Assignment 3: Classification, Logistic Regression, and Gradient Descent

Machine Learning

Fall 2019

Learning Objectives

- Learn about the framing of the classification problem in machine learning.
- Learn about the logistic regression algorithm.
- Learn about gradient descent for optimization.
- Some C&E topic.

Prior Knowledge Utilized

- Supervised learning problem framing.
- Training / testing splits.

Recall: Supervised Learning Problem Setup

We are given a training set, $(\mathbf{x_1}, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$ where each $\mathbf{x_i}$ represents an element of an input space (e.g., a d-dimensional feature vector) and each y_i represents an element of an output space (e.g., a scalar target value). Our goal is to determine a function $\hat{f}$ that maps from the input space to the output space.

We assume there is a loss function, ℓ, that determines the amount of loss that a particular prediction $\hat{y}_i$ incurs due to a mismatch with the actual output y_i. The best possible model, $\hat{f}^\star$, is the one that minimizes these losses over the training set. This notion can be expressed with the following equation.

$$\hat{f}^\star = \arg\min_{\hat{f}} \sum_{i=1}^{n} \ell\left(\hat{f}(\mathbf{x_i}), y_i\right) \tag{1}$$

1 *The Classification Problem*

So far in this class we've looked at supervised learning problems where the responses y_i are continuous-valued and the loss function is quadratic ($\ell(y, \hat{y}) = (y - \hat{y})^2$). This is an example of a regression problem. There are many times, however, where it is unnatural to frame a problem as a regression. For instance, it may be the case that y_i does not come from a continuous range but can only take on a few different values. This sort of problem is known as a classification problem. For instance, you might want to have a system that can take in an image of a person and predict their identity. The identity could be thought of as the output, y_i, and it could only take on one of several values (each value might represent a particular person the system was trained to recognize). In this assignment you'll learn about a special case of the classification problem known as binary classification (where y_i is either 0 or 1, e.g., a Paul versus Sam detector).

In this assignment will formalize the binary classification problem and see a very useful algorithm for solving it called *logistic regression*. You will also see that the logistic regression algorithm is a very natural extension of linear regression. Our plan for getting there is going to be pretty similar to what we did for linear regression.

- Build some mathematical foundations
- Introduce logistic regression from a top-down perspective
- Learn about logistic regression from a bottom-up perspective

2 Formalizing the Classification Problem

Let's start by making the binary classification problem more formal. Suppose, we are given a training set, $(\mathbf{x_1}, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$, where each $\mathbf{x_i}$ is an element of the input space (e.g., a vector) and each y_i is a binary number (either 1 or 0). In this setting we will attempt to use the training data to determine a function, $\hat{f}^\star$, that predicts the corresponding output, y, for any possible input, $\mathbf{x}$. To make this concrete with an example, $\mathbf{x}$ could be an image and y could be a binary number that takes on value 1 when the picture contains a puppy.

Exercise 1 (10 minutes)

(a) Given this setup of the binary classification problem, the only thing left to specify is the loss function, ℓ. Recall that ℓ takes as input the actual output y, and the predicted output $\hat{y}$. What function could you use for ℓ that would result in the learning algorithm choosing a good model. If the choice of ℓ depends on the application, how so?

(b) One natural choice, that you may have already come up with in the previous question, is to define our loss function as $\ell(y, \hat{y}) = \mathbb{I}[y \neq \hat{y}]$ (the funny looking $\mathbb{I}$ is the indicator function that takes on value 1 when the condition inside is true and 0 otherwise. Given these choices, the supervised learning problem becomes:

$$\hat{f}^\star = \arg\min_{\hat{f}} \sum_{i=1}^{n} \mathbb{I}\left[\hat{f}(\mathbf{x_i}) \neq, y_i\right] \quad . \tag{2}$$

Convert Equation 2 to English to make sure you understand it.

The loss function given in Exercise 2(b) is a totally reasonable choice since it will result in choosing the model that makes the fewest mistakes on the training. It turns out, however, that it has the following drawbacks.

- It is all or nothing. Either we are completely right or completely wrong.

