Classification

How do we predict a target that is a categorical rather than a number (as in the Regression task) ?

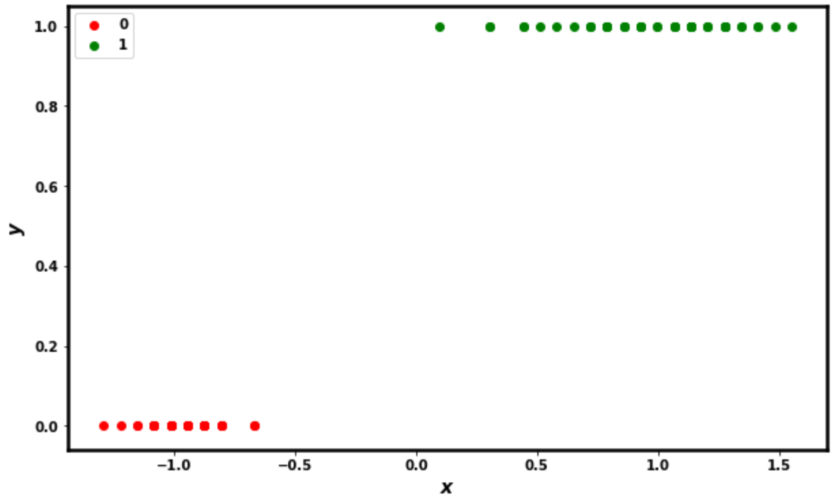
To be concrete: we consider the *Binary Classification* task

- Two classes (categories)
- Refer to the classes as Positive and Negative

Consider examples with a single feature.

We will encode Positive/Negative as the numbers 1/0 and plot them as Green/Red

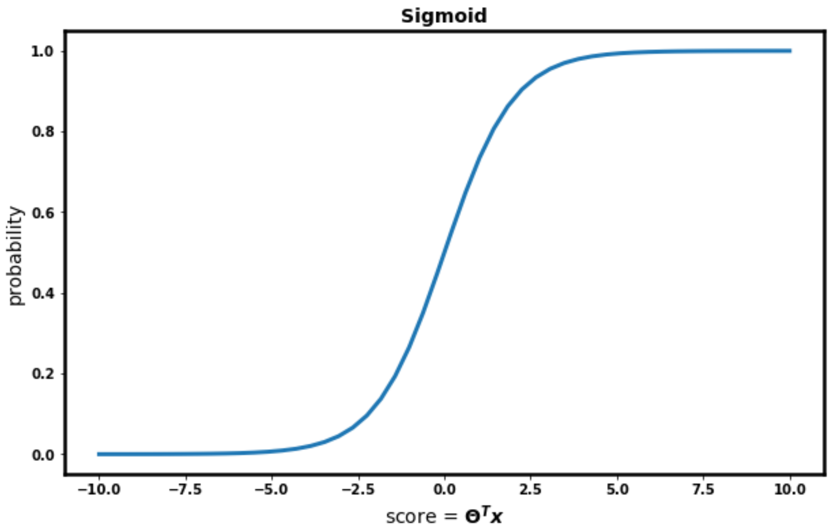
In [15]:
```
X_ls, y_ls = lsh.load_iris()

fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot_y_vs_x(ax, X_ls[:,0], y_ls)
```



As you can see, a model that fits straight line (Linear Regression) would be a poor fit.

There is a function, the *sigmoid*, that has the right shape:

In [16]:
```
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot_sigmoid(ax)
```



We

- fit a *linear function* of $\mathbf{x}$ called the "score" (i.e., $\Theta^T \mathbf{x}$) or *logit*
- and translate the score into something like a binary choice by mapping it via the sigmoid.

So fitting the sigmoid function to the data might be a reasonable solution to the Binary Classification Task.

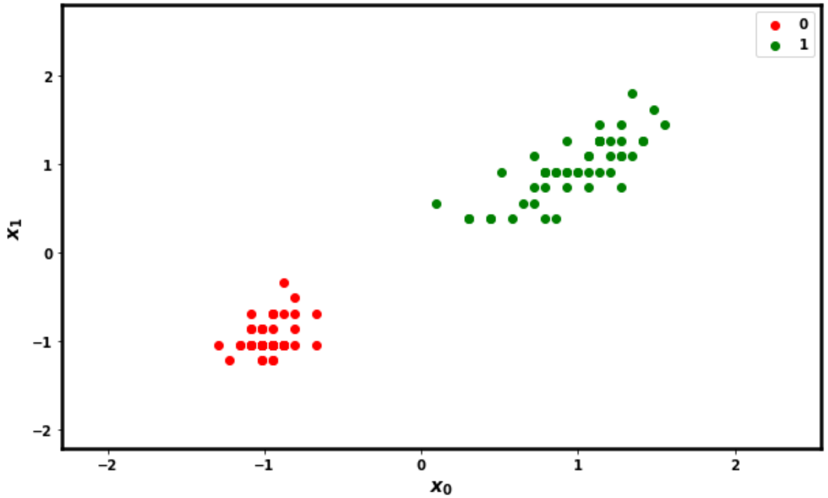
It turns out that we can adapt the Linear Regression model to this end.

