Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 19

A gentle introduction to classification

Overview

- Writing formal notation
- Using logistic regression
- Working with a confusion matrix
- Understanding multiclass classification

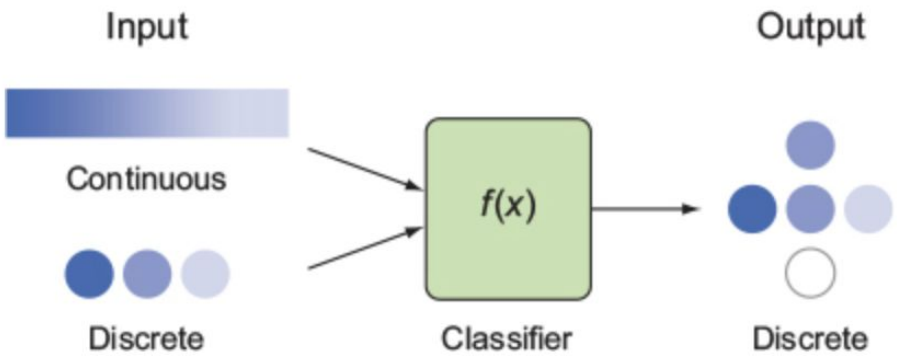
Overview

Classifiers

Type	Pros	Cons
Linear regression	Simple to implement	Not guaranteed to work Supports only binary labels
Logistic regression	Highly accurate Flexible ways to regularize model for custom adjustment Model responses are measures of probability Easy-to-update model with new data	Supports only binary labels
Softmax regression	Supports multiclass classification Model responses are measures of probability	More complicated to implement

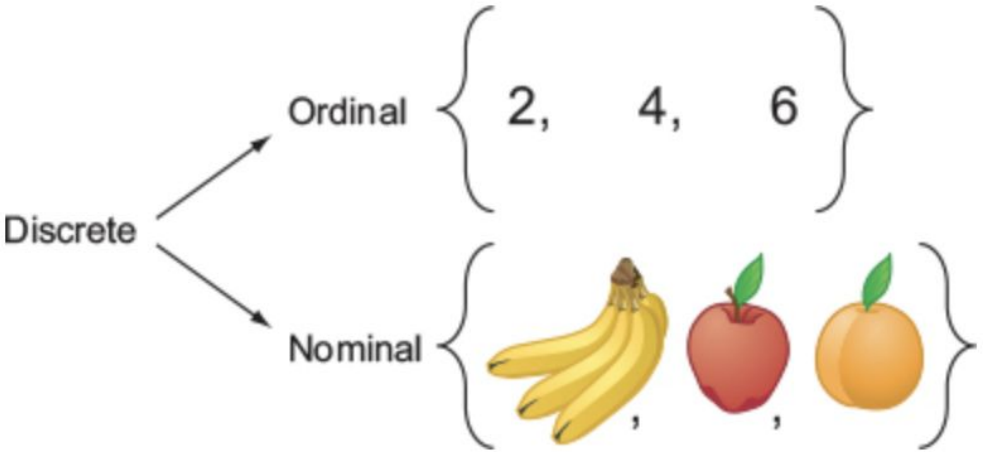
Formal notation

A classifier produces discrete outputs but may take either continuous or discrete inputs.



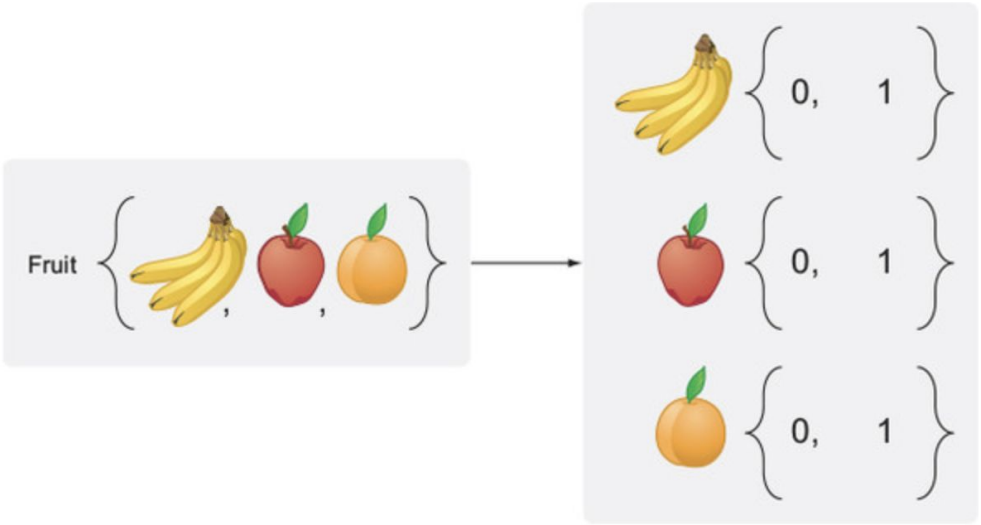
Formal notation

There are two types of discrete sets: those with values that can be ordered (ordinal) and those with values that can’t (nominal).



Formal notation

If the values of a variable are nominal, they might need to be preprocessed. One solution is to treat each nominal value as a Boolean variable, as shown on the right: banana, apple, and orange are three newly added variables, each having a value of 0 or 1. The original fruit variable is removed.



