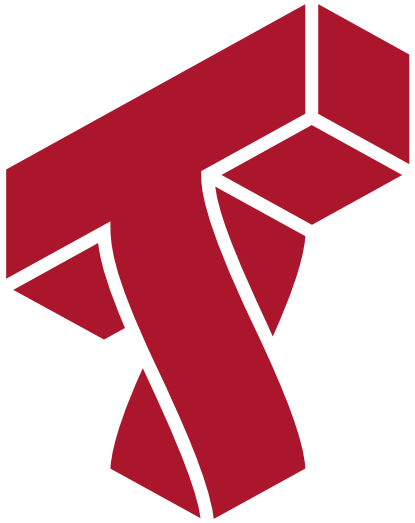


Lecture 12: Support Vector Machines

Part 1: Classification Margins

In this lecture, we are going to cover Support Vector Machines (SVMs), one the most successful classification algorithms in machine learning.

We start the presentation of SVMs by defining the classification *margin*.

Review: Binary Classification

Consider a training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$.

We distinguish between two types of supervised learning problems depending on the targets $y^{(i)}$.

1. **Regression**: The target variable $y \in \mathcal{Y}$ is continuous: $\mathcal{Y} \subseteq \mathbb{R}$.
2. **Binary Classification**: The target variable y is discrete and takes on one of $K = 2$ possible values.

In this lecture, we assume $\mathcal{Y} = \{-1, +1\}$.

Review: Linear Model Family

In this lecture, we will work with linear models of the form:

$$f_\theta(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \ldots + \theta_d \cdot x_d$$

where $x \in \mathbb{R}^d$ is a vector of features and $y \in \{-1, 1\}$ is the target. The θ_j are the *parameters* of the model.

We can represent the model in a vectorized form

$$f_\theta(x) = \theta^\top x + \theta_0.$$

Notation and The Iris Dataset

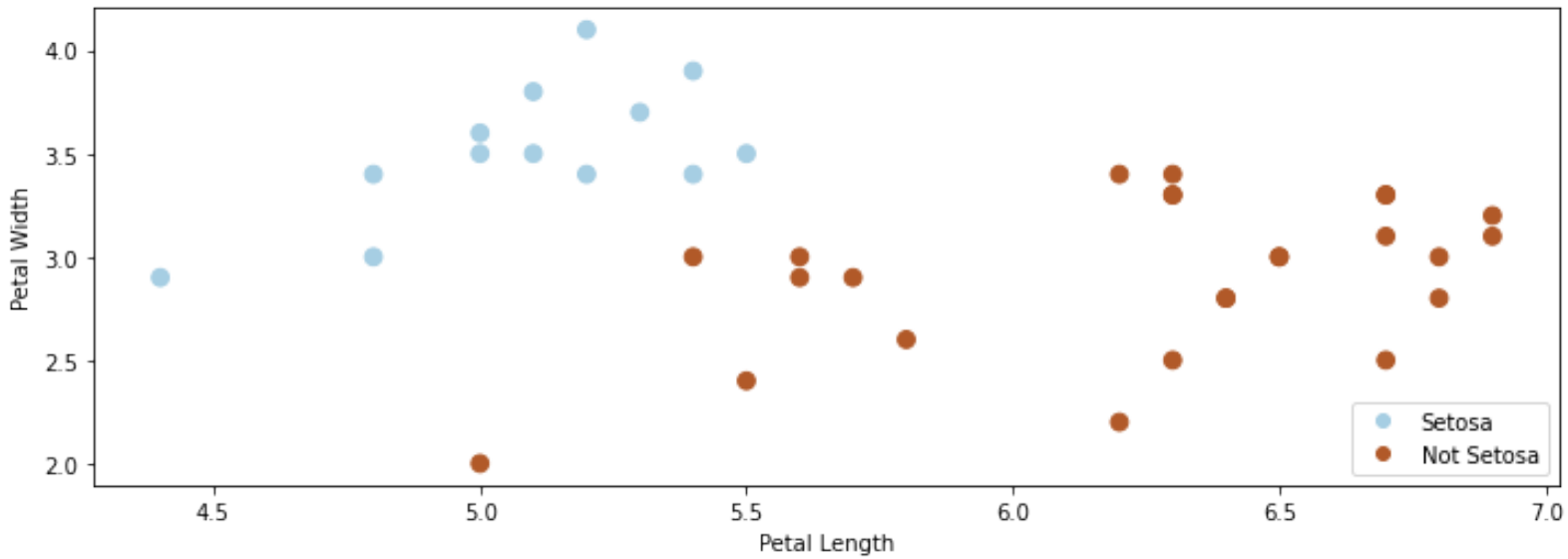
In this lecture, we are going to again use the Iris flower dataset.