Let's explore this idea further

Here is a example with two features.

Again, Positive/Negative examples are plotted in Green/Red.

In [17]:
```
clf_ls = lsh.fit_LR(X_ls,y_ls)
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot(ax, clf_ls, X_ls, y_ls, draw_boundary=False, scores=np.array([]))
```



Might it be possible to use Linear Regression to fit a line that separates Positive and Negative examples ?

An obvious idea:

- Use the features $\mathbf{x^{(i)}}$ to compute a "score" (*logit*) $\hat{s}^{(\mathbf{i})}$
- Compare the predicted score to a threshold
- Predict Positive if the score exceeds the threshold; Negative otherwise

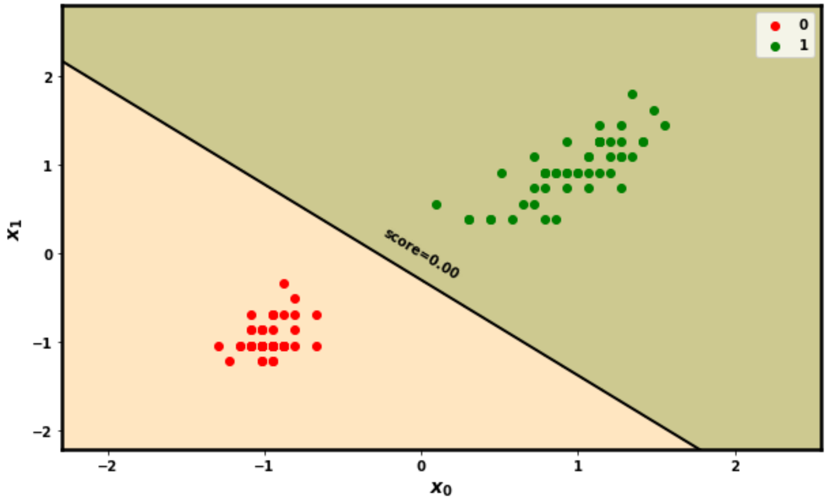
$$\hat{\mathbf{y}}^{(\mathbf{i})} = \begin{cases} \text{Negative} & \text{if } \hat{s}^{(\mathbf{i})} < 0 \\ \text{Positive} & \text{if } \hat{s}^{(\mathbf{i})} \geq 0 \end{cases}$$

If the score has the form of a Linear Regression

$$s(\mathbf{x}) = \Theta^T \mathbf{x}$$

then we can plot the surface (line in two dimensions) where the value of the score is equal to a constant (e.g., 0).

In [18]:
```
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot(ax, clf_ls, X_ls, y_ls, draw_prob=False)
```



That is: the score $s(\mathbf{x})$

- Is linear in features $\mathbf{x}$
- Separates Positive from Negative examples
    - Examples $(\mathbf{x}_0, \mathbf{x}_1)$ withs non-negative scores (i.e, points above the line) get classified as Positive
    - Examples $(\mathbf{x}_0, \mathbf{x}_1)$ withs negative scores (i.e, points below the line) get classified as Negative

If we can successfully classify by this method, the dataset set is *linearly separable*

A classifier for linearly separable data fits a hyperplane (e.g., the line $\hat{s} = 0$) to the training data such that

- Examples lying above the plane are classified as Positive
- Examples lying below the plane are classified as Negative

$$s = \Theta^T \mathbf{x}$$

Can be interpreted as

- using template matching on the features $\mathbf{x}$ to produce a "score"
  $s = \Theta^T \mathbf{x} = \Theta \cdot \mathbf{x}$

Transforming Binary Classification into Linear Regression

How do we fit the scoring function ?

We adapt Linear Regression.