Formal notation

Exercise 1: Is it a better idea to treat each of the following as a regression or classification task? (a) Predicting stock prices; (b) Deciding which stocks you should buy, sell, or hold; (c) Rating the quality of a computer on a 1–10 scale

Formal notation

Exercise 1: Is it a better idea to treat each of the following as a regression or classification task? (a) Predicting stock prices; (b) Deciding which stocks you should buy, sell, or hold; (c) Rating the quality of a computer on a 1–10 scale

ANSWER: (a) Regression, (b) Classification, (c) Either

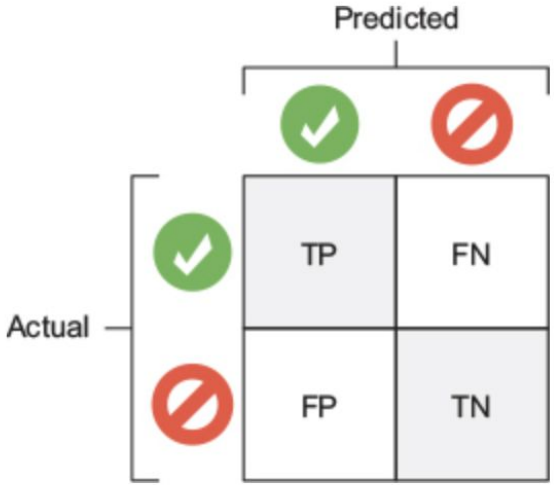
Measuring performance

Accuracy

$$accuracy = \frac{\#correct}{\#total}$$

Measuring performance

You can compare predicted results to actual results by using a matrix of positive (green check mark) and negative (red forbidden) labels.



Measuring performance

The ratio of true positives to total positive examples is called precision.

$$precision = \frac{TP}{TP + FP}$$

The ratio of true positives to all possible positives is called recall.

$$recall = \frac{TP}{TP + FN}$$

Measuring performance

An example of a confusion matrix for evaluating the performance of a classification algorithm

Confusion matrix		Predicted	
		Cat	Dog
Actual	Cat	30 True positives	20 False positives
	Dog	10 False negatives	40 True negatives

Measuring performance

Exercise 4.2: What are the precision and recall for cats? What's the accuracy of the system?

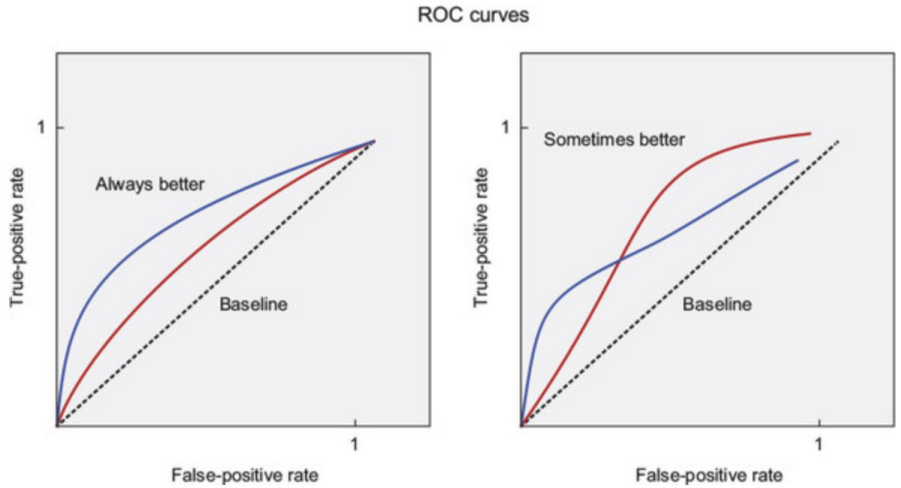
Measuring performance

Exercise 4.2: What are the precision and recall for cats? What's the accuracy of the system?

ANSWER: For cats, the precision is 30 / (30 + 20) or 3/5. The recall is 30 / (30 + 10), or 3/4. The accuracy is (30 + 40) / 100, or 70%.

Measuring performance

The principled way to compare algorithms is by examining their ROC curves. When the true-positive rate is greater than the false-positive rate in every situation, it's straightforward to declare that one algorithm is dominant in terms of its performance. If the true-positive rate is less than the false-positive rate, the plot dips below the baseline shown by the dotted line.



Measuring performance

Exercise 3: How would a 100% correct rate (all true positives, no false positives) look as a point on an ROC curve?

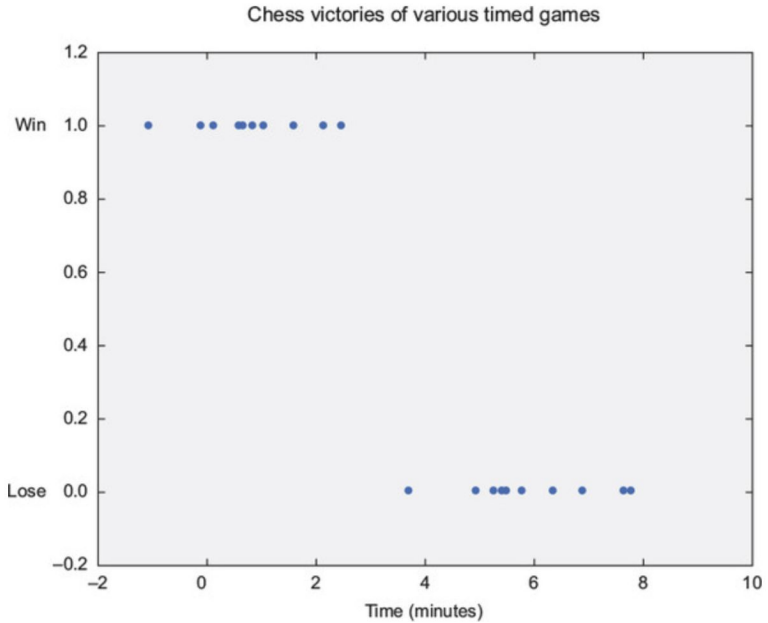
Measuring performance

Exercise 3: How would a 100% correct rate (all true positives, no false positives) look as a point on an ROC curve?

ANSWER: The point for a 100% correct rate would be located on the positive y-axis of the ROC curve.

