Week 1 – Linear Classifiers

A classifier will take a set of inputs and use them to produce an output, which is a class. For instance, in sentiment analysis the classifier takes sentences and classifies them as either positive or negative sentiment.

In sentiment analysis, the input data are sentences. A linear classifier takes each word in the sentence and associates it with a weight (a coefficient) that describes how positive or negative that word is. Strongly positive words get large positive values, strongly negative words get large negative values and words that are not important get values close to zero. The classification of a sentence is simply the sum of the weights of each word in the sentence. If the sum is positive then the class is positive sentiment, if the sum is negative then the class is negative sentiment.

The weights of the words are trained from data. The data are sentences where each sentence is labeled as either positive sentiment or negative sentiment.

As with other machine learning workflows, we would break the data into a training and validation set. We learn the coefficients on the training set and test our estimate of generalization error on the validation set.

Decision Boundaries

For example, if we are classifying based on just two words (so all other words have zero weight) like this;

Word	Weight
awesome	+1.0
awful	-1.5

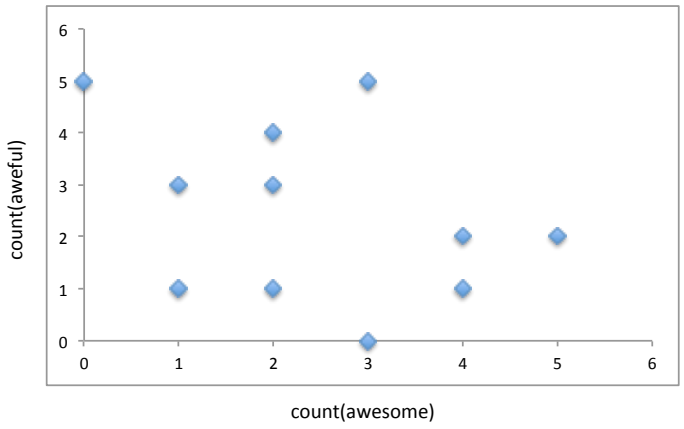
Then the score of any sentence x is given by

$$Score(x) = 1.0 \times count(awesome) - 1.5 \times count(awful)$$

We can also plot the sentence of a graph where the x axis is the number of awesomes in the sentence and the y axis is the number of awfuls in the sentence. Given this set of sentences and their counts;

	count(awesome)	count(awful)
sentence 1	0	5
sentence 2	1	3
sentence 3	2	4
sentence 4	2	3
sentence 5	3	5
sentence 6	1	1
sentence 7	4	2
sentence 8	3	0
sentence 9	4	1
sentence 10	5	2
sentence 11	2	1

We get this graph;



If we apply our decision function;

```
Score(x) = 1.0 x count(awesome) – 1.5 x count(awful)
```

to each sentence, and then determine our class as follows;

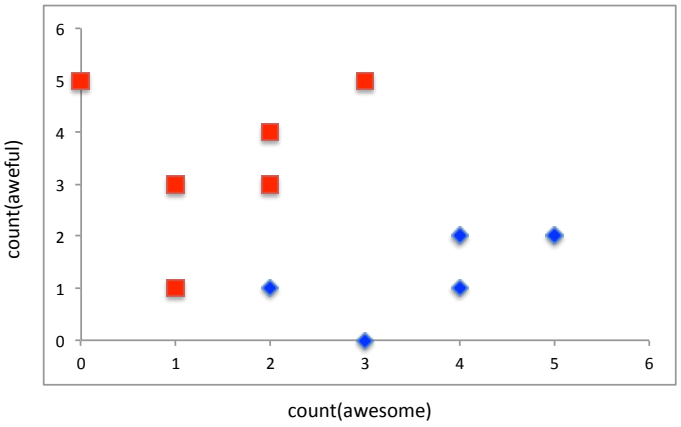
```
Class(x) = sign(Score(x))
```

we get this data;

score(x) = 1.0*count(awesome) - 1.5*count(awful)

	count(awesome)	count(awful)	score(x)	class
sentence 1	0	5	-7.5	-
sentence 2	1	3	-3.5	-
sentence 3	2	4	-4.0	-
sentence 4	2	3	-2.5	-
sentence 5	3	5	-4.5	-
sentence 6	1	1	-0.5	-
sentence 7	4	2	1.0	+
sentence 8	3	0	3.0	+
sentence 9	4	1	2.5	+
sentence 10	5	2	2.0	+
sentence 11	2	1	0.5	+