As we just mentioned, we make two additional assumptions:

- We will only consider binary classification problems.
- We will use $\mathcal{Y} = \{-1, 1\}$ as the label space.

```
# https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [12, 4]
import warnings
warnings.filterwarnings("ignore")

# create 2d version of dataset and subsample it
X = iris_X.to_numpy()[:,:2]
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))

# Plot also the training points
p1 = plt.scatter(X[:, 0], X[:, 1], c=iris_y2, s=60, cmap=plt.cm.Paired)
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.legend(handles=p1.legend_elements()[0], labels=['Setosa', 'Not Setosa'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x12b01cd30>
```



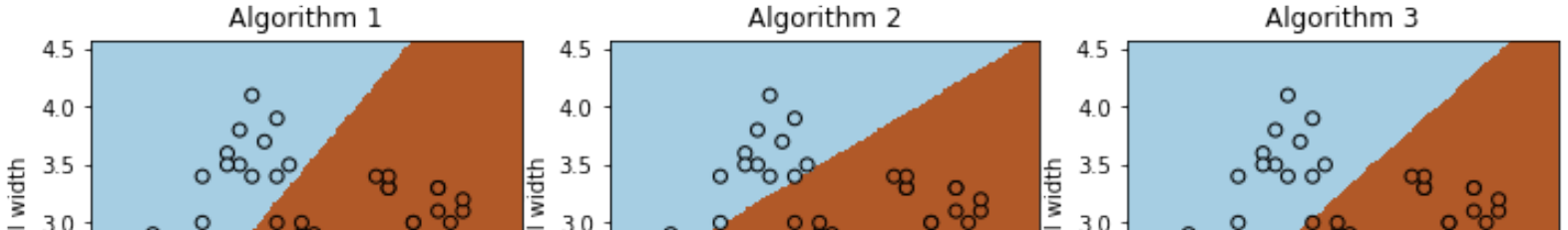
Comparing Classification Algorithms

We have seen different types approaches to classification.

When fitting a model, there may be many valid decision boundaries. How do we select one of them?

Consider the following three classification algorithms from `sklearn`. Each of them outputs a different classification boundary.

```
from sklearn.linear_model import LogisticRegression, Perceptron, RidgeClassifier
models = [LogisticRegression(), Perceptron(), RidgeClassifier()]

def fit_and_create_boundary(model):
    model.fit(X, iris_y2)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return Z

plt.figure(figsize=(12,3))
for i, model in enumerate(models):
    plt.subplot('13%d' % (i+1))
    Z = fit_and_create_boundary(model)
    plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=iris_y2, edgecolors='k', cmap=plt.cm.Paired)
    plt.title('Algorithm %d' % (i+1))
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')

plt.show()
```



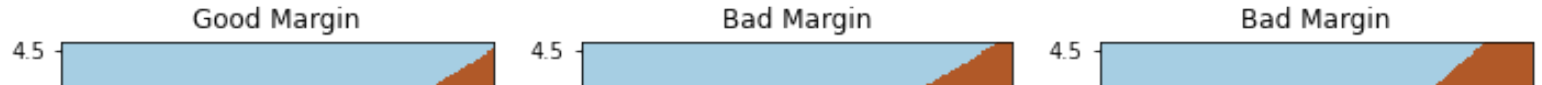
The Max-Margin Principle

Intuitively, we want to select boundaries with high *margin*.

This means that we are as confident as possible for every point and we are as far as possible from the decision boundary.

Several of the separating boundaries in our previous example had low margin: they came too close to the boundary.

```
from sklearn.linear_model import Perceptron, RidgeClassifier
from sklearn.svm import SVC
models = [SVC(kernel='linear', C=10000), Perceptron(), RidgeClassifier()]

def fit_and_create_boundary(model):
    model.fit(X, iris_y2)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return Z

plt.figure(figsize=(12,3))
for i, model in enumerate(models):
    plt.subplot('13%d' % (i+1))
    Z = fit_and_create_boundary(model)
    plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=iris_y2, edgecolors='k', cmap=plt.cm.Paired)
    if i == 0:
        plt.title('Good Margin')
    else:
        plt.title('Bad Margin')
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')

plt.show()
```



Below, we plot a decision boundary between the two classes (solid line) that has a high margin. The two dashed lines that lie at the margin.

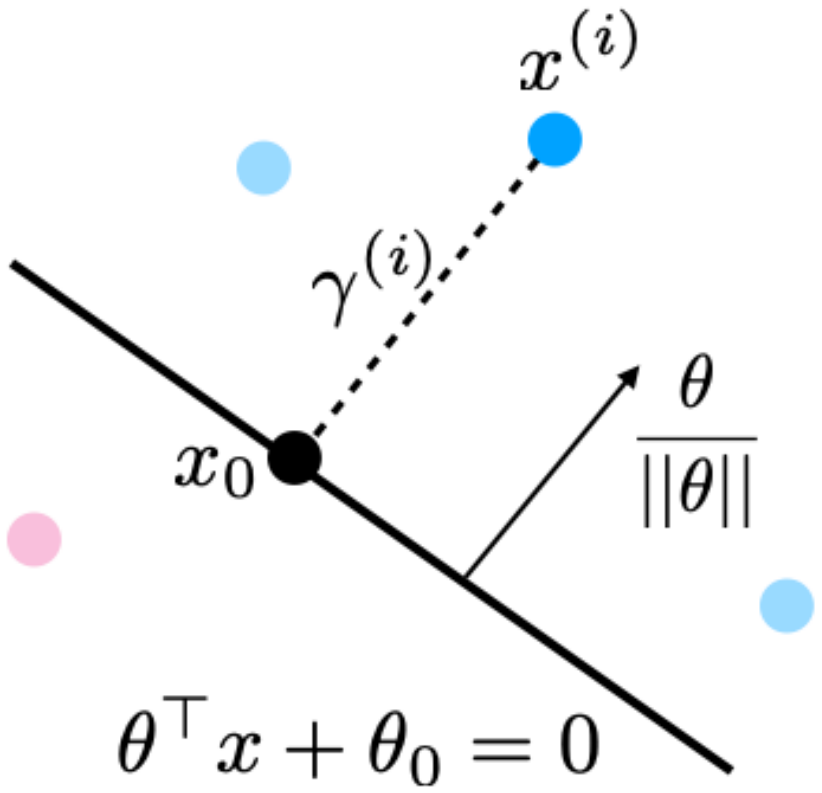
Points that are the margin are highlighted in black. A good decision boundary is as far away as possible from the points at the margin.

```
#https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html
from sklearn import svm

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel='linear', C=1000) # we'll explain this algorithm shortly
clf.fit(X, iris_y2)

plt.figure(figsize=(5,5))
plt.scatter(X[:, 0], X[:, 1], c=iris_y2, s=30, cmap=plt.cm.Paired)
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

# plot decision boundary and margins
plt.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.xlim([4.6, 6])
plt.ylim([2.25, 4])
```

```
(2.25, 4.0)
```