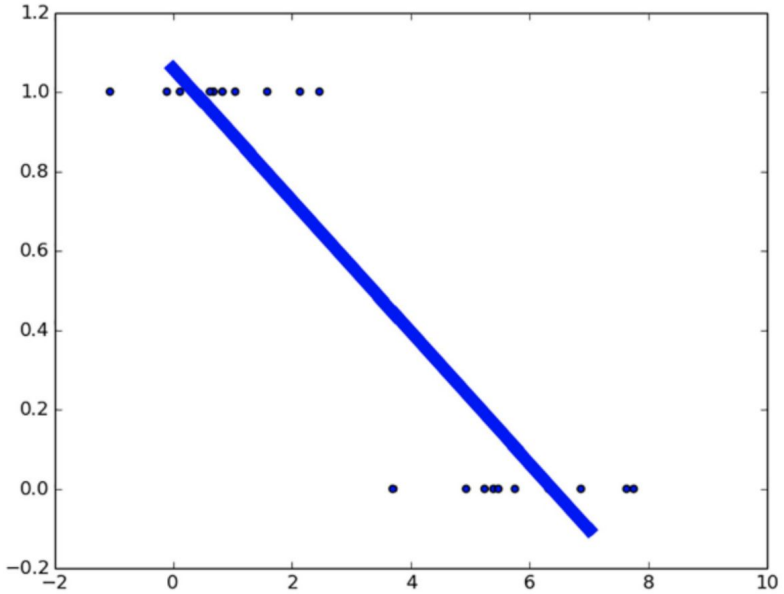
Using linear regression for classification

A visualization of a binary classification training dataset. The values are divided into two classes: all points where y = 1, and all points where y = 0.



Using linear regression for classification

The diagonal line is the best-fit line on a classification dataset. Clearly, the line doesn't fit the data well, but it provides an imprecise approach for classifying new data.



Using linear regression for classification

Exercise 4: What are the disadvantages of using linear regression as a tool for classification?

ANSWER: Linear regression is sensitive to outliers in your data, so it isn’t an accurate classifier.

Using linear regression for classification

Using linear regression for classification

```
import tensorflow as tf                                          1
import numpy as np                                               1
import matplotlib.pyplot as plt                                  1

x_label0 = np.random.normal(5, 1, 10)                            2
x_label1 = np.random.normal(2, 1, 10)                            2
xs = np.append(x_label0, x_label1)                               2
labels = [0.] * len(x_label0) + [1.] * len(x_label1)             3

plt.scatter(xs, labels)                                          4

learning_rate = 0.001                                            5
training_epochs = 1000                                           5
X = tf.placeholder("float")                                      6
Y = tf.placeholder("float")                                      6

def model(X, w):                                                 7
    return tf.add(tf.multiply(w[1], tf.pow(X, 1)),               7
                  tf.multiply(w[0], tf.pow(X, 0)))               7

w = tf.Variable([0., 0.], name="parameters")                     8
y_model = model(X, w)                                            9
cost = tf.reduce_sum(tf.square(Y-y_model))                       10
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) 11
```

1 Imports TensorFlow for the core learning algorithm, NumPy for manipulating data, and matplotlib for visualizing
2 Initializes fake data, 10 instances of each label
3 Initializes the corresponding labels
4 Plots the data
5 Declares the hyperparameters
6 Sets up the placeholder nodes for the input/output pairs
7 Defines a linear y = w1 * x + w0 model
8 Sets up the parameter variables
9 Defines a helper variable, because you'll refer to this multiple times
10 Defines the cost function
11 Defines the rule to learn the parameters

Using linear regression for classification

Executing the graph

```
sess = tf.Session()                                          1
init = tf.global_variables_initializer()                        1
sess.run(init)                                             1

for epoch in range(training_epochs):                              2
    sess.run(train_op, feed_dict={X: xs, Y: labels})              2
    current_cost = sess.run(cost, feed_dict={X: xs, Y: labels})  3
    if epoch % 100 == 0:
        print(epoch, current_cost)                                4

w_val = sess.run(w)                                           5
print('learned parameters', w_val)                               5

sess.close()                                                 6

all_xs = np.linspace(0, 10, 100)                                 7
plt.plot(all_xs, all_xs*w_val[1] + w_val[0])                        7
plt.show()                                                   7
```

1 Opens a new session, and initializes the variables
2 Runs the learning operation multiple times
3 Records the cost computed with the current parameters
4 Prints out log info while the code runs
5 Prints the learned parameters
6 Closes the session when no longer in use
7 Shows the best-fit line

Using linear regression for classification

Measuring accuracy

```
correct_prediction = tf.equal(Y, tf.to_float(tf.greater(y_model, 0.5)))   1
accuracy = tf.reduce_mean(tf.to_float(correct_prediction))
2

print('accuracy', sess.run(accuracy, feed_dict={X: xs, Y: labels}))       3
```

1 When the model's response is greater than 0.5, it should be a positive label, and vice versa.
2 Computes the percent of success
3 Prints the success measure from provided input

The preceding code produces the following output:

```
('learned parameters', array([ 1.2816, -0.2171], dtype=float32))
('accuracy', 0.95)
```

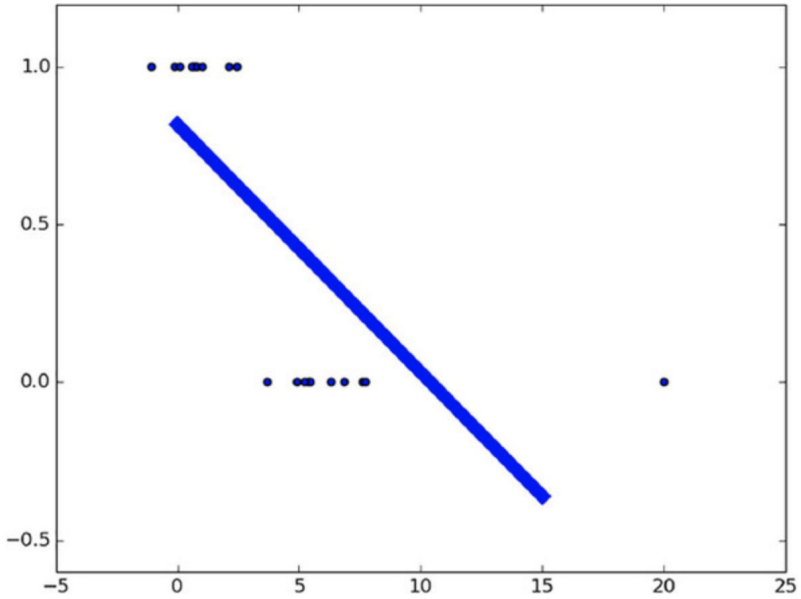
Using linear regression for classification