- It is not a particularly easy function to work with mathematically. In fact, for many common classes of models, it will be difficult for the learning algorithm to find the best possible model[1].

[1] One of the key challenges that must be in met in machine learning, and modeling in general, is balancing computational considerations (e.g., how long does it take to find the best possible model) with the realism of the model (e.g., how directly does the task you pose to the learning algorithm match the problem you are solving. Sometimes these things are in conflict and you must make tradeoffs.

It turns out that we can create a much more natural loss function by thinking about predictions in terms of probabilities.

3 *Probability and the log loss*

Imagine that instead of our model, $\hat{f}$, spitting out either 0 or 1, it outputs a confidence that the input x_i has an output, y_i, that was 1. In other words, rather than giving us its best guess, the classifier would indicate to us its degree of certainty regarding its prediction. This notion of "certainty" can be formalized using the concept of a probability. That is, the model can output a probability that the output for a particular input is 1.

We haven't formally defined probability in this class, and we won't do so here (we'll be working with probabilities extensively in module 2 and we don't need a formal treatment to derive the classification algorithms we will study in this module). Here are a few things to keep a few things in mind about probabilities.

- A probability, p, specifies the chance that some event occurs ($p = 0$ means that the even will definitely not occur and $p = 1$ means that it will definitely occur.
- A probability, p, must be between 0 and 1 ($0 \leq p \leq 1$).
- If the probability an event occurs is p, then the probability that the event doesn't occur is $1 - p$.

✔ Understanding Check

Note: these sorts of boxes are here to help you test your understanding of a concept you have just read. We are trying to use these to breakup long blocks of reading. You do not need to submit these.

TODO: Quick check of understanding

3.1 *Log loss*

Thinking back to the binary classification problem, if we think in terms of probability, then our model output a probability p when supplied with an input $\mathbf{x}$ (i.e., $\hat{f}(\mathbf{x}) = p$). We might then ask ourselves the question of what would be a reasonable loss function to quantify how good a prediction p is given the actual output y (recall that for the binary classification the output is either 0 or 1). To make to this more intuitive, consider the task of quantifying the quality of a weatherperson's predictions. Further, let's assume that on any given day the weather is either sunny (call this output 1) or rainy (call this output 0). Suppose that each night the weather person provides a probability that indicates the chance of it being sunny the next day. In order to quantify the loss of each prediction we need to define a loss function. Here are two potential choices.

1. **0-1 loss:** we will extract from the weatherperson's prediction the most likely outcome (e.g., if $p = 0.75$, that would be sunny, if $p = 0.4$, that would be rainy). If this most likely outcome matches the actual outcome we give a loss of 0, otherwise we give a loss of 1 (this is similar to Equation 2).

2. **squared loss:** one downside of *0-1 loss* is that it doesn't take into account the certainty expressed by the weatherperson. The weatherperson gets the same loss if it is rainy and they predicted $p = 0.51$ or $p = 1$. For squared loss we can compute the difference between the outcome and p and square it to arrive at the loss. So if we predict $p = 0.51$ and it is sunny we get a loss of $(1 - 0.51)^2$. If it was rainy in this same example, we get a loss of $(0 - 0.51)^2$.

As an example, here are hypothetical predictions from two forecasters, the actual weather, and the resulting loss with either *0-1* loss or *squared loss*.

actual weather	forecast 1	0-1 loss	squared loss	forecast 2 1	0-1 loss	squared loss
sunny (y = 1)	$p = 0.3$	1	$(1 - 0.2)^2 = 0.64$	$p = 0.9$	0	$(1 - 0.9)^2 = 0.01$
rainy (y = 0)	$p = 0.6$	1	$(0 - 0.6)^2 = 0.36$	$p = 0.999$	1	$(0 - 0.999)^2 = 0.998$
sunny (y = 1)	$p = 0.8$	0	$(1 - 0.8)^2 = 0.16$	$p = 0.99$	0	$(1 - 0.99)^2 = 0.0001$
average		0.667	0.347		0.333	0.336

✓ Understanding Check

According to the table above, which forecaster is better with regards to *0-1 loss*? Which forecaster is better with regards to *squared loss*?

