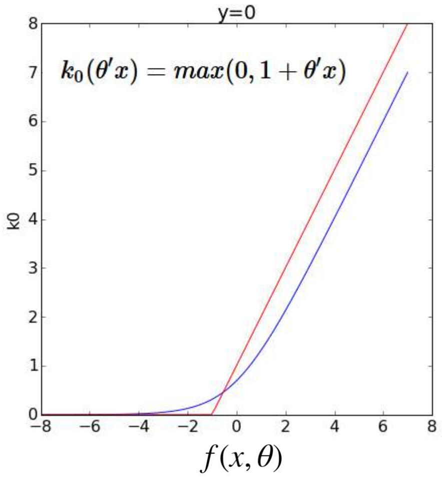
support vector machines



- In this case the contribution to the cost needs to be small when the model predicts high values (>0) and large when the model predicts low values (<0).
- For SVMs we see that the contribution to cost decreases linearly and becomes zero when

$$\theta' x \geq 1$$

support vector machines



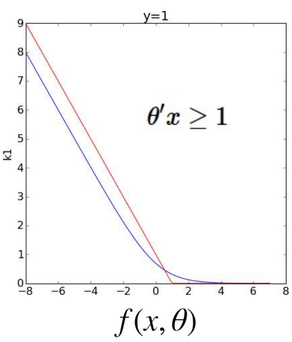
- In this case the contribution to the cost needs to be large when the model predicts high values (>0) and small when the model predicts low values (<0).
- For SVMs we see that the contribution to the cost is zero for

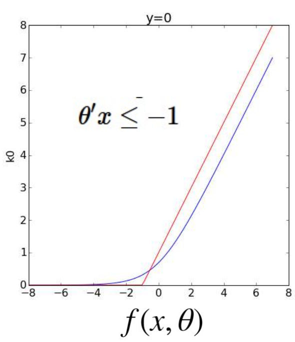
$$\theta' x \leq -1$$

and then increases linearly.

support vector machines

$$J(\theta) = \left[C \sum_{i=1}^{n} y^{(i)} k_1(\theta' x^{(i)}) + (1 - y^{(i)}) k_0(\theta' x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$



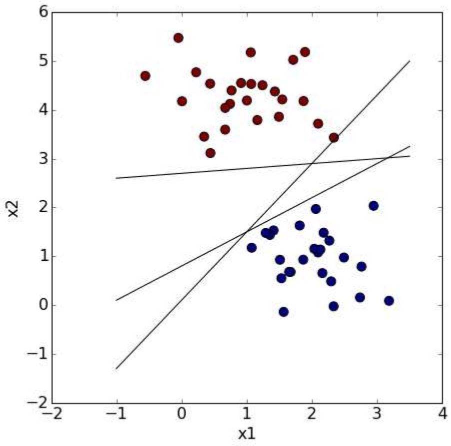


- Consider a train set with two classes that are perfectly linearly separable.
- The piecewise cost function can be made zero.
- The SVM objective can be written as

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

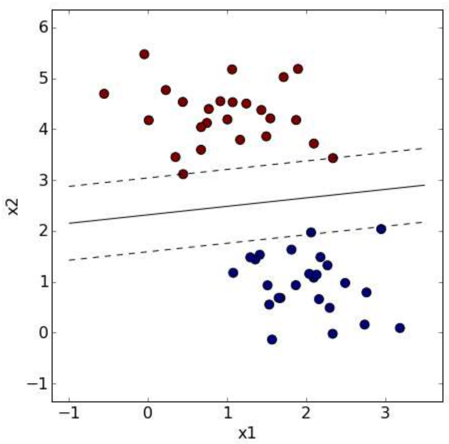
$$\text{subject to } \theta' x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$
$$\theta' x^{(i)} \leq -1 \text{ if } y^{(i)} = 0$$

support vector machines



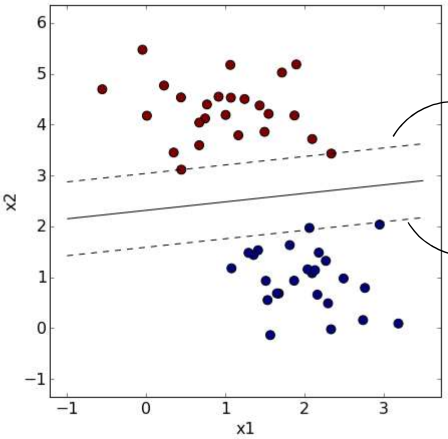
- Consider a train set with two classes that are perfectly linearly separable.
- many possible decision boundaries