Linear regression failing miserably for classification

```
x_label0 = np.append(np.random.normal(5, 1, 9), 20)
```

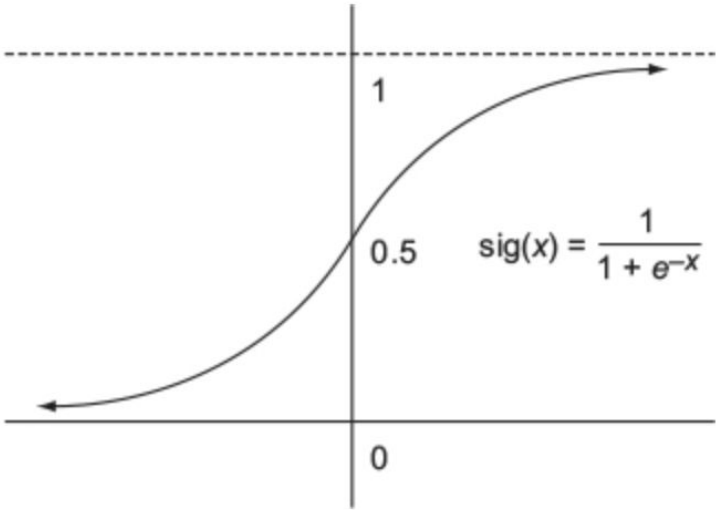
Using linear regression for classification

A new training element of value 20 greatly influences the best-fit line. The line is too sensitive to outlying data, and therefore linear regression is a sloppy classifier.



Using logistic regression

A visualization of the sigmoid function



Using logistic regression

The new cost function between the actual value y and model response h will be a two-part equation as follows:

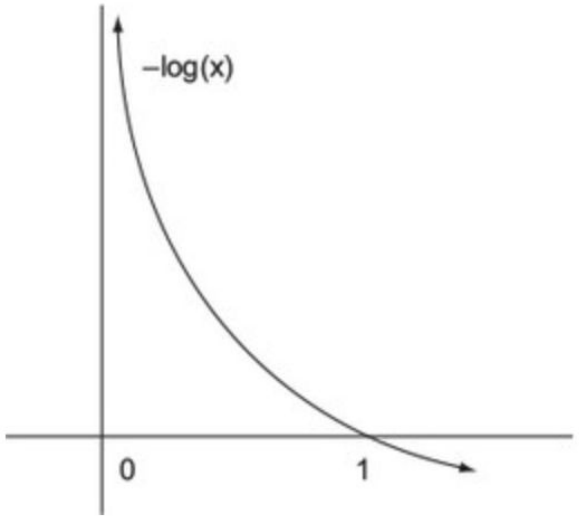
$$Cost(y, h) = \begin{cases} -\log(h), & \text{if } y = 1 \\ -\log(1 - h), & \text{if } y = 0 \end{cases}$$

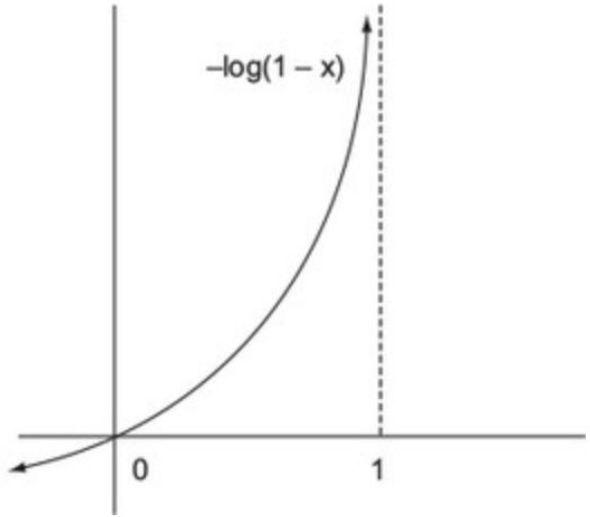
You can condense the two equations into one long equation:

$$Cost(y, h) = -y \log(h) - (1 - y)\log(1 - h)$$

Using logistic regression

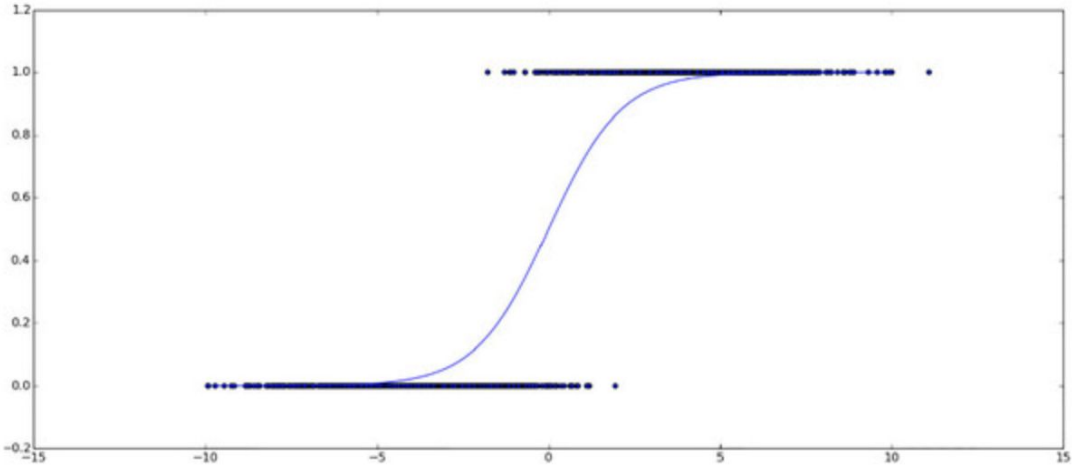
Here’s a visualization of how the two cost functions penalize values at 0 and 1. Notice that the left function heavily penalizes 0 but has no cost at 1. The right cost function displays the opposite phenomena. This loss is called entropy.