One entry in the table above is particularly interesting. In the third row, the second forecaster assigned a probability of 0.999 to the fact that it was going to be sunny. It turned out to rain (boo!!!). That is, the forecaster was almost certain it would be sunny and it wasn't. The 0-1 loss of course doesn't capture this at all. The squared loss seems to assign a fairly large loss. One might argue, though, that this loss does not fully capture how bad the prediction was. This last observation motivates a third loss function that we can use to evaluate probabilistic predictions: the log loss.

External Resource(s) (30 minutes)

`wiki.fast.ai` has some really nice resources on a number of topics. They have a nice concise writeup that explains the concept of log loss. We ask that you read about log loss on NB so you can take advantage of some notes from us to help guide your reading and add your own notes as well. If you want the original page (e.g., to click on the links), you can access the log loss page on wiki.fast.ai.

Exercise 2

TODO Here are some questions to test your understanding of the log loss reading.

4 Logistic Regression (top-down)

Now that we have built up some understanding of how probabilities can be used as a way of quantifying confidence in predictions, you are ready to learn about the logistic regression algorithm.

As always, we assume we are given a training set of inputs and outputs. As in linear regression we will assume that each of our inputs is a d-dimensional vector $\mathbf{x_i}$ and since we are dealing with binary classification, the outputs, y_i, will each be binary numbers (indicating whether the corresponding input is of class 0 or 1). Our hypothesis functions, $\hat{f}$, output the probability that a given input has a corresponding output of 1. What's cool is that we can borrow a lot of what we did in the linear regression case. In fact, all we're going to do in order to make sure that that output of $\hat{f}$ is between 0 and 1 is pass $\mathbf{w}^\top\mathbf{x}$ through a function that "squashes" its input so that the output is between 0 and 1. This idea is shown graphically in Figure 1

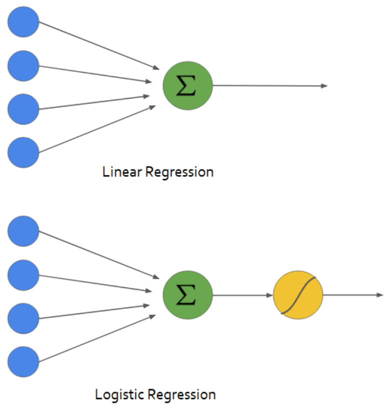


Figure 1: Graphical representation of both linear and logistic regression. The key difference is the application of the squashing function shown in yellow. Original source.

To make the intuition we've described concrete, we define each $\hat{f}$ to have the following form (note: this equation looks daunting. We have some tips for interpreting it below).

$$\begin{aligned}\hat{f}(\mathbf{x}) &= \text{probability that } y \text{ is 1 for } \mathbf{x} \\ &= \frac{1}{1+e^{-\mathbf{w}^\top\mathbf{x}}}\end{aligned} \tag{3}$$

Here are a few things to notice about this equation:

1. The weight vector that we saw in linear regression, $\mathbf{w}$, has made a comeback. As we said before, we are using the dot product between $\mathbf{x}$ and $\mathbf{w}$ (which creates a weighted sum of the x_i's) (just as we did in linear regression)!

2. As indicated in the picture above, the dot product $\mathbf{w}^\top\mathbf{x}$ has been passed through a squashing function known as the sigmoid function. The graph of $\sigma(u) = \frac{1}{1+e^{-u}}$ is shown in Figure 2. If we take $\sigma(\mathbf{w}^\top\mathbf{x})$ we will wind up with exactly what we have in Equation 3.

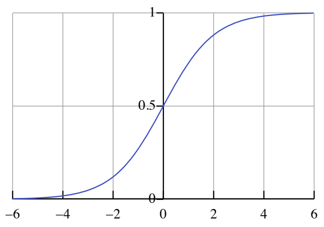


Figure 2: a graph of the sigmoid function $\frac{1}{1+e^{-x}}$.