We can mark each point with it's class to show the distribution of positive and negative sentences in the space;



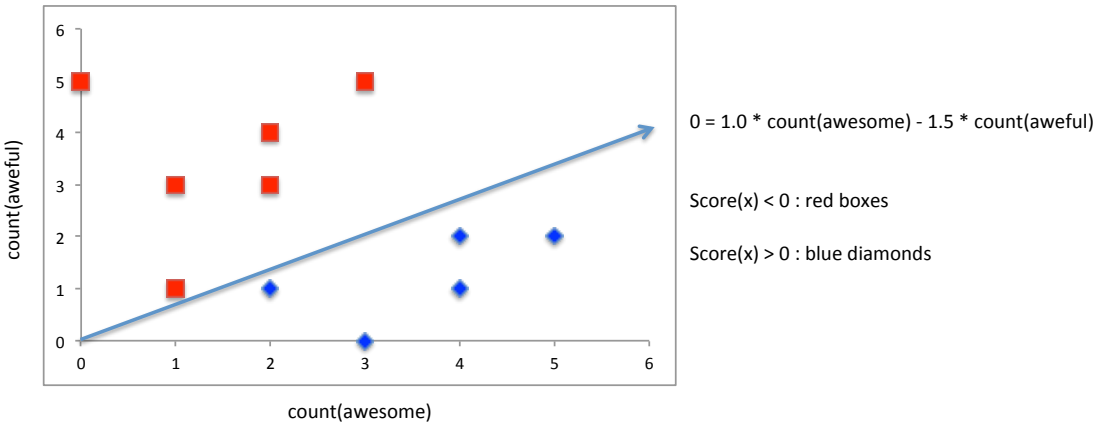
Score(x) < 0 : red boxes

Score(x) > 0 : blue diamonds

The class of each point (each sentence) is based on the sign of the score function. Setting the score function to zero shows where the positive and negative boundary is;

```
0 = 1.0 x count(awesome) – 1.5 x count(awful)
```

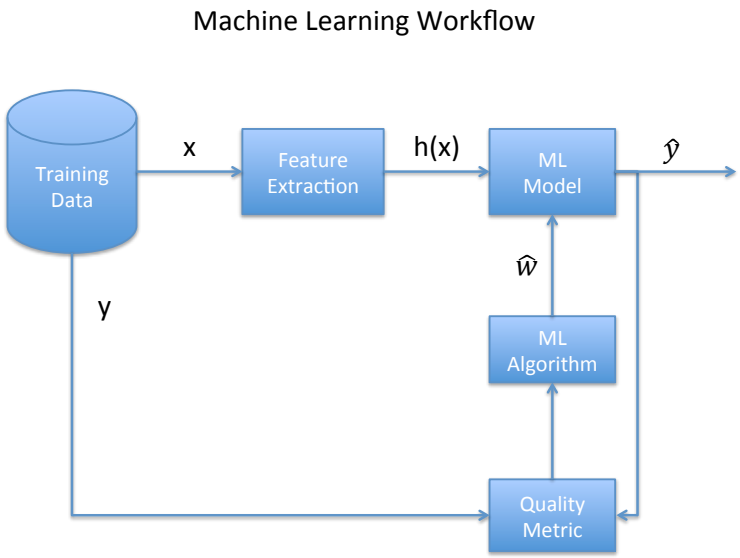
This forms a line (because we have two features). Above the line the class is negative, below the line the class is positive. This is the decision boundary for our two features; everywhere on this line the Score(x) = 0.



When we have 3 features, we have a plane that separates the positive from the negatives. If we have more than 3 features, we have a high dimensional hyperplane that separates the positives from the negatives.