support vector machines



- Consider a train set with two classes that are perfectly linearly separable.
- many possible decision boundaries
- SVM picks to one that maximizes the margin between the classes

support vector machines



$\theta' x^{(i)} = 1$

$\theta' x^{(i)} = -1$

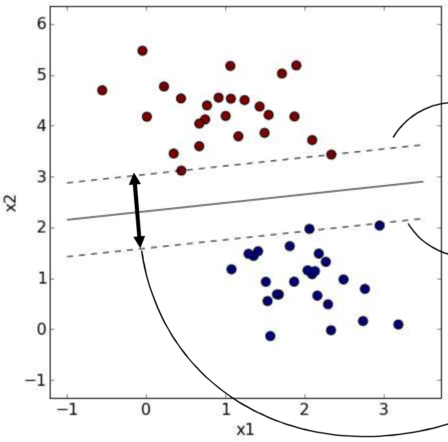
- red points statisfy $\theta' x^{(i)} \geq 1$
- blue points satisfy $\theta' x^{(i)} \leq -1$
- distance bewteen two dashed lines is

$$\frac{2}{\sqrt{\sum_{j=1}^{m} \theta_j^2}}$$

- SVM objective was

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

support vector machines



$\theta' x^{(i)} = 1$

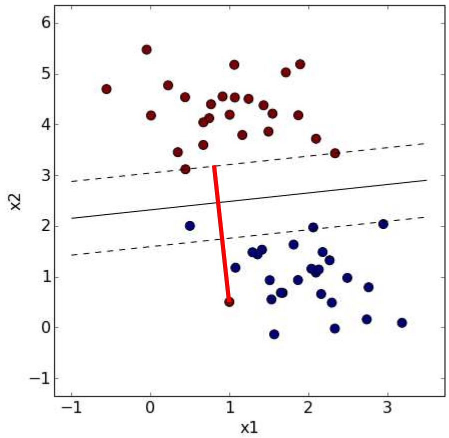
$\theta' x^{(i)} = -1$

margin

- red points statisfy $\theta' x^{(i)} \geq 1$
- blue points satisfy $\theta' x^{(i)} \leq -1$
- distance bewteen two dashed lines is

$$\frac{2}{\sqrt{\sum_{j=1}^{m} \theta_j^2}}$$

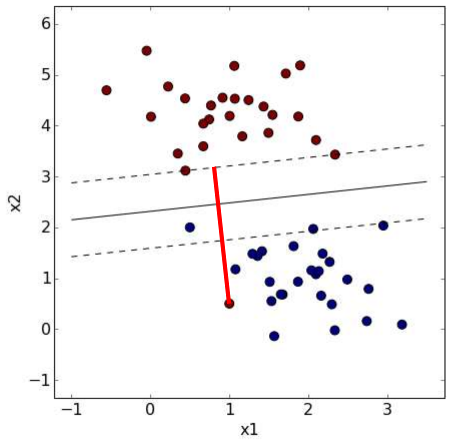
support vector machines



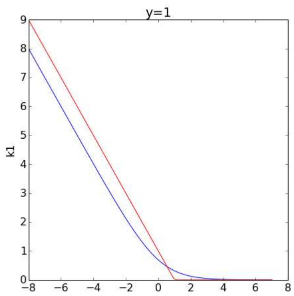
- no model parameters that satisfy

$$\theta' x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$
$$\theta' x^{(i)} \leq -1 \text{ if } y^{(i)} = 0$$

support vector machines

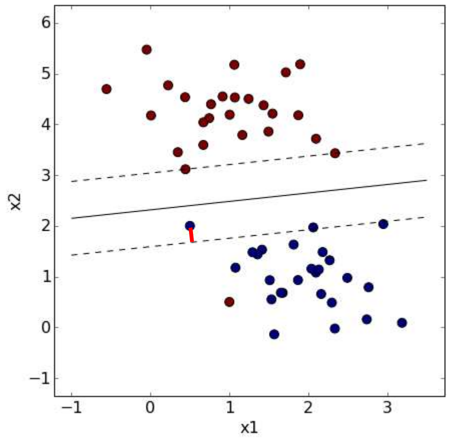


- For misclassified red points (y=1) the contribution to the cost increases linearly with the distance from the upper dashed line.