4.1 Motivating Example: Sensor Networks

⚠ Notice

Pretty much every writeup of logistic regression contains a simple 2-feature ($d = 2$) example of logistic regression. Usually these examples involve things like predicting who would be approved for a credit card or who will be admitted to a college. We are going to do something a little messier and bit different. If you want the admission to college example, you can read about it in Building a Logistic Regression in Python.

External Resource(s) (TODO minutes)

Switch on over to the Assignment 3 Companion Notebook for a presentation of the motivating example.

5 *Putting it All Together: Deriving the Logistic Regression Learning Rule*

Let's summarize what we've done thus far in this assignment.

- We motivated the binary classification problem.
- We presented a particular useful loss function (log loss).
- We met the logistic regression model and tried it out on a real dataset.

Next, we're going to build on these pieces and formalize the logistic regression problem so we can derive a learning rule to solve it (i.e., compute the optimal weights). The formalization of logistic regression will combine Equation 3 with selected ℓ to be the log-loss to arrive at the following objective function.

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -y_i \ln \sigma(\mathbf{w}^\top \mathbf{x_i}) - (1-y_i)\ln(1-\sigma(\mathbf{w}^\top \mathbf{x_i})) \tag{4}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -y_i \ln\left(\frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x_i}}}\right) - (1-y_i)\ln\left(1-\frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x_i}}}\right) \quad \text{expanded out if you prefer this form} \tag{5}$$

Our typical recipe for finding $\mathbf{w}^\star$ has been to take the gradient of the expression inside the $\arg\min$, set it to 0, and solve for $\mathbf{w}^\star$. The last two steps will be a bit different for reasons that will become clear soon, but we will need to find the gradient. This will be the focus of the next parts.

5.1 *Useful Properties of the Sigmoid Function*

Looking at Equation 5 it looks really, really hairy! We see that in order to compute the gradient we will have to compute the gradient of $\mathbf{x}^\top\mathbf{w}$ with respect to $\mathbf{w}$ (we just wrapped our minds around this last assignment). Additionally, we will have to take into account how the application of the sigmoid function and the log function changes this gradient. In this section we'll learn some properties for manipulating the sigmoid function and computing its derivative.

Exercise 3 (60 minutes)

In this exercise you will be working to better understand some of the properties of the sigmoid function. Remember, the sigmoid function, σ, is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad . \tag{6}$$

(a) Show that $\sigma(-x) = 1 - \sigma(x)$.

(b) Show that the derivative of the logistic function $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$

5.2 Chain Rule for Gradients

We now know how to take derivatives of each of the major pieces of Equation 5. What we need is a way to put these derivatives together. You probably remember that in the case of single variable calculus you have just such a tool. This tool is known as the chain rule. The chain rule tells us how to compute the derivative of the composition of two single variable functions f and g.

$$\begin{aligned} h(x) &= g(f(x)) && \text{h(x) is the composition of } f \text{ with } g \\ h'(x) &= g'(f(x))f'(x) && \text{this is the chain rule!} \end{aligned} \tag{7}$$

Suppose that instead of the input being a scalar x, the input is now a vector, $\mathbf{w}$. In this case h takes a vector input and returns a scalar, f takes a vector input and returns a scalar, and g takes a scalar input and returns a scalar.

$$\begin{aligned} h(\mathbf{w}) &= g(f(\mathbf{w})) && \text{h}(\mathbf{w}) \text{ is the composition of } f \text{ with } g \\ \nabla_{\mathbf{w}} h(\mathbf{w}) &= g'(f(\mathbf{w}))\nabla_w f(\mathbf{w}) && \text{this is the multivariable chain rule} \end{aligned} \tag{8}$$

Exercise 4 (60 minutes)

(a) Suppose $h(x) = \sin(x^2)$, compute $h'(x)$ (x is a scalar so you can apply the single-variable chain rule).

(b) Compute the gradient of $h(\mathbf{v}) = (\mathbf{c}^\top \mathbf{v})^2$.