Let's reinterpret the targets/labels $\mathbf{y}^{(\mathbf{i})}$ as a probability $p^{(\mathbf{i})}$

$$p^{(\mathbf{i})} = p(\mathbf{y}^{(\mathbf{i})} = \text{Positive} \mid \mathbf{x}^{(\mathbf{i})})$$

So

- $\mathbf{y^{(i)}} = \text{Positive}$ is equivalent to $p^{(\mathbf{i})} = 1$: the target for example i is Positive with 100% probabiity
- $\mathbf{y^{(i)}} = \text{Negative}$ is equivalent to $p^{(\mathbf{i})} = 0$: the target for example i is Positive with 0% (i.e., is Negative)

We can go further: map $\hat{s}^{(\mathbf{i})}$, which is continuous, into a continuous probability $\hat{p}^{(\mathbf{i})} \in [0, 1]$.

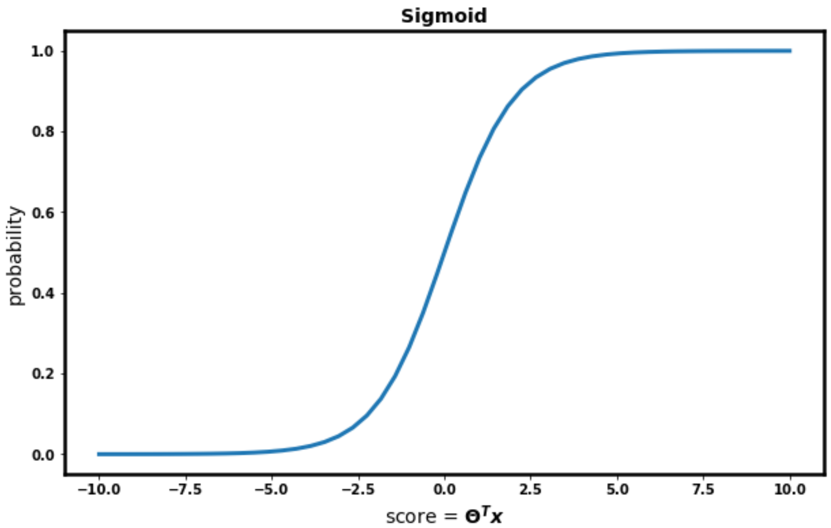
A very large predicted score $\hat{s}^{(\mathbf{i})}$ (far greater than a threshold) corresponds to $\hat{p}^{(\mathbf{i})} \approx 1$.

The *Logistic Function* $\sigma(s)$ transforms a number s (e.g., score) into a probability

$$\hat{p} = \sigma(s) = \frac{1}{1 + e^{-s}}$$

Let's plot the logistic function to gain some intuition:

In [19]:
```
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot_sigmoid(ax)
```



As you can see, it acts almost like a binary "switch"

- range is mostly 0 or 1

So this function creates a sharp boundary (measured in probability).

Now that we can convert between scores and probabilities the following two forms of classification are equivalent

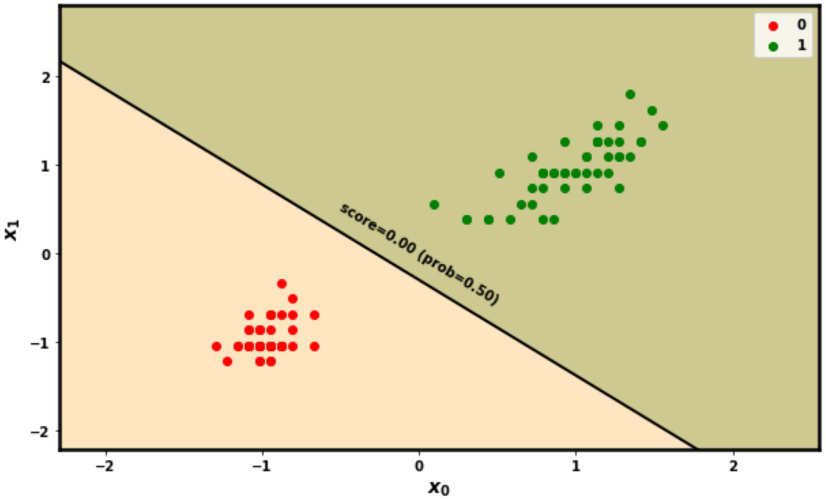
$$\hat{\mathbf{y}}^{(\mathbf{i})} = \begin{cases} \text{Negative} & \text{if } \hat{s}^{(\mathbf{i})} < 0 \\ \text{Positive} & \text{if } \hat{s}^{(\mathbf{i})} \geq 0 \end{cases}$$

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \begin{cases} \text{Negative} & \text{if } \hat{p}^{(\mathbf{i})} < 0.5 \\ \text{Positive} & \text{if } \hat{p}^{(\mathbf{i})} \geq 0.5 \end{cases}$$

This follows since $\sigma(0) = 0.5$.