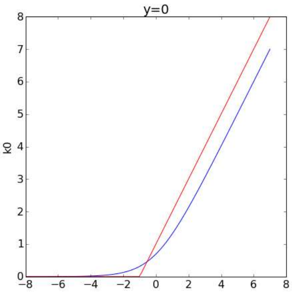


$f(x, \theta)$

support vector machines

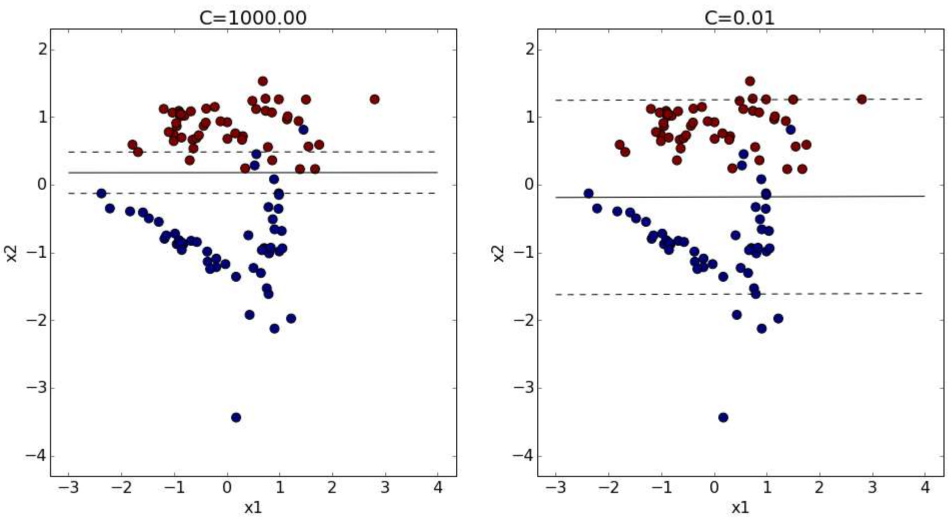


- For misclassified blue points (y=0) the contribution to the cost increases linearly with the distance from the lower dashed line.

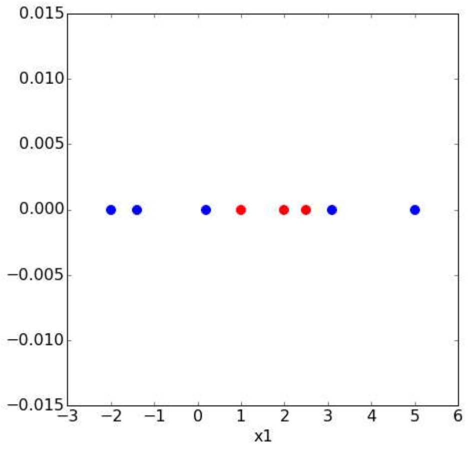


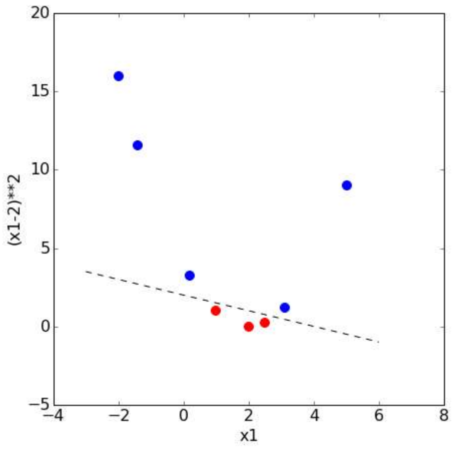
$f(x,\theta)$

support vector machines



kernel support vector machines





kernel support vector machines

SVMs can also be formulated as a linear function of the samples (dual form) instead of the features as

$$f(x,\theta) = \sum_{i=1}^{n} \theta_i (x \cdot x^{(i)}) + \theta_0$$

that can be reformulated as a non-linear function using what is know as a kernel function

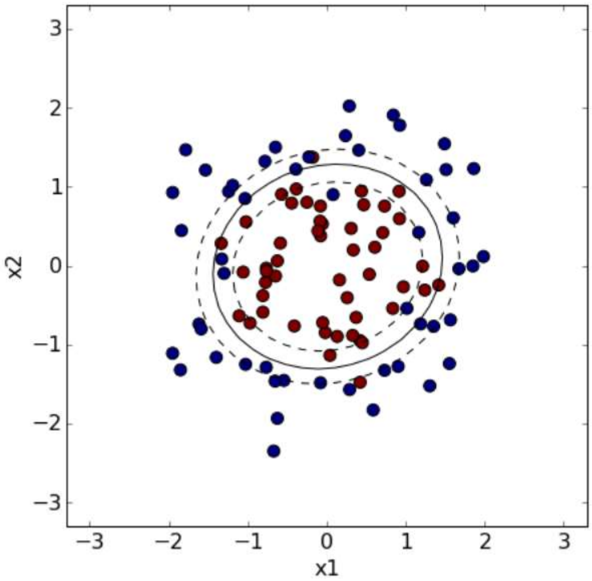
$$K(x^{(i)}, x^{(j)})$$

to become

$$f(x,\theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

The data points $x^{(i)}$ for which $\theta_i > 0$ are called the support vectors.