(c) Compute the gradient of the expression from Equation 5

$$\sum_{i=1}^{n} -y_i \ln \sigma(\mathbf{w}^\top \mathbf{x_i}) - (1-y_i)\ln\left(1 - \sigma(\mathbf{w}^\top \mathbf{x_i})\right) \quad . \tag{9}$$

You can either use the chain rule and the identities you learned about sigmoid, or expand everything out and work from that.

5.3 *Gradient Descent for Optimization*

If we were to follow our derivation of linear regression we would set our expression for the gradient to 0 and solve for $\mathbf{w}$. It turns out this equation will be difficult to solve due to the σ function. Instead, we can use an iterative approach to optimization where we start with some initial value of $\mathbf{w^0}$ and iteratively adjust it by moving down the gradient (the gradient represents the direction of fastest increase for our function, therefore, moving along the negative gradient is the direction where the loss is decreasing the fastest).

External Resource(s) (45 minutes)

There are tons of great resources that explain gradient descent with both math and compelling visuals. TODO: post these on NB

- Recommended: Gradient descent, how neural networks learn | Deep learning, chapter 2
- An Introduction to Gradient Descent
- The Wikipedia page on Gradient Descent (contains a nice visual)
- There are quite a few resources out there, do you have some suggestions? (suggest so on NB)

TODO: some understanding checks

If we take the logic of Gradient Descent and apply it to the logistic regression problem, we arrive at the following learning rule. Given some initial weights $\mathbf{w^0}$, and a learning rate η, we can iteratively update our weights using the formula below.

$$\begin{aligned}
\mathbf{w^{n+1}} &= \mathbf{w^n} - \eta \nabla \sum_{i=1}^{n} -y_i \ln \sigma(\mathbf{w}^\top \mathbf{x_i}) - (1-y_i)\ln(1-\sigma(\mathbf{w}^\top \mathbf{x_i})) \\
&= \mathbf{w^n} - \eta \sum_{i=1}^{n} -(y_i - \sigma(\mathbf{w}^\top \mathbf{x_i}))\mathbf{x_i} \quad \text{applying the result from exercise 4} \qquad (10) \\
&= \mathbf{w^n} + \eta \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^\top \mathbf{x_i}))\mathbf{x_i} \qquad (11)
\end{aligned}$$

This beautiful equation turns out to be provide us with the recipe we need to implement logistic regression.

6 *Reading on TODO*

TODO

7 *Suggestions for Additional Challenge*

We are asking folks for whom this is review to push themselves a bit farther. Based on the first two assignments, we thought that we could help out with some suggestions of our own. Please don't construe this as work that you have to do. None of the course content will assume that you've done any of these additional activities.

- Implement logistic regression using gradient descent. You will need to search for a good learning rate or you may consider implementing some strategies for automatically tuning the learning rate.
- Some additional reading?

Open in Colab

Analysis of Data from a Sensor Network to Determine Room Occupancy

A wireless sensor network consists of a bunch of sensors (e.g., light meters, barometers, microphones, cameras) that send data wirelessly throughout a network. Oftentimes, the sensor data is aggregated on a central server where algorithms may be running to make sense of what's happening in the environment. While this sounds an awful lot like surveillance, the data can also be used for less nefarious purposes. For instance, we may be able to optimize energy usage in a building of we know whether particular rooms are occupied (e.g., by adjusting climate control systems). It may be difficult to design a system to process the raw sensor data and convert it into actionable information (such as the occupancy of various rooms). Machine learning can help by automatically determining such information from a training set.

As a quick example of this sort of problem, we downloaded the occupancy detection dataset from the UCI Machine Learning repository (note: if you're interested, you can read about the original analysis of the data. The dataset consists of 20,560 data instances each with the following information:

- date time year-month-day hour:minute:second*
- Temperature, in Celsius
- Relative Humidity, %
- Light, in Lux
- CO2, in ppm
- Humidity Ratio, Derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air
- Occupancy, 0 or 1, 0 for not occupied, 1 for occupied status

Predicting Occupancy (Examining Single Variables)

In this example, we're going to see if we can predict whether there is someone in the room using the data above.