Linear classifier model

Remember our Machine Learning Workflow;



In the case of sentiment analysis of text;

y	is the labeled (known) value of each sentence, in this case either positive or negative class, indicated by +1 and -1 respectively.
x	is the training data, in this case the sentences
h(x)	are the extracted features; in this case the word counts extracted from each sentence.
$\hat{y}$	is the predicted output value (the predicted class) of each sentence
$\hat{w}$	are the estimated coefficients (feature weights) for the model

These are used as follows

- We extract features from our training data
- We apply a model and set of coefficients to the data to produce a set of predicted values
- We compare the predicted values to known values using a quality metric.
- If the quality metric is not met, we use the results of the quality metric and the Machine Learning algorithm to calculate a new set of estimated coefficients for the model.
- Once the quality metric is met, we have our coefficients for prediction and we can use the model for prediction

General Notation
Notation used in the rest of this course for specifying the input data (like a sentence) and the features on the input data (like the word counts):

Output:	y	a scalar, it is the observed output value
Inputs:	$\vec{\boldsymbol{x}} = (\vec{\boldsymbol{x}}[1], \vec{\boldsymbol{x}}[2], \vec{\boldsymbol{x}}[3], \ldots, \vec{\boldsymbol{x}}[d])$	the input values a d-dimensional **vector** of scalar

$\vec{\boldsymbol{x}}[j]$ The j-th input value (a scalar) like a word in the sentence

$\overrightarrow{\boldsymbol{Bold}}$ is used to indicate a vector in the videos. In these notes I am indicating a vector using the arrow above the value.

Our data sets will have many output values and their associated input values. Given a dataset with N output values each associated with d input values;

$\vec{\boldsymbol{y}}$	The vector of N observed output values
$\vec{\boldsymbol{y_i}}$	The i-th observation in the data
$\vec{\boldsymbol{x_i}}$	The input vector for the i-th data point in the data set, like the i-th sentence in a dataset of sentences
$\vec{\boldsymbol{x_i}}[j]$	The j-th input of the i-th data point in the dataset, like the number of time the work awesome appears in the i-th sentence in the dataset
d	small-d represents the number of inputs, so it is the length of the input vector $\vec{\boldsymbol{x_i}}$.

Simple Hyperplane Model

$$\hat{y}_i = sign(Score(\boldsymbol{x_i}))$$

Where

$$Score(\boldsymbol{x_i}) = w_0 + w_1\vec{\boldsymbol{x}}_{\boldsymbol{i}}[1] + w_2\vec{\boldsymbol{x}}_{\boldsymbol{i}}[2] + \cdots w_d\vec{\boldsymbol{x}}_{\boldsymbol{i}}[d]$$

This can be simplified using vector notation to:

$$Score(\boldsymbol{x_i}) = \vec{w}^T\vec{\boldsymbol{x}}_{\boldsymbol{i}}$$

In sentiment analysis, our features might be

feature 1 = constant feature, like 1.
feature 2 = x[1], like number of times ‘awesome’ appears
feature 3 = x[2], like number of times ‘awful’ appears
…
feature d = x[d-1], like number of time ‘ramen’ appears