Geometric Margin

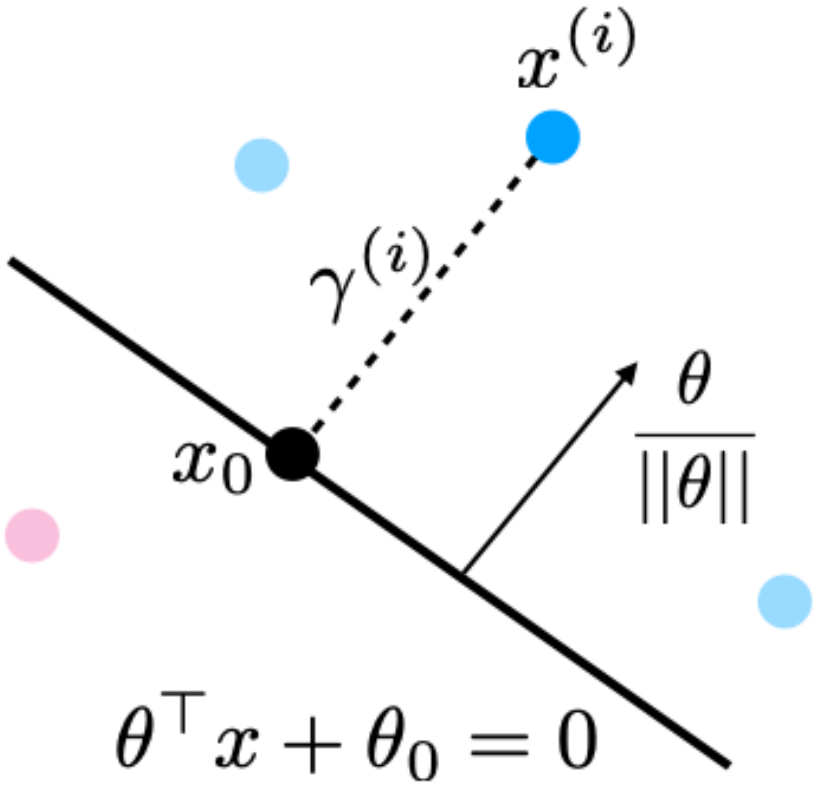
The margin $\gamma^{(i)} = y^{(i)} \left(\frac{\theta^\top x^{(i)} + \theta_0}{||\theta||} \right)$ is the distance from $x^{(i)}$ to the separating hyperplane $\theta^\top x + \theta_0 = 0$ (dashed line below).



Suppose that $y^{(i)} = 1$ ($x^{(i)}$ lies on positive side of boundary). Then:

1. The points x that lie on the decision boundary are those for which $\theta^\top x + \theta_0 = 0$ (score is precisely zero, and between 1 and -1).

1. The vector $\frac{\theta}{||\theta||}$ is perpendicular to the hyperplane $\theta^\top x + \theta_0$ and has unit norm (fact from calculus).



1. Let x_0 be the point on the boundary closest to $x^{(i)}$. Then by definition of the margin $x^{(i)} = x_0 + \gamma^{(i)} \frac{\theta}{||\theta||}$ or

$$x_0 = x^{(i)} - \gamma^{(i)} \frac{\theta}{||\theta||}.$$

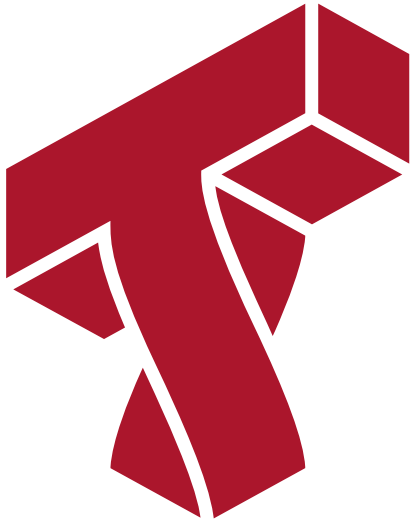
1. Since x_0 is on the hyperplane, $\theta^\top x_0 + \theta_0 = 0$, or

$$\theta^\top \left(x^{(i)} - \gamma^{(i)} \frac{\theta}{||\theta||} \right) + \theta_0 = 0.$$

1. Solving for $\gamma^{(i)}$ and using the fact that $\theta^\top \theta = ||\theta||^2$, we obtain

$$\gamma^{(i)} = \frac{\theta^\top x^{(i)} + \theta_0}{||\theta||}.$$

Which is our geometric margin. The case of $y^{(i)} = -1$ can also be proven in a similar way.