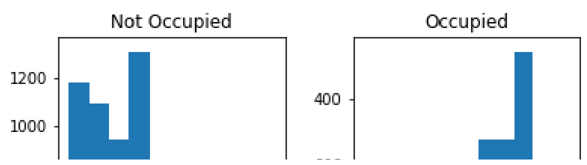
To get started, we'll read the data and create some plots that show the values that some of the features take on when the occupancy is either 0 or 1.

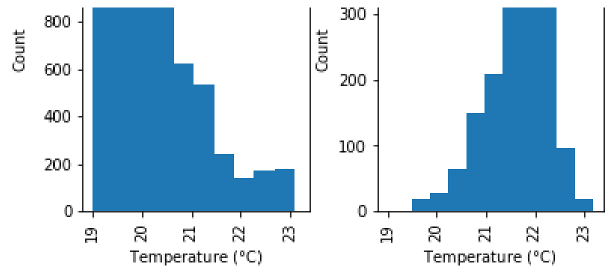
In [0]:

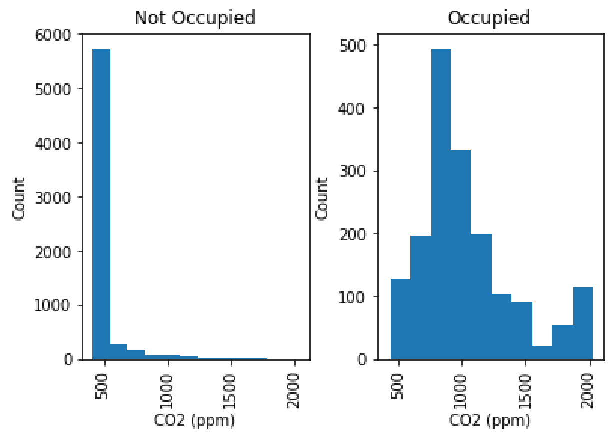
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
df = pd.read_csv('https://drive.google.com/uc?export=download&id=1DX5L9-e7I5B18RW2-
DFTORhk4O27WyR0')
# For simplicity we'll just look at these four columns
df = df[['CO2','Light', 'Humidity', 'Temperature', 'Occupancy']]
```

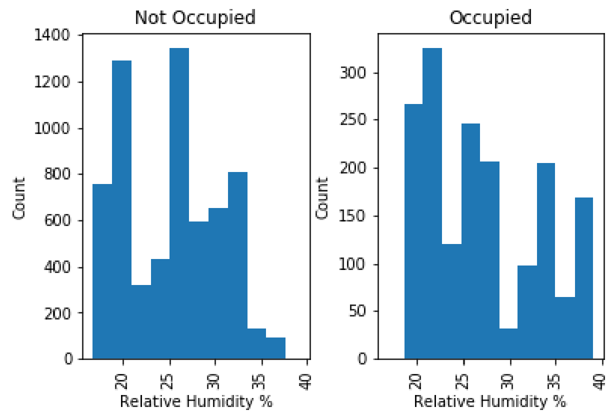
In [2]:

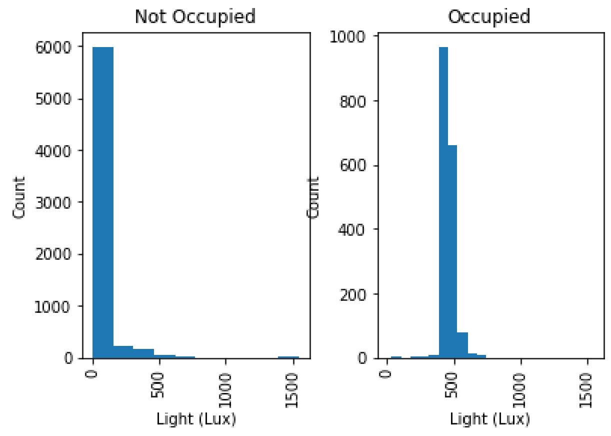
```
import matplotlib.pyplot as plt
%matplotlib inline