Effect of coefficient values on decision boundary

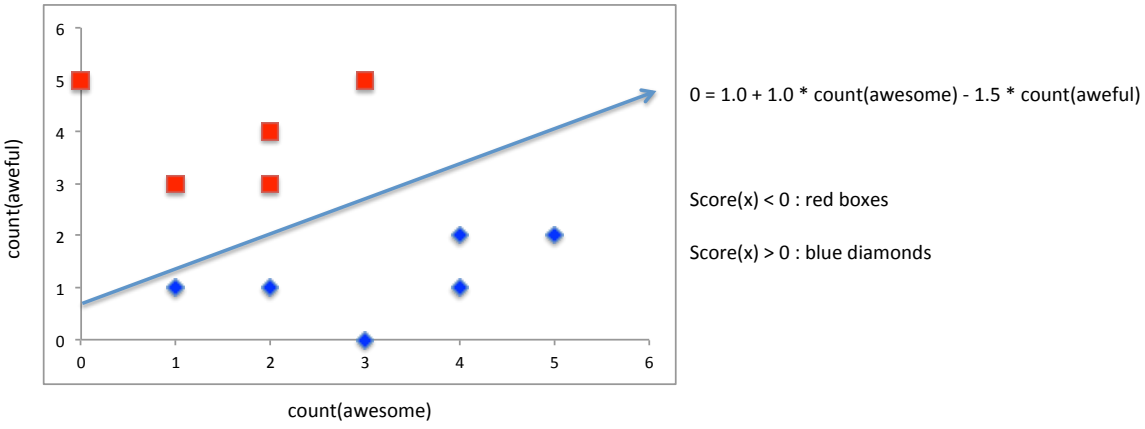
In our prior example, we assumes w_0 was 0. But what if it was one. In that case, it would raise the curve so that it intercepts at count(awful) = 0.75, which we can calculate from;

$$0 = 1.0 + 1.0*\text{count(awesome)} - 1.5*\text{count(awful)}.$$

If we choose count(awesome) to be zero then

$$0 = 1.0 - 1.5 * \text{count(awful)}$$
$$1.5 * \text{count(awful)} = 1.0$$
$$\text{count(awful)} = 0.75 \text{ at count(awesome)} = 0 \text{ when Score(x)} = 0$$

	score(x) = 1.0 + 1.0*count(awesome) - 1.5*count(awful)			
	count(awesome)	**count(awful)**	**score(x)**	**class**
sentence 1	0	5	-6.5	-
sentence 2	1	3	-2.5	-
sentence 3	2	4	-3.0	-
sentence 4	2	3	-1.5	-
sentence 5	3	5	-3.5	-
sentence 6	1	1	0.5	+
sentence 7	4	2	2.0	+
sentence 8	3	0	4.0	+
sentence 9	4	1	3.5	+
sentence 10	5	2	3.0	+
sentence 11	2	1	1.5	+

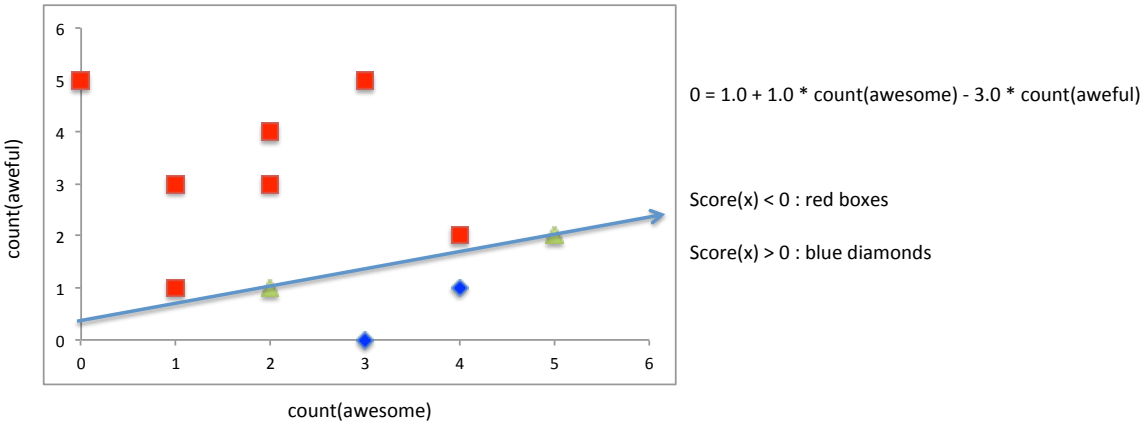


You can see that the score for the point (1,1) is now below the line, so it is a positive sentiment. If we change our coefficients further, such at w_2 = -3.0, then;