Using logistic regression

Here's a best-fit sigmoid curve for a binary classification dataset. Notice that the curve resides within y = 0 and y = 1. That way, this curve isn't that sensitive to outliers.



Using logistic regression

Using one-dimensional logistic regression

```
import numpy as np                                  1
import tensorflow as tf                            1
import matplotlib.pyplot as plt                       1
learning_rate = 0.01                               2
training_epochs = 1000                                2

def sigmoid(x):                                   3
    return 1. / (1. + np.exp(-x))                      3
```

1 Imports relevant libraries
2 Sets the hyperparameters
3 Defines a helper function to calculate the sigmoid function

Using logistic regression

```
x1 = np.random.normal(-4, 2, 1000)                    4
x2 = np.random.normal(4, 2, 1000)                     4
xs = np.append(x1, x2)                          4
ys = np.asarray([0.] * len(x1) + [1.] * len(x2))      4

plt.scatter(xs, ys)                             5
```

4 Initializes fake data
5 Visualizes the data

Using logistic regression

```
X = tf.placeholder(tf.float32, shape=(None,), name="x")                    6
Y = tf.placeholder(tf.float32, shape=(None,), name="y")                    6
w = tf.Variable([0., 0.], name="parameter", trainable=True)                7
y_model = tf.sigmoid(w[1] * X + w[0])                                      8
cost = tf.reduce_mean(-Y * tf.log(y_model) - (1 - Y) * tf.log(1 –
y_model))9

train_op =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) 10
```

6 Defines the input/output placeholders
7 Defines the parameter node
8 Defines the model using TensorFlow's sigmoid function
9 Defines the cross-entropy loss function
10 Defines the minimizer to use

Using logistic regression

```
with tf.Session() as sess:                                    11
    sess.run(tf.global_variables_initializer())                   11
    prev_err = 0                                          12
    for epoch in range(training_epochs):                          13
        err, _ = sess.run([cost, train_op], {X: xs, Y: ys})       14
        print(epoch, err)
        if abs(prev_err - err) < 0.0001:                          15
            break
        prev_err = err                                        16
    w_val = sess.run(w, {X: xs, Y: ys})                           17
```

11 Opens a session, and defines all variables
12 Defines a variable to keep track of the previous error
13 Iterates until convergence or until the maximum number of epochs is reached
14 Computes the cost, and updates the learning parameters
15 Checks for convergence—if you're changing by < .01% per iteration, you're done
16 Updates the previous error value
17 Obtains the learned parameter value

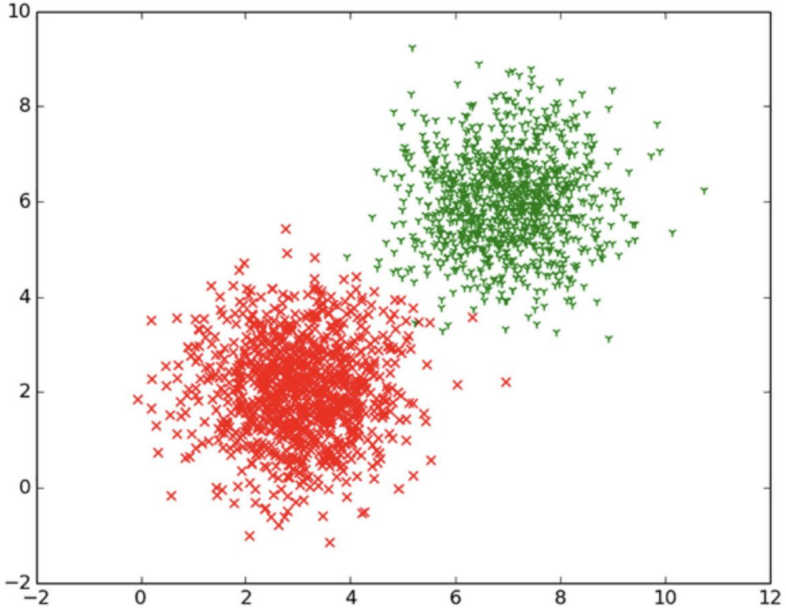
Using logistic regression

```
all_xs = np.linspace(-10, 10, 100)                                  18
plt.plot(all_xs, sigmoid((all_xs * w_val[1] + w_val[0])))                  18
plt.show()                                                    18
```

18 Plots the learned sigmoid function

Using logistic regression

The x-axis and y-axis represent the two independent variables. The dependent variable holds two possible labels, represented by the shape and color of the plotted points.



Using logistic regression

Setting up data for two-dimensional logistic regression

```
import numpy as np                                          1
import tensorflow as tf                                     1
import matplotlib.pyplot as plt                             1

learning_rate = 0.1                                         2
training_epochs = 2000                                      2

def sigmoid(x):                                             3
    return 1. / (1. + np.exp(-x))                           3

x1_label1 = np.random.normal(3, 1, 1000)                    4
x2_label1 = np.random.normal(2, 1, 1000)                    4
x1_label2 = np.random.normal(7, 1, 1000)                    4
x2_label2 = np.random.normal(6, 1, 1000)                    4
x1s = np.append(x1_label1, x1_label2)                       4
x2s = np.append(x2_label1, x2_label2)                       4
ys = np.asarray([0.] * len(x1_label1) + [1.] * len(x1_label2))      4
```