def make_dual_histogram(df, column, xlabel):
    subplots = df.hist(by='Occupancy', column = column, sharex=True)
    [subplot.set_xlabel(xlabel) for subplot in subplots]
    [subplot.set_ylabel('Count') for subplot in subplots]
    subplots[0].set_title('Not Occupied')
    subplots[1].set_title('Occupied')
    plt.show()

make_dual_histogram(df, 'Temperature', 'Temperature ($\degree$C)')
make_dual_histogram(df, 'CO2', 'CO2 (ppm)')
make_dual_histogram(df, 'Humidity', 'Relative Humidity %')
make_dual_histogram(df, 'Light', 'Light (Lux)')
```











Notebook Exercise 1

The plots shown above are known as histograms. They specify the count of the number of times a value in a particular range was seen in the dataset. The histograms in the left column correspond to the case where *the room was not* occupied and those on the right correspond to the case where *the room was* occupied. For example, if you look at the bottom row of plots, they tell us that were about $1,000$ instances where the room was occupied and the light value was just below $5,000$. In contrast, there were only about 100 instances where the light value was just below $5,000$.

Based on the plots above, which of these features looks like it will be useful for building a model to determine when the room is occupied? Are there any features that look like they will not be very useful? Justify your answers.

Predicting Occupancy With Logistic Regression

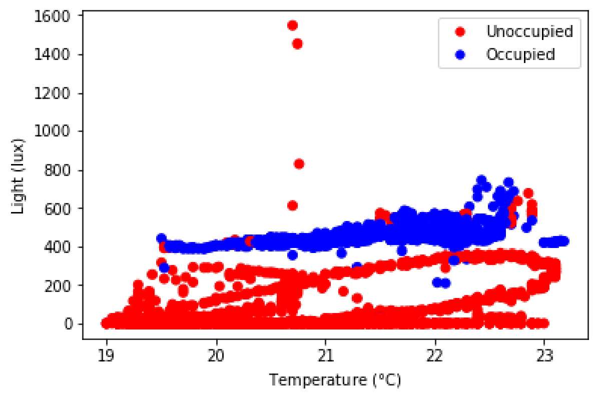
Now that we've looked at the variables one-by-one (always a good idea), we're going to use logistic regression to create a predictor for occupancy that considers multiple variables.

As a first cut, let's try to create a model that considers two of the independent variables we examined previously. We'll start out by looking at using temperature and light to predict occupancy.

Before we actually fit the model, we'll create a scatter plot so you can see how the occupancy status varies across both light and temperature.

In [3]:

```
from matplotlib.lines import Line2D

plt.scatter(df['Temperature'], df['Light'], c=df['Occupancy'].map(lambda x: 'b' if x else 'r'))
plt.xlabel('Temperature ($\degree$C)')
plt.ylabel('Light (lux)')

custom_lines = [Line2D([0], [0], marker='.', label='Unoccupied',
                       markerfacecolor='r', color='w', markersize=15),
                Line2D([0], [0], marker='.', label='Occupied',
                       markerfacecolor='b', color='w', markersize=15)]

plt.legend(custom_lines, ['Unoccupied', 'Occupied'])