Part 2: The Max-Margin Classifier

We have seen a way to measure the confidence level of a classifier at a data point using the notion of a *margin*.

Next, we are going to see how to maximize the margin of linear classifiers.

Maximizing the Margin

We want to define an objective that will result in maximizing the margin. As a first attempt, consider the following optimization problem.

$$\max_{\theta, \theta_0, \gamma} \gamma$$

$$\text{subject to } \; y^{(i)} \frac{(x^{(i)})^\top \theta + \theta_0}{||\theta||} \geq \gamma \text{ for all } i$$

This is maximizes the smallest margin over the $(x^{(i)}, y^{(i)})$. It guarantees each point has margin at least γ.

Maximizing the Margin

This problem is difficult to optimize because of the division by $||\theta||$ and we would like to simplify it. First, consider the equivalent problem:

$$\max_{\theta,\theta_0,\gamma} \gamma$$
$$\text{subject to } \; y^{(i)}((x^{(i)})^\top \theta + \theta_0) \geq \gamma||\theta|| \text{ for all } i$$

Note that this problem has an extra degree of freedom:

- Suppose we multiply θ, θ_0 by some constant $c > 0$
- This yields another valid solution!

To enforce uniqueness, we add another constraint that doesn't change the minimizer:

$$||\theta|| = 1/\gamma.$$

Maximizing the Margin

If the constraint $||\theta|| \cdot \gamma = 1$ holds, then we know that $\gamma = 1/||\theta||$ and we can replace γ in the optimization problem to obtain:

$$\max_{\theta,\theta_0} \frac{1}{||\theta||}$$
$$\text{subject to } \; y^{(i)}((x^{(i)})^\top \theta + \theta_0) \geq 1 \text{ for all } i$$

The solution of this problem is still the same.

Maximizing the Margin: Final Version

Finally, instead of maximizing $1/||\theta||$, we can minimize $||\theta||$, or equivalently we can minimize $\frac{1}{2}||\theta||^2$.

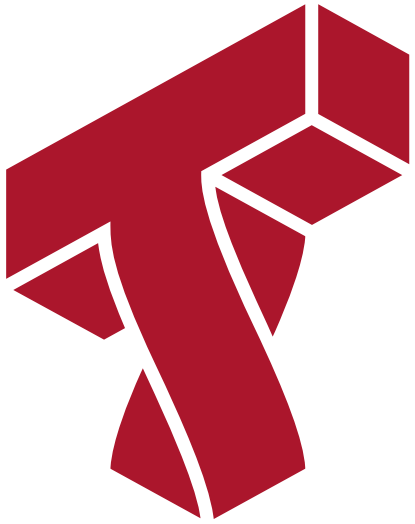
$$\min_{\theta,\theta_0} \frac{1}{2}||\theta||^2$$
$$\text{subject to } \; y^{(i)}((x^{(i)})^\top \theta + \theta_0) \geq 1 \text{ for all } i$$

This is now a quadratic program that can be solved using off-the-shelf optimization algorithms!

Algorithm: Linear Support Vector Machine Classification

- **Type**: Supervised learning (binary classification)
- **Model family**: Linear decision boundaries.
- **Objective function**: Max-margin optimization.
- **Optimizer**: Quadratic optimization algorithms.
- **Probabilistic interpretation**: No simple interpretation!

Later, we will see several other versions of this algorithm.



Part 3: Soft Margins and the Hinge Loss

Let's continue looking at how we can maximize the margin.

Non-Separable Problems

So far, we have assume that a linear hyperplane exists. However, what if the classes are non-separable? Then our optimization problem does not have a solution and we need to modify it.

Our solution is going to be to make each constraint "soft", by introducing "slack" variables, which allow the constraint to be violated.

$$y^{(i)}((x^{(i)})^\top \theta + \theta_0) \geq 1 - \xi_i.$$

- If we can classify each point with a perfect score of ≥ 1, the $\xi_i = 0$.
- If we cannot assign a perfect score, we assign a score of $1 - \xi_i$.
- We define optimization such that the ξ_i are chosen to be as small as possible.

In the optimization problem, we assign a penalty C to these slack variables to obtain:

$$\begin{aligned} &\min_{\theta,\theta_0,\xi} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to } \; y^{(i)}\left((x^{(i)})^\top\theta + \theta_0\right) \geq 1 - \xi_i \text{ for all } i \\ &\qquad\qquad \xi_i \geq 0 \end{aligned}$$

Towards an Unconstrained Objective

Let's further modify things. Moving around terms in the inequality we get:

$$\min_{\theta,\theta_0,\xi} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } \xi_i \geq 1 - y^{(i)}\left((x^{(i)})^\top\theta + \theta_0\right) \qquad \xi_i \geq 0 \text{ for all } i$$

Because I'm minimizing ξ_i, one of the inequalities has to hold with equality.