1 Imports relevant libraries
2 Sets the hyperparameters
3 Defines a helper sigmoid function
4 Initializes fake data

Using logistic regression

You have two independent variables ($x1$ and $x2$). A simple way to model the mapping between the input x's and output $M(x)$ is the following equation, where w is the parameter to be found using TensorFlow:

$$M(x; w) = sig(w_2x_2 + w_1x_1 + w_0)$$

Using logistic regression

Using TensorFlow for multidimensional logistic regression

```
X1 = tf.placeholder(tf.float32, shape=(None,), name="x1")                1
X2 = tf.placeholder(tf.float32, shape=(None,), name="x2")                1
Y = tf.placeholder(tf.float32, shape=(None,), name="y")                  1
w = tf.Variable([0., 0., 0.], name="w", trainable=True)                  2

y_model = tf.sigmoid(w[2] * X2 + w[1] * X1 + w[0])                       3
cost = tf.reduce_mean(-tf.log(y_model * Y + (1 - y_model) * (1 - Y)))    4
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
4
with tf.Session() as sess:                                               5
    sess.run(tf.global_variables_initializer())                          5
    prev_err = 0                                                         5
    for epoch in range(training_epochs):                                 5
        err, _ = sess.run([cost, train_op], {X1: x1s, X2: x2s, Y: ys})   5
        print(epoch, err)                                                5
        if abs(prev_err - err) < 0.0001:                                 5
            break                                                        5
        prev_err = err                                                   5
    w_val = sess.run(w, {X1: x1s, X2: x2s, Y: ys})                       6
```

1 Defines the input/output placeholder nodes
2 Defines the parameter node
3 Defines the sigmoid model using both input variables
4 Defines the learning step
5 Creates a new session, initializes variables, and learns parameters until convergence
6 Obtains the learned parameter value before closing the session

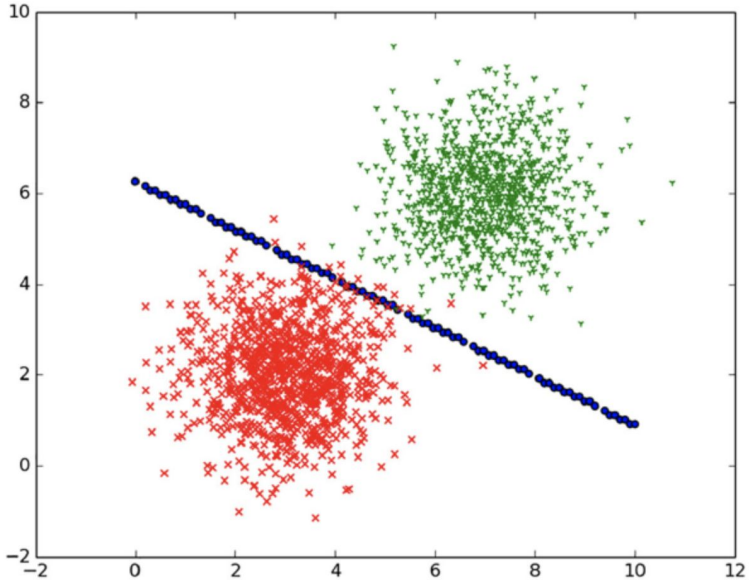
Using logistic regression

```
x1_boundary, x2_boundary = [], []                                   7
for x1_test in np.linspace(0, 10, 100):                             8
    for x2_test in np.linspace(0, 10, 100):                         8
        z = sigmoid(-x2_test*w_val[2] - x1_test*w_val[1] - w_val[0])      9
        if abs(z - 0.5) < 0.01:                                     9
            x1_boundary.append(x1_test)                             9
            x2_boundary.append(x2_test)                             9

plt.scatter(x1_boundary, x2_boundary, c='b', marker='o', s=20)          10
plt.scatter(x1_label1, x2_label1, c='r', marker='x', s=20)              10
plt.scatter(x1_label2, x2_label2, c='g', marker='1', s=20)              10
                                                                    10
plt.show()                                                          10
```

7 Defines arrays to hold boundary points
8 Loops through a window of points
9 If the model response is close the 0.5, updates the boundary points
10 Shows the boundary line along with the data

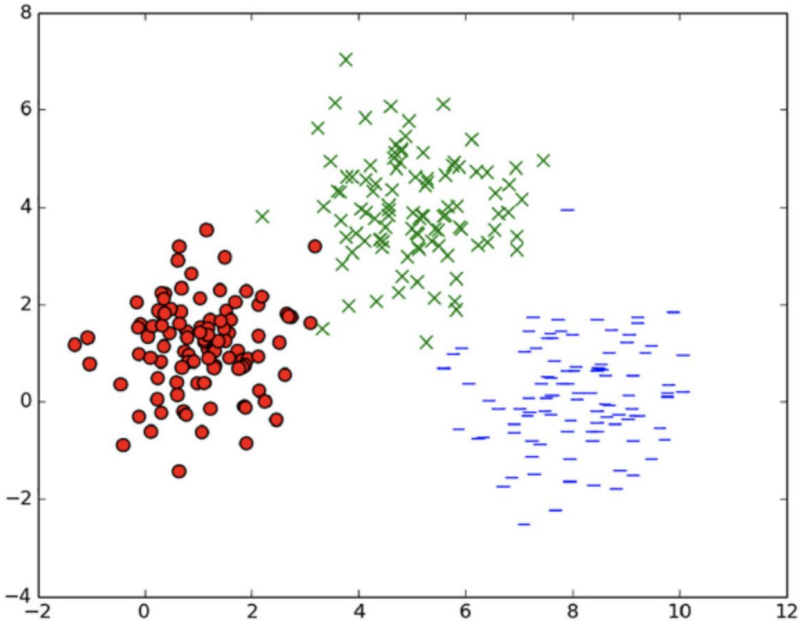
Using logistic regression

The diagonal dotted line represents when the probability between the two decisions is split equally. The confidence of making a decision increases as data lies farther away from the line.