Score(x) = 1.0 + 1.0 * count(awesome) – 3.0 * count(awful)

	score(x) = 1.0 + 1.0*count(awesome) - 3.0*count(awful)			
	count(awesome)	**count(awful)**	**score(x)**	**class**
sentence 1	0	5	-14.0	-
sentence 2	1	3	-7.0	-
sentence 3	2	4	-9.0	-
sentence 4	2	3	-6.0	-
sentence 5	3	5	-11.0	-
sentence 6	1	1	-1.0	-
sentence 6	4	2	-1.0	-
sentence 7	3	0	4.0	+
sentence 8	4	1	2.0	+
sentence 9	5	2	0.0	+
sentence 10	2	1	0.0	+

so when Score(x) = 0 and count(awesome) = 0, then count(awful) = 0.33
and when Score(x) = 0 and count(awesome) = 5 then count(awful) = 2,
which creates this boundary line;



With the new coefficients, our model sign(Score(x)) now says (1,1) and (4,2) are negative and (2,2) and (5,2) are on the line (they are zero).

Using features of the inputs
We can restate our model using generic features of the inputs;

$$\hat{y}_i = sign(Score(\vec{x}_i))$$

Where

$$Score(\vec{x}_i) = w_0 h_0(\vec{x}_i) + w_1 h_1(\vec{x}_i) + \cdots + w_D h_D(\vec{x}_i)$$

Our data sets will have many output values and their associated input values. Given a dataset with N output values each associated with d input values;

$\vec{y}$	The vector of N observed output values
$\vec{y_i}$	The i-th observation in the data
$h_j(\vec{x})$	The j-th feature (a scalar) of the input x, This could be just the input value as in a simple hyperplane model, Or more generally this could be could be any function of one or more on the input values of **x**, for instance we could take the logarithm of one input and add it to another input.
$\vec{x_i}$	The input vector for the i-th data point in the data set, like the i-th sentence in a dataset of sentences
N	Capital-N will represent that number of observations in the dataset.
D	Capital-D represents the number of features we extract from those inputs (and since there is a coefficient for each feature, it determines the number of coefficients in the model, which is D+1 because there is the constant feature implied).

This can be restated as a summation;

$$Score(\vec{x}_i) = \sum_{j=0}^{D} w_j h_j(\vec{x}_i)$$

which can be further simplified using vector notation to:

$$Score(\vec{x_i}) = \vec{w}^T h(\vec{x_i})$$

In sentiment analysis, our extracted features might be

feature 1 = $h_0(x)$, eg the constant feature, like 1.
feature 2 = $h_1(x)$, like number of times ‘awesome’ appears
feature 3 = $h_2(x)$, like number of times ‘awful’ appears
…
feature d = $h_{D-1}(x)$, like number of time ‘ramen’ appears

really, $h_j(x)$ can be any function of the input, such as the log(x[2]) or tf-idf(x[2]).

So our model for sentiment analysis, the ML Model referred to in the Machine Learning workflow is;

$$\hat{y}_i = sign(\vec{w}^T h(\vec{x_i}))$$

which produces with +1 or -1.

Predicting class probabilities
How confident is the prediction? We only produce +1 or -1, but we understand that some review are clearly positive or negative and some are harder to interpret. We handle this by calculated a probability along with our prediction.

If we say that the probability that a review is positive is 0.7, then we would expect that on the average for a dataset of reviews, 70% of the reviews would be positives. Or in terms of data, 70% of the rows would be positive.

Interpreting Probabilities as degrees of belief (or degree of assuredness).
If we make the prediction that y = +1, then the probability the y is +1 is given by

$$\mathrm{P}(\mathrm{y} = +1)$$

this can range from 0 to 1. So the inverse, the probability that a review is negative is given by;

$$P(y = -1) \leftarrow 1 - P(y = +1)$$

If $\mathrm{P}(\mathrm{y} = +1)$ is 1, then we are saying that 100% of all reviews are positive, so the inverse is $0 = 1$ - $\mathrm{P}(\mathrm{y} = +1)$ so no review can be negative.