Thus, $\xi_i = \max\left(1 - y^{(i)}\left((x^{(i)})^\top\theta + \theta_0\right), 0\right)$.

We simplify notation a bit by using the notation $(x)^+ = \max(x, 0)$.

This yields:

$$\xi_i = \max\left(1 - y^{(i)}\left((x^{(i)})^\top \theta + \theta_0\right), 0\right) := \left(1 - y^{(i)}\left((x^{(i)})^\top \theta + \theta_0\right)\right)^+$$

Towards an Unconstrained Objective

We can now take

$$\min_{\theta,\theta_0,\xi} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n} \xi_i$$
$$\text{subject to } \xi_i = \left(1 - y^{(i)}\left((x^{(i)})^\top\theta + \theta_0\right)\right)^+ \text{for all } i$$

And we turn it into the following by plugging in the definition of ξ_i:

$$\min_{\theta,\theta_0} \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n} \left(1 - y^{(i)}\left((x^{(i)})^\top\theta + \theta_0\right)\right)^+$$

Since it doesn't matter which term we multiply by $C > 0$, this is equivalent to

$$\min_{\theta, \theta_0, \xi} \sum_{i=1}^{n} \left(1 - y^{(i)} \left((x^{(i)})^\top \theta + \theta_0\right)\right)^+ + \frac{\lambda}{2} ||\theta||^2$$

for some $\lambda > 0$.

An Unconstrained Objective

We have now turned our optimization problem into an unconstrained form:

$$\min_{\theta,\theta_0} \sum_{i=1}^{n} \underbrace{\left(1 - y^{(i)}\left((x^{(i)})^\top \theta + \theta_0\right)\right)^+}_{\text{hinge loss}} + \underbrace{\frac{\lambda}{2}||\theta||^2}_{\text{regularizer}}$$

- The hinge loss penalizes incorrect predictions.
- The L2 regularizer ensures the weights are small and well-behaved.

The Hinge Loss

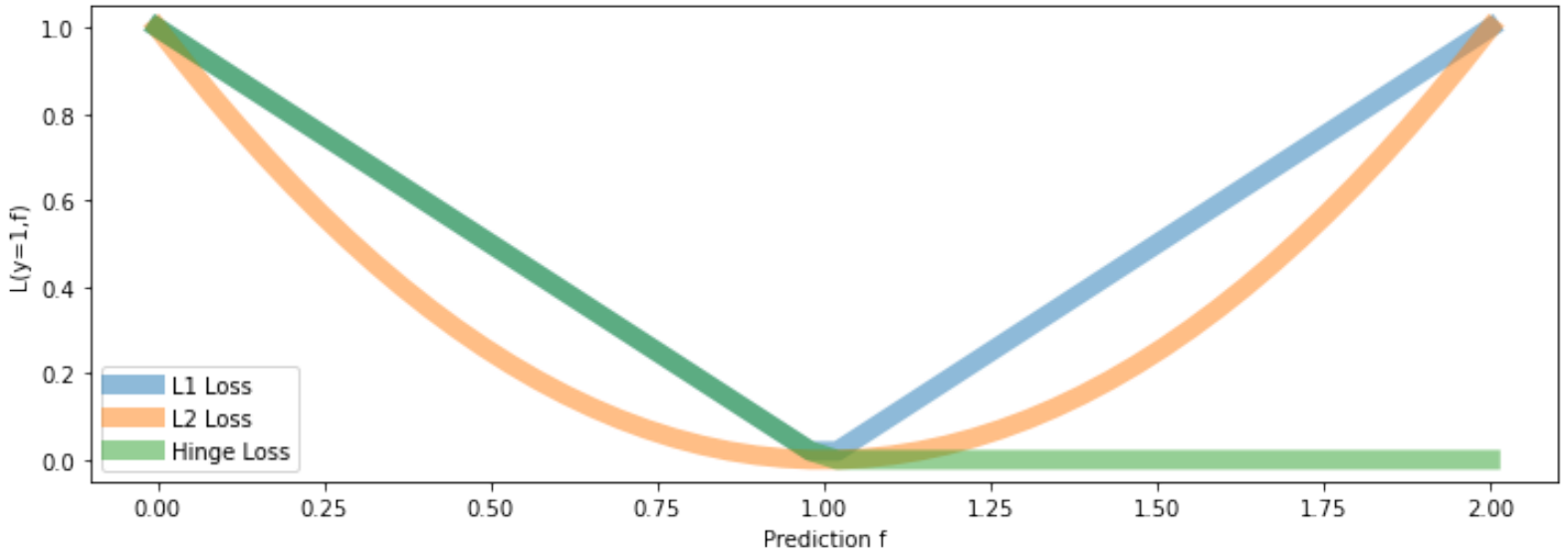
Consider again our new loss term for a label y and a prediction f:

$$L(y, f) = \max(1 - y \cdot f, 0).$$

Let's visualize a few losses $L(y=1, f)$, as a function of f:

```
# define the losses for a target of y=1
hinge_loss = lambda f: np.maximum(1 - f, 0)
l2_loss = lambda f: (1-f)**2
l1_loss = lambda f: np.abs(f-1)

# plot them
fs = np.linspace(0, 2)
plt.plot(fs, l1_loss(fs), fs, l2_loss(fs), fs, hinge_loss(fs), linewidth=9, alpha=0.5)
plt.legend(['L1 Loss', 'L2 Loss', 'Hinge Loss'])
plt.xlabel('Prediction f')
plt.ylabel('L(y=1,f)')
```