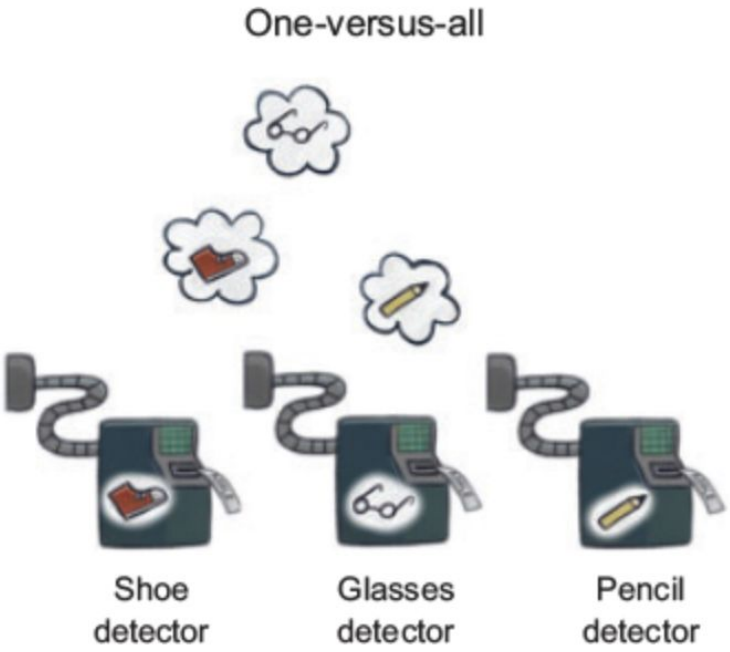
Multiclass classifier

The independent variable is two-dimensional, indicated by the x-axis and y-axis. The dependent variable can be one of three labels, shown by the color and shape of the data points.



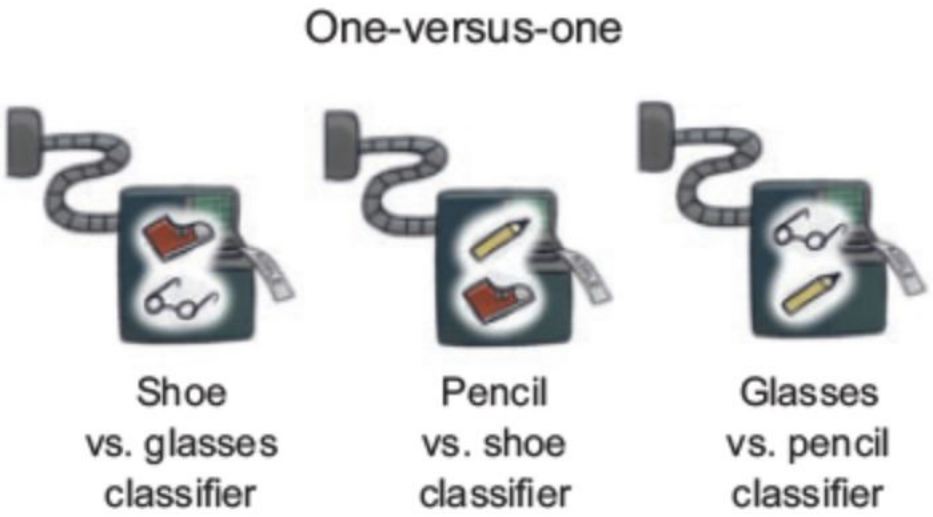
Multiclass classifier

One-versus-all is a multiclass classifier approach that requires a detector for each class.



Multiclass classifier

In one-versus-one multiclass classification, there's a detector for each pair of classes.



Multiclass classifier

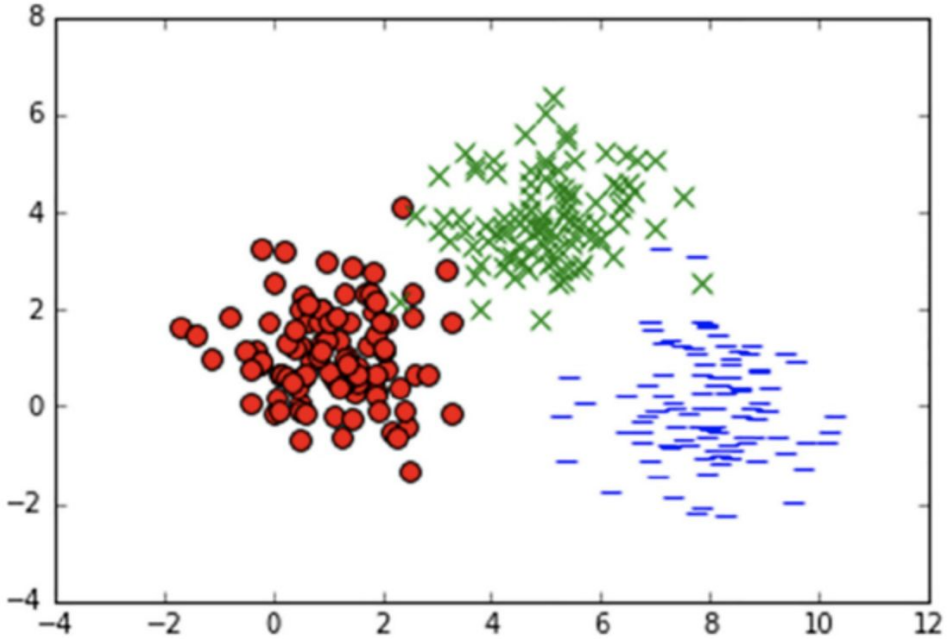
Visualizing multiclass data

```
import numpy as np                                          1
import matplotlib.pyplot as plt                              1

x1_label0 = np.random.normal(1, 1, (100, 1))                    2
x2_label0 = np.random.normal(1, 1, (100, 1))                    2
x1_label1 = np.random.normal(5, 1, (100, 1))                    3
x2_label1 = np.random.normal(4, 1, (100, 1))                    3
x1_label2 = np.random.normal(8, 1, (100, 1))                    4
x2_label2 = np.random.normal(0, 1, (100, 1))                    4

plt.scatter(x1_label0, x2_label0, c='r', marker='o', s=60)        5
plt.scatter(x1_label1, x2_label1, c='g', marker='x', s=60)        5
plt.scatter(x1_label2, x2_label2, c='b', marker='_', s=60)        5
plt.show()                                                  5
```

1 Imports NumPy and matplotlib
2 Generates points near (1, 1)
3 Generates points near (5, 4)
4 Generates points near (8, 0)
5 Visualizes the three labels on a scatter plot

Multiclass classifier

2D training data for multi-output classification



Multiclass classifier

Exercise 5

One-hot encoding might appear to be an unnecessary step. Why not just have a one-dimensional output with values of 1, 2, and 3 representing the three classes?

