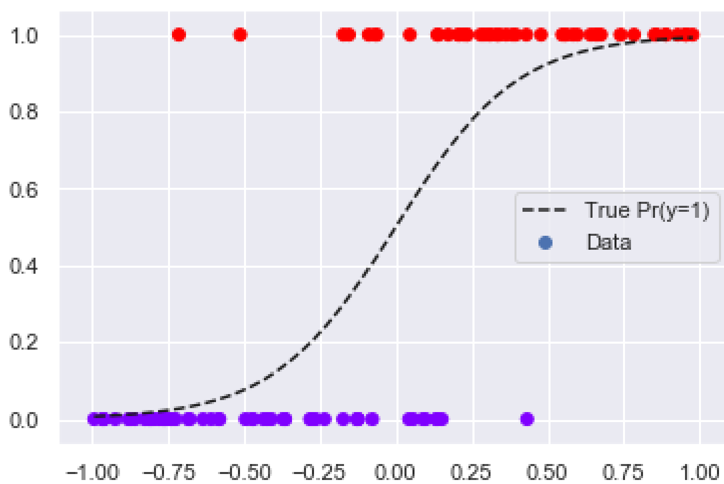
In [1]:
```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

# Create simple logistic data
rng = np.random.RandomState(0)
n_samples = 100
x = 2*(rng.rand(n_samples)-0.5)

theta_true = 5
# Very simplified logistic regression model with only 1 parameter
def model(x, theta):
    return 1 / (1 + np.exp(-x.dot(theta)))
y = (rng.rand(*x.shape) <= model(x, theta_true)).astype(float)
print(y.shape)
# Plot data
xq = np.linspace(np.min(x), np.max(x))
plt.plot(xq, model(xq, theta_true), '--k', label='True Pr(y=1)')
plt.scatter(x, y, c=y, cmap='rainbow', label='Data')
plt.legend()
```

```
(100,)
```

Out[1]: <matplotlib.legend.Legend at 0x1a238395c0>

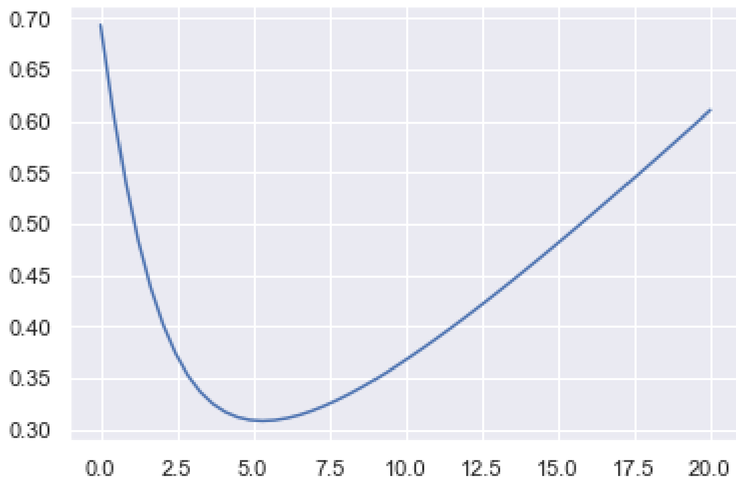


Define the logistic regression objective

In [2]:
```
def objective(x, y, theta):
    prob = model(x, theta)
    return -np.mean(y * np.log(prob) + (1-y) * np.log(1 - prob))

theta_arr = np.linspace(0, 20)
obj_arr = [objective(x, y, theta) for theta in theta_arr]
plt.plot(theta_arr, obj_arr)
```

Out[2]: [<matplotlib.lines.Line2D at 0x1a238fce10>]



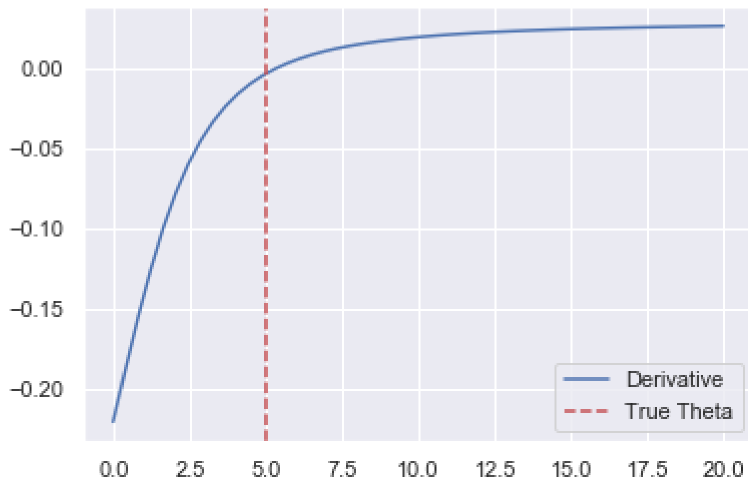
NOTE: Because of randomness, empirical estimate will not exactly match true model. (e.g., this has a minimum closer to 5.3 or so

Explicitly compute (by hand) the gradient of the function

In [3]:
```
def grad_objective(x, y, theta):
    #See https://web.stanford.edu/~jurafsky/slp3/5.pdf section 5.8 for d
erivation
    return np.mean((model(x, theta) - y) * x)

grad_arr = [grad_objective(x, y, theta) for theta in theta_arr]
plt.plot(theta_arr, grad_arr, label='Derivative')
plt.axvline(5, linestyle='--', color='r', label='True Theta')
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x1a239ef828>



(Stochastic) Gradient Descent with various step sizes

In [4]:

```
# Gradient descent parameters
max_iter = 5
step_size = 10 # 10, 100, 300
sgd = False
if sgd: max_iter *= 5 # Increase number of iterations for SGD
rng = np.random.RandomState(0)

# Initialization
theta_hat = 10.2 # An arbitrary starting value
theta_hat_arr = [theta_hat]
obj_hat_arr = [objective(x, y, theta_hat)]

# Gradient descent iterations
for it in range(max_iter):
    if sgd:
        # Select random data point
        rand_idx = rng.randint(len(y))
        xg, yg = x[rand_idx:rand_idx+1], y[rand_idx:rand_idx+1]
    else:
        # Use all data points in gradient calculation
        xg, yg = x, y
    grad = grad_objective(xg, yg, theta_hat)
    theta_hat = theta_hat - step_size * grad

    # Save estimates for visualization
    theta_hat_arr.append(theta_hat)
    obj_hat_arr.append(objective(x, y, theta_hat))

vis_arr = np.linspace(np.minimum(np.min(theta_hat_arr), theta_true), np.
maximum(np.max(theta_hat_arr), theta_true))
plt.plot(vis_arr, [objective(x, y, theta) for theta in vis_arr], label=
'Objective')
plt.plot(theta_hat_arr, obj_hat_arr, 'o-', label='Gradient steps')
plt.plot(np.ones(2)*theta_true, plt.ylim(), label='True value')
plt.legend()
```

Out[4]: <matplotlib.legend.Legend at 0x1a23b1c4e0>