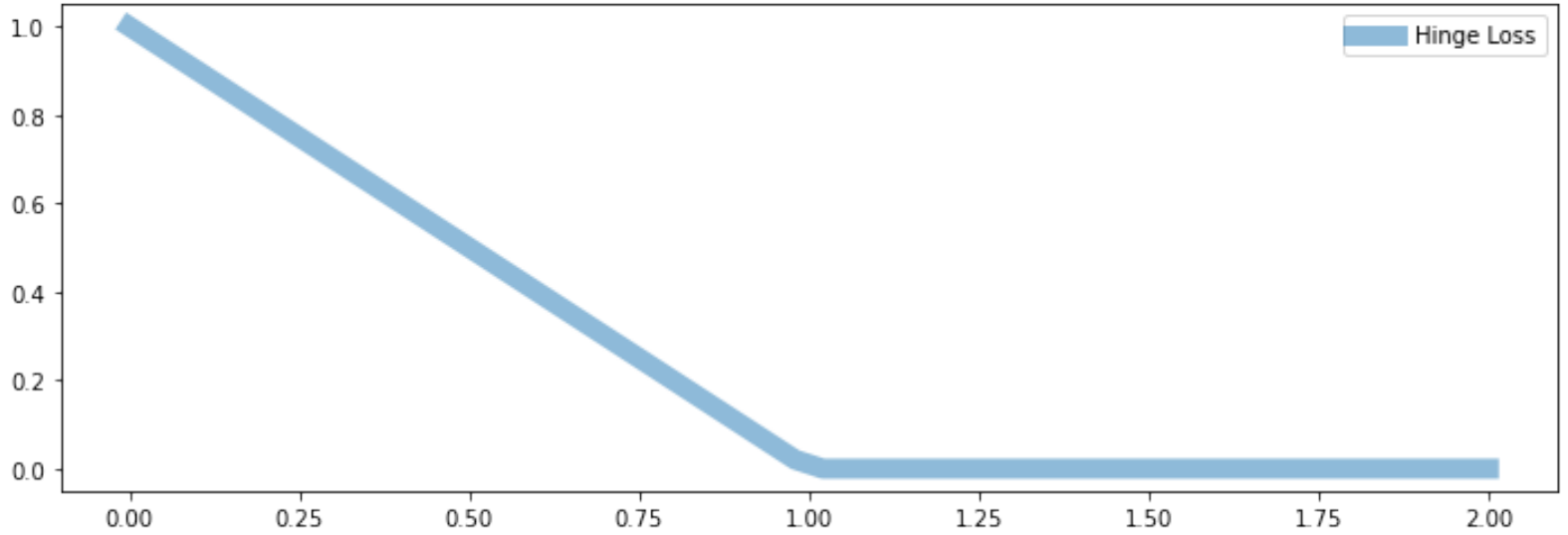
```
Text(0, 0.5, 'L(y=1,f)')
```



- The hinge loss is linear like the L1 loss.
- But it only penalizes errors that are on the "wrong" side:
    - We have an error of $|f - y|$ if true class is 1 and $f < 1$
    - We don't penalize for predicting $f > 1$ if true class is 1.

```
plt.plot(fs, hinge_loss(fs), linewidth=9, alpha=0.5)
plt.legend(['Hinge Loss'])
```

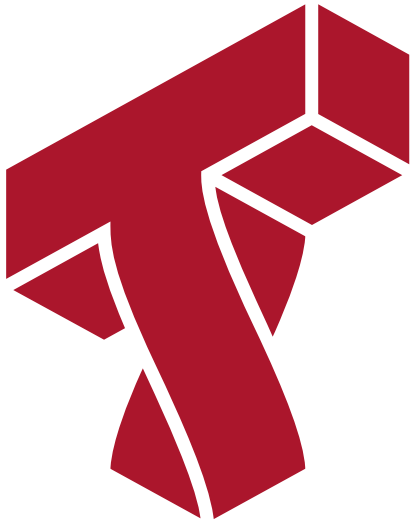
```
<matplotlib.legend.Legend at 0x12e750a58>
```



Properties of the Hinge Loss

The hinge loss is one of the best losses in machine learning!

- It penalizes errors "that matter", hence is less sensitive to outliers.
- Minimizing a regularized hinge loss optimizes for a high margin.
- The loss is non-differentiable at point, which may make it more challenging to optimize.



Part 4: Optimization for SVMs

We have seen a new way to formulate the SVM objective. Let's now see how to optimize it.

Review: SVM Objective

Maximizing the margin can be done in the following form:

$$\min_{\theta,\theta_0,\xi} \sum_{i=1}^{n} \underbrace{\left(1 - y^{(i)}\left((x^{(i)})^\top \theta + \theta_0\right)\right)^+}_{\text{hinge loss}} + \underbrace{\frac{\lambda}{2}||\theta||^2}_{\text{regularizer}}$$

- The hinge loss penalizes incorrect predictions.
- The L2 regularizer ensures the weights are small and well-behaved.

We can easily implement this objective in `numpy`.