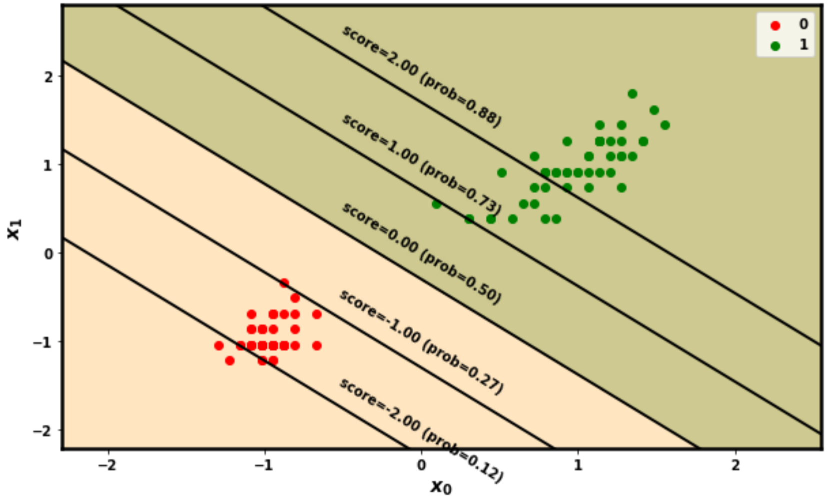
Relabeling the separating line with both score and probability:

In [20]:
```
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot(ax, clf_ls, X_ls, y_ls)
```



One can see the relationship between score and probability by looking at lines of constant score/probability.

In [21]:
```
fig, ax = plt.subplots(figsize=(10,6))
_= lsh.plot(ax, clf_ls, X_ls, y_ls, scores = np.arange(-2, 3,1))

fig.savefig(os.path.join("/tmp",'class_overview_prob_lines.jpg') )
```



- Increasingly positive scores result in increasing probability of Positive
- Increasingly negative scores result in decreasing probability of Positive (and hence increasing probability of Negative)

When the score is infinite, the probability becomes 100% (positive infinity) or 0% (negative infinity)

Logistic Regression

Because we use the Logistic function to map scores to probabilities, this method is called *Logistic Regression*.

To recap:

$$\begin{aligned} s &= \Theta^T \mathbf{x} \\ \hat{p} &= \sigma(s) \end{aligned}$$

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \begin{cases} \text{Negative} & \text{if } \hat{p}^{(\mathbf{i})} < 0.5 \\ \text{Positive} & \text{if } \hat{p}^{(\mathbf{i})} \geq 0.5 \end{cases}$$

Preview

The expression for $\hat{p}$

$$\hat{p} = \sigma(\Theta^T \mathbf{x})$$

which involves

- template matching of features versus template (Θ) since $\Theta^T \mathbf{x} = \Theta \cdot \mathbf{x}$
- convert the score into a probability with the sigmoid function

will reappear in the Deep Learning part of the course.

Of all the functions to "squeeze" score s into the range $[0, 1]$, why choose the Logistic Function ?

Let's invert the relationship induced by the Logistic Function

$$\hat{p} = \sigma(s)$$

between probability $\hat{p}$ and s

$$
\begin{aligned}
\frac{\hat{p}}{1-\hat{p}} &= \frac{\frac{1}{1+e^{-s}}}{1-\frac{1}{1+e^{-s}}} \\
&= \frac{\frac{1}{1+e^{-s}}}{\frac{e^{-s}}{1+e^{-s}}} \\
&= e^{s} \\
\log_e \frac{\hat{p}}{1-\hat{p}} &= s
\end{aligned}
$$

So, using the logistic function to compute $\hat{p}$ results in

$$\log_e \frac{\hat{p}}{1-\hat{p}} = \Theta^T \mathbf{x}$$

The above equation has the form of Linear Regression where target $\mathbf{y}$ has been transformed to

$$\log_e \frac{\hat{p}}{1-\hat{p}}$$

The term $\frac{\hat{p}}{1-\hat{p}}$ is called the *odds* (of being Positive) so the dependent variable is the *log odds*.

We have thus transformed Binary Classification into Linear Regression.

This introduction glosses over several problems, which we will subsequently address

- the log odds is positive infinity when $p = 1$
- the log odds is negative infinity when $p = 0$

This means that MSE can't be used as a Loss Function for fitting since some residuals are infinite.

In [22]:

```python
print("Done")
```

Done