Review of basics of probabilities

Probabilities are always between 0 and 1 inclusive.
So for our two events;

- $0 <= \mathrm{P}(\mathrm{y} = +1) <= 1$
- $0 <= \mathrm{P}(\mathrm{y} = \text{-}1) <= 1$

Probabilities of mutually exclusive, complementary events sum to 1.
Positive sentiment and negative sentiment are complementary; if it is not one then it must be the other and they are mutually exclusive; sentiment cannot be both positive and negative. So the probabilities of these two events must sum to 1.

- $\mathrm{P}(\mathrm{y} = +1) + \mathrm{P}(\mathrm{y} = \text{-}1) = 1$

In the case where the two events are equally likely, then

- $\mathrm{P}(\mathrm{y} = +1) = \mathrm{P}(\mathrm{y} = \text{-}1) = 0.5$

That is for two classes. We can use the same notation and properties when using multiple classes. For instance, if we are classifying pictures of animals as one of dog, cat or bird and nothing else, then we have 3 classes and all the following must hold;

- $0 <= \mathrm{P}(\mathrm{y} = \mathrm{dog}) <= 1$

- $0 <= P(y = \text{cat}) <= 1$
- $0 <= P(y = \text{bird}) <= 1$
- $P(y = \text{dog}) + P(y = \text{cat}) + P(y = \text{bird}) = 1$

Review of basics of conditional probabilities
If we state that "The probability of a review with 3 awesomes and 1 awful is positive is 0.9", then we are saying that on the average, if we look in our dataset for reviews with 3 awesomes and one awful, then we expect that 90% will be classified as positive reviews.

Here the condition is that the review has 3 awesomes and 1 awful AND that it will be positive.

We can use a slightly enhanced notation to capture conditional probabilities;

$P(y = +1 \mid x = \text{"All the sushi was delicious."})$

This is the probability of that the output is positive given an input of "All the sushi was delicious." The bar 'I' means 'given'.

A sure event has a probability of 1.
An impossible event has a probability of 0.

y = +1 and y = -1 are mutually exclusive, complementary events **when given the same input**. However, this is not true if the inputs are different. There is no relationship between the possible right sides.

Given the same inputs;
- $0 <= P(y=+1 \mid x_i) <= 1$
- $0 <= P(y=-1 \mid x_i) <= 1$
- $P(y=+1 \mid x_i) + P(y=-1 \mid x_i) = 1$

For out example of multiple classes of when classifying pictures into dog, cat, bird, given the same input picture;
- $0 <= P(y=\text{dog} \mid x_i) <= 1$
- $0 <= P(y=\text{cat} \mid x_i) <= 1$
- $0 <= P(y=\text{bird} \mid x_i) <= 1$
- $P(y=\text{dog} \mid x_i) + P(y=\text{cat} \mid x_i) + P(y=\text{bird} \mid x_i) = 1$

Using probabilities in classification
$P(y \mid x)$ is the probability of the output label y given the input sentence x. Many classifiers provide a probability of an output label for a given input sentence. In fact, many classifiers make learning the probabilities a goal such that they then use the learned probabilities to predict the output label. The estimated probabilities are derived by finding the best set of estimated coefficients.

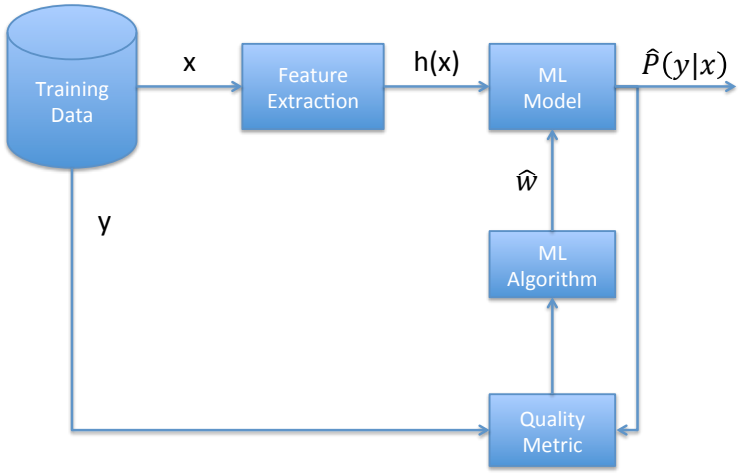
$\hat{P}(y|x)$ is the estimate of class probabilities. If $\hat{P}(y = +1|x) > 0.5$ then we would predict +1, else we would predict -1.

$$\text{if } \hat{P}(y = +1|x) > 0.5 \text{ then } \hat{y} = +1 \text{ else } \hat{y} = -1$$

In addition to providing our predictive output, $\hat{P}(y|x)$ tells us how sure we are of the prediction.