First we define the model.

```
def f(X, theta):
    """The linear model we are trying to fit.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional data matrix

    Returns:
    y_pred (np.array): n-dimensional vector of predicted targets
    """
    return X.dot(theta)
```

And then we define the objective.

```
def svm_objective(theta, X, y, C=.1):
    """The cost function, J, describing the goodness of fit.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional design matrix
    y (np.array): n-dimensional vector of targets
    """
    return (np.maximum(1 - y * f(X, theta), 0) + C * 0.5 * np.linalg.norm(theta[:-1])**2).mean()
```

A Gradient for the Hinge Loss?

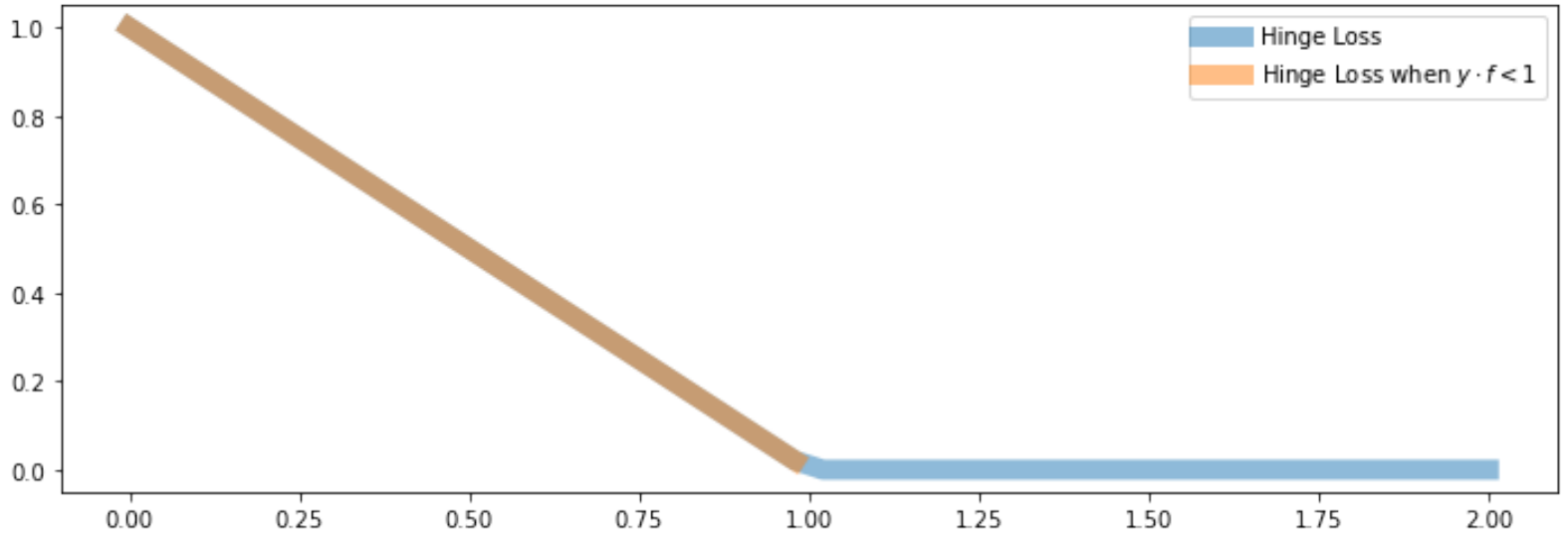
What is the gradient for the hinge loss with a linear f?

$$J(\theta) = \max\left(1 - y \cdot f_\theta(x), 0\right) = \max\left(1 - y \cdot \theta^\top x, 0\right).$$

Here, you see the linear part of J that behaves like $1 - y \cdot f_\theta(x)$ (when $y \cdot f_\theta(x) < 1$) in orange:

```
plt.plot(fs, hinge_loss(fs),fs[:25], hinge_loss(fs[:25]), linewidth=9, alpha=0.5)
plt.legend(['Hinge Loss', 'Hinge Loss when $y \cdot f < 1$'])
```

```
<matplotlib.legend.Legend at 0x12ea6f940>
```



When $y \cdot f_\theta(x) < 1$, we are in the "orange line" part and $J(\theta)$ behaves like $1 - y \cdot f_\theta(x)$.

Hence the gradient in this regime is:

$$\nabla_\theta J(\theta) = -y \cdot \nabla f_\theta(x) = -y \cdot x$$

where we used $\nabla_\theta \theta^\top x = x$.

When $y \cdot f_\theta(x) \geq 1$, we are in the "flat" part and $J(\theta) = 0$.

Hence the gradient is also just zero!

A Steepest Descent Direction for the Hinge Loss

We can define a "gradient" like function $\tilde{\nabla}_\theta J(\theta)$ for the hinge loss

$$J(\theta) = \max\left(1 - y \cdot f_\theta(x), 0\right) = \max\left(1 - y \cdot \theta^\top x, 0\right).$$

It equals:

$$\tilde{\nabla}_\theta J(\theta) = \begin{cases} -y \cdot x & \text{if } y \cdot f_\theta(x) < 1 \\ 0 & \text{otherwise} \end{cases}$$

(Sub-)Gradient Descent for SVM

Putting this together, we obtain a gradient descent algorithm (technically, it's called subgradient descent).

```
theta, theta_prev = random_initialization()
while abs(J(theta) - J(theta_prev)) > conv_threshold:
    theta_prev = theta
    theta = theta_prev - step_size * approximate_gradient
```

Let's implement this algorithm.