Multiclass classifier

Exercise 5

One-hot encoding might appear to be an unnecessary step. Why not just have a one-dimensional output with values of 1, 2, and 3 representing the three classes?

ANSWER

Regression may induce a semantic structure in the output. If outputs are similar, regression implies that their inputs were also similar. If you use just one dimension, you're implying that labels 2 and 3 are more similar to each other than 1 and 3. You must be careful about making unnecessary or incorrect assumptions, so it's a safe bet to use one-hot encoding.

Multiclass classifier

Setting up training and test data for multiclass classification

```
xs_label0 = np.hstack((x1_label0, x2_label0))
1
xs_label1 = np.hstack((x1_label1, x2_label1))
1
xs_label2 = np.hstack((x1_label2, x2_label2))
1
xs = np.vstack((xs_label0, xs_label1, xs_label2))
1

labels = np.matrix([[1., 0., 0.]] * len(x1_label0) + [[0., 1., 0.]] *
    len(x1_label1) + [[0., 0., 1.]] * len(x1_label2))                  2

arr = np.arange(xs.shape[0])                                          3
np.random.shuffle(arr)                                              3
xs = xs[arr, :]                                                   3
labels = labels[arr, :]                                             3
```

1 Combines all input data into one big matrix
2 Creates the corresponding one-hot labels
3 Shuffles the dataset

Multiclass classifier

```
test_x1_label0 = np.random.normal(1, 1, (10, 1))                    4
test_x2_label0 = np.random.normal(1, 1, (10, 1))                    4
test_x1_label1 = np.random.normal(5, 1, (10, 1))                    4
test_x2_label1 = np.random.normal(4, 1, (10, 1))                    4
test_x1_label2 = np.random.normal(8, 1, (10, 1))                    4
test_x2_label2 = np.random.normal(0, 1, (10, 1))                    4
test_xs_label0 = np.hstack((test_x1_label0, test_x2_label0))
4
test_xs_label1 = np.hstack((test_x1_label1, test_x2_label1))
4
test_xs_label2 = np.hstack((test_x1_label2, test_x2_label2))
4

test_xs = np.vstack((test_xs_label0, test_xs_label1, test_xs_label2))
4
test_labels = np.matrix([[1., 0., 0.]] * 10 + [[0., 1., 0.]] * 10 + [[0., 0.,
    1.]] * 10)                                                      4

train_size, num_features = xs.shape                                 5
```

4 Constructs the test dataset and labels
5 The shape of the dataset tells you the number of examples and features per example.

Multiclass classifier

Exercise 6

Which of the following functions is continuous?

```
f(x) = x2
f(x) = min(x, 0)
f(x) = tan(x)
```

Multiclass classifier

Exercise 6

Which of the following functions is continuous?

```
f(x) = x2
f(x) = min(x, 0)
f(x) = tan(x)
```

ANSWER

The first two are continuous. The last one, tan(x), has periodic asymptotes, so there are some values for which there are no valid results.

Multiclass classifier

Using softmax regression

```
import tensorflow as tf

learning_rate = 0.01                                                    1
training_epochs = 1000                                                  1
num_labels = 3                                                          1
batch_size = 100                                                        1

X = tf.placeholder("float", shape=[None, num_features])                 2
Y = tf.placeholder("float", shape=[None, num_labels])                   2

W = tf.Variable(tf.zeros([num_features, num_labels]))                   3
b = tf.Variable(tf.zeros([num_labels]))                                 3
y_model = tf.nn.softmax(tf.matmul(X, W) + b)                            4

cost = -tf.reduce_sum(Y * tf.log(y_model))                              5
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) 5

correct_prediction = tf.equal(tf.argmax(y_model, 1), tf.argmax(Y, 1))   6
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))         6
```

1 Defines hyperparameters
2 Defines the input/output placeholder nodes
3 Defines the model parameters
4 Designs the softmax model
5 Sets up the learning algorithm
6 Defines an op to measure success rate

Multiclass classifier

Executing the graph

```
with tf.Session() as sess:                                          1
    tf.global_variables_initializer().run()                          1

    for step in range(training_epochs * train_size // batch_size):          2
        offset = (step * batch_size) % train_size                           3
        batch_xs = xs[offset:(offset + batch_size), :]                       3
        batch_labels = labels[offset:(offset + batch_size)]
        err, _ = sess.run([cost, train_op], feed_dict={X: batch_xs, Y:
     batch_labels})                                                          4
        print (step, err)                                                   5

    W_val = sess.run(W)                                                      6
    print('w', W_val)                                                       6
    b_val = sess.run(b)                                                      6
    print('b', b_val)                                                       6
    print("accuracy", accuracy.eval(feed_dict={X: test_xs, Y: test_labels}))7
```

1 Opens a new session and initializes all variables
2 Loops only enough times to complete a single pass through the dataset
3 Retrieves a subset of the dataset corresponding to the current batch
4 Runs the optimizer on this batch
5 Prints ongoing results
6 Prints the final learned parameters
7 Prints the success rate

Multiclass classifier

The final output of running the softmax regression algorithm on the dataset is the following:

```
('w', array([[-2.101, -0.021,  2.122],
             [-0.371,  2.229, -1.858]], dtype=float32))
('b', array([10.305, -2.612, -7.693], dtype=float32))
Accuracy 1.0
```

Application of classification

- Large Movie Review Dataset: http://mng.bz/60nj
- Sentiment Labelled Sentences Data Set: http://mng.bz/CzSM
- Twitter Sentiment Analysis Dataset: http://mng.bz/2M4d