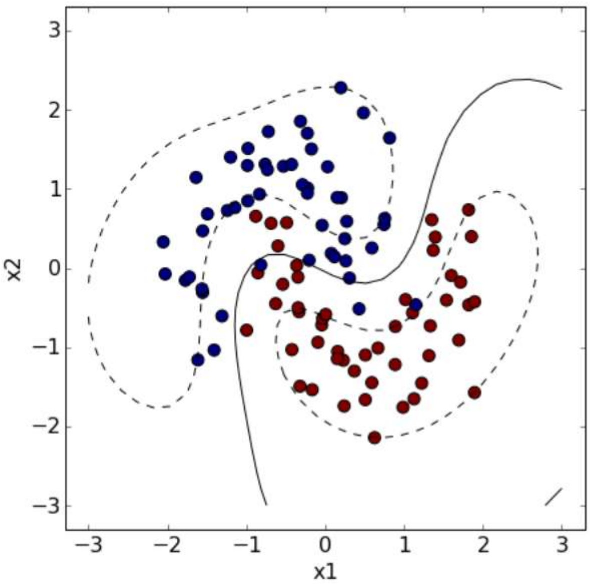
kernel support vector machines



$$f(x,\theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

$$K(x^{(i)}, x^{(j)}) = (x^{(i)} \cdot x^{(j)} + c)^d$$

kernel support vector machines



$$f(x,\theta) = \sum_{i=1}^{n} \theta_i K(x, x^{(i)}) + \theta_0$$

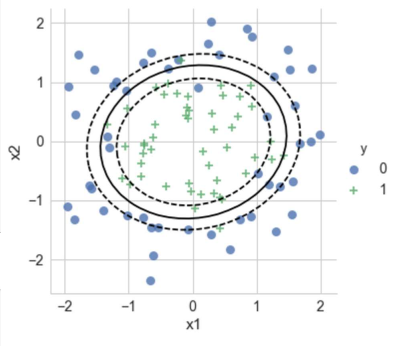
$$K(x^{(i)}, x^{(j)}) = \exp\left[-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right]$$

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

dataset1 = pd.read_csv('svm_example1.csv')
X = dataset1.copy()
y = X.pop('y')

clf = SVC(kernel='linear',C=1)
polynomial_features = PolynomialFeatures(degree=2)
model = Pipeline([("polynomial_features", polynomial_features),
                  ("SVC", clf)])
model.fit(X,y)

sns.lmplot(x="x1", y="x2", data=dataset1, hue='y', markers=['o','+'],
           fit_reg=False, size=5, scatter_kws={"s": 80})
compomics_import.plot_svm_decision_function(model)
plt.show()
```



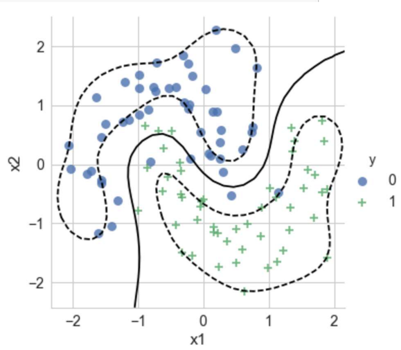

```
model = SVC(kernel='poly',C=1,degree=2)
model.fit(X,y)

sns.lmplot(x="x1", y="x2", data=dataset1, hue='y', markers=['o','+'],
           fit_reg=False, size=5, scatter_kws={"s": 80})
compomics_import.plot_svm_decision_function(model)
plt.show()
```

```
dataset2 = pd.read_csv("svm_example2.csv")
X = dataset2.copy()
y = X.pop('y')

model = SVC(kernel='rbf',C=1,gamma=1)
model.fit(X,y)

sns.lmplot(x="x1", y="x2", data=dataset2, hue='y', markers=['o','+'],
           fit_reg=False, size=5, scatter_kws={"s": 80})
compomics_import.plot_svm_decision_function(model)
plt.show()
```



```python
from sklearn import cross_validation
from sklearn.grid_search import GridSearchCV

search_space = np.logspace(-10, 14, 10, base=2)

params = dict(C=search_space)
grid_search = GridSearchCV(model, param_grid=params)

grid_search.fit(X, y)

for params, mean_score, scores in grid_search.grid_scores_:
    print("%0.3f (+/-%0.03f) for %r" % (mean_score, scores.std() * 2, params))
```