First we implement the approximate gradient.

```
def svm_gradient(theta, X, y, C=.1):
    """The (approximate) gradient of the cost function.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional design matrix
    y (np.array): n-dimensional vector of targets

    Returns:
    subgradient (np.array): d-dimensional subgradient
    """
    yy = y.copy()
    yy[y*f(X,theta)>=1] = 0
    subgradient = np.mean(-yy * X.T, axis=1)
    subgradient[:-1] += C * theta[:-1]
    return subgradient
```

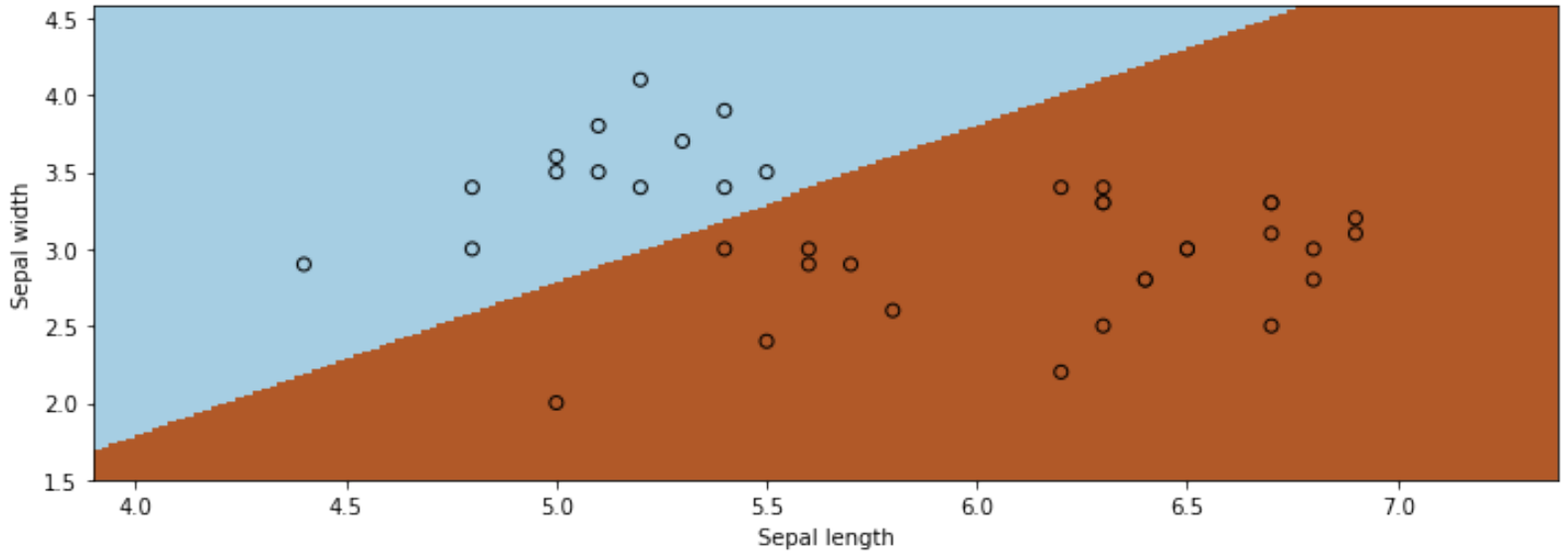
And then we implement subgradient descent.

```
threshold = 5e-4
step_size = 1e-2

theta, theta_prev = np.ones((3,)), np.zeros((3,))
iter = 0
iris_X['one'] = 1
X_train = iris_X.iloc[:,[0,1,-1]].to_numpy()
y_train = iris_y2.to_numpy()

while np.linalg.norm(theta - theta_prev) > threshold:
    if iter % 1000 == 0:
        print('Iteration %d. J: %.6f' % (iter, svm_objective(theta, X_train, y_train)))
    theta_prev = theta
    gradient = svm_gradient(theta, X_train, y_train)
    theta = theta_prev - step_size * gradient
    iter += 1
```

```
Iteration 0. J: 3.728947
Iteration 1000. J: 0.376952
Iteration 2000. J: 0.359075
Iteration 3000. J: 0.351587
Iteration 4000. J: 0.344411
Iteration 5000. J: 0.337912
Iteration 6000. J: 0.331617
Iteration 7000. J: 0.326604
Iteration 8000. J: 0.322224
Iteration 9000. J: 0.319250
Iteration 10000. J: 0.316727
Iteration 11000. J: 0.314800
Iteration 12000. J: 0.313181
Iteration 13000. J: 0.311843
```

We can visualize the results to convince ourselves we found a good boundary.

```
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z = f(np.c_[xx.ravel(), yy.ravel(), np.ones(xx.ravel().shape)], theta)
Z[Z<0] = 0
Z[Z>0] = 1

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.show()
```



Algorithm: Linear Support Vector Machine Classification

- **Type**: Supervised learning (binary classification)
- **Model family**: Linear decision boundaries.
- **Objective function**: L2-regularized hinge loss.
- **Optimizer**: Subgradient descent.
- **Probabilistic interpretation**: No simple interpretation!