plt.show()
```



Recall that logistic regression models the probability of the output being 1 (which in our problem corresponds to the room being occupied) as:

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

If we think of $\mathbf{x}$ as consisting of $x_1 = \text{temperature}$, $x_2 = \text{light}$, and $x_3 = 1$ then the equation becomes:

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-(w_1 \times \text{temperature} + w_2 \times \text{light} + w_3)}}$$

In order to determine $\mathbf{w}$ we will be using the logistic regression algorithm. Later in the assignment document we will precisely define the objective that is optimized by the logistic regression algorithm. Here, we can simply think of logistic regression as trying to find a line that best divides the blue points from the red points in the figure above. For now, we'll use an off-the-shelf implementation of logistic regression that is built into scikit learn.

The output shows the learned weights (w_1, w_2, w_3) along with the accuracy of the model on both a training and a test set (50% of the data was used for training and 50% was used for test).

In [4]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split


def fit_model(columns_to_use):
    # Note: this is might not really be a fair way to do a training / testing split
    # since it will result in instances that were very close in time falling in
    # both the training and testing set
    X_train, X_test, y_train, y_test = \
        train_test_split(df[columns_to_use], df['Occupancy'])

    model = LogisticRegression()
    model.fit(X_train, y_train)
    for i, c in enumerate(columns_to_use):
        print('w_%d (the weight for temperature) = %f' % (i+1, model.coef_[0, i]))

    print('Accuracy on training set', (model.predict(X_train) == y_train).mean())
    print('Accuracy on testing set', (model.predict(X_test) == y_test).mean())
    return model

fit_model(['Temperature', 'Light']);
```

```
w_1 (the weight for temperature) = -0.417291
w_2 (the weight for temperature) = 0.025623
Accuracy on training set 0.9882102505321761
Accuracy on testing set 0.9872298624754421
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Notebook Exercise 2

(a) Interpret the values for the fitted weights w_1 and w_2 (w_3 tricky to interpret in the way we are currently using the data. When we talk about normalization, you'll learn how to interpret the bias term). What do they mean interms of how the model would make predictions on new data?

(b) For a given temperature, $temp$, and light value, l, what would the model say is the probability of the room being occupied.

(c) Try different combinations of features. In terms of accuracy, what seems to be the best model?

Predicting Occupancy (Visualizing the Model)

To put a cap on this example, we're now going to visualize the model predictions as a function of the model inputs. This visualization should further reinforce how logistic regression maps from inputs to the probability of the output being 1. The visualization shows the probabilities (represented as a heat map) for various temperature / light combinations. Shown for convenience is the data used to create the model.

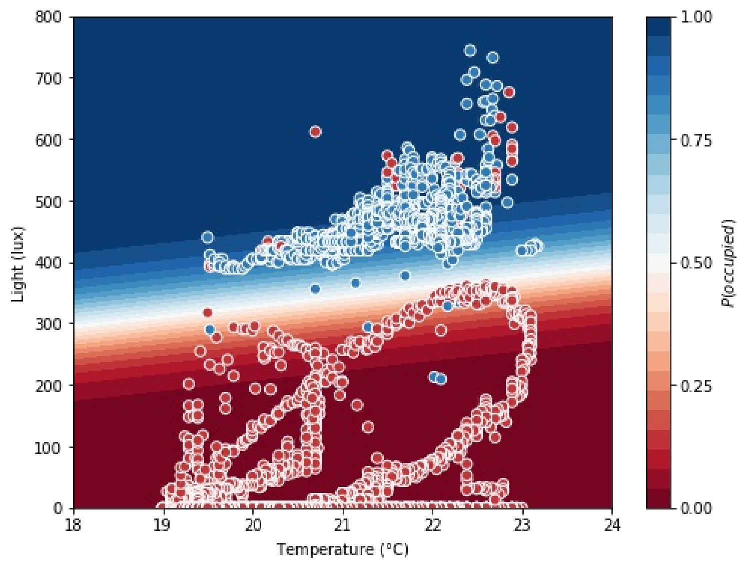
In [5]:

```
import numpy as np

model = fit_model(['Temperature', 'Light']);

xx, yy = np.mgrid[18:24:.01, 0:800:1]
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model.predict_proba(grid)[:, 1].reshape(xx.shape)

f, ax = plt.subplots(figsize=(8, 6))
contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",
                      vmin=0, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label("$P(occupied)$")
ax_c.set_ticks([0, .25, .5, .75, 1])

ax.scatter(df['Temperature'], df['Light'], c=df['Occupancy'], s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

ax.set(xlim=(18, 24), ylim=(0, 800),
       xlabel="Temperature ($\degree$C)", ylabel="Light (lux)")
```

```
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
w_1 (the weight for temperature) = -0.432152
w_2 (the weight for temperature) = 0.026388
Accuracy on training set 0.9885377435729491
Accuracy on testing set 0.9862475442043221
```



Notebook Exercise 3

Explain how the weights affect the visualization above. You might consider explaining how the signs of the weights influence the orientation of the lines of equal probability.