Predicting class probabilities with (generalized) linear models
Now, instead of predicting the class directly, we will estimate the probability of a class given an input. So the ML workflow looks like this;

Machine Learning Linear Classifier Workflow



In our prior work, we calculated a score and turned that into a class based on it's sign;

$$Score(\vec{x}_i) = \vec{w}^T h(\vec{x}_i)$$

$$\hat{y}_i = sign(\vec{w}^T h(\vec{x}_i))$$

In fact, score can theoretically range from $-\infty \; to \; +\infty$. The sign function then maps these values to either -1 or + 1. Now, instead of mapping to just the two classes, we want to map the score to a probability, which can range from 0 to 1. So we want to map $-\infty$ as 0 (surely negative) and $+\infty$ as 1 (surely positive). Importantly, we need to handle those values that are exactly on the decision boundary, where Score(x) = 0. In this case, we saying we don't know if the sentiment is positive or negative; it could be either. This maps to P(y=+1|x) =

0.5; note that in this case then P(y=-1|x) = 1 – P(y=+1|x) = 0.5. So it is equally likely that the sentiment is positive or negative, which is what we want.

Now we just need some way to do the mapping between the score(x) space of $-\infty \; to \; +\infty$ and the P(y|x) space of 0 to 1. We do this with a 'link' function that maps from one space to another space. This creates a **Generalized Linear Model**; a regression model that uses a link function. The link function is annotated simply

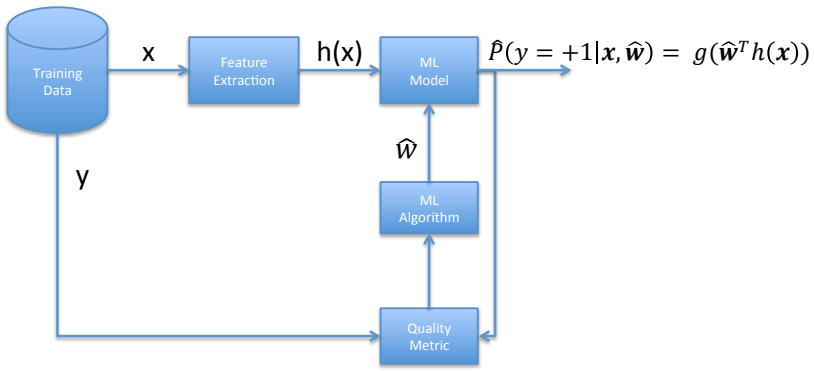
$$\text{Link function:} \quad g(\widehat{\boldsymbol{w}}^T h(\vec{\boldsymbol{x}}_{\boldsymbol{i}}))$$

So now we seek this is our ML workflow;

$$\hat{P}(y = +1|\boldsymbol{x}_{\boldsymbol{i}}, \widehat{\boldsymbol{w}}) = \; g(\widehat{\boldsymbol{w}}^T h(\boldsymbol{x}_{\boldsymbol{i}}))$$

This is calculating the estimated probability of y being +1 given the input $\boldsymbol{x}_{\boldsymbol{i}}$ and the estimated parameters $\widehat{\boldsymbol{w}}$.

Machine Learning Linear Classifier Workflow



The sigmoid (or logistic) link function

Logistic Regression is a specific kind of Generalized Linear Regression that uses the Logistic Function (also called the Sigmoid or Logit function) as the link function.

Logistic (Sigmoid) Function: $y \; = \frac{1}{1+e^{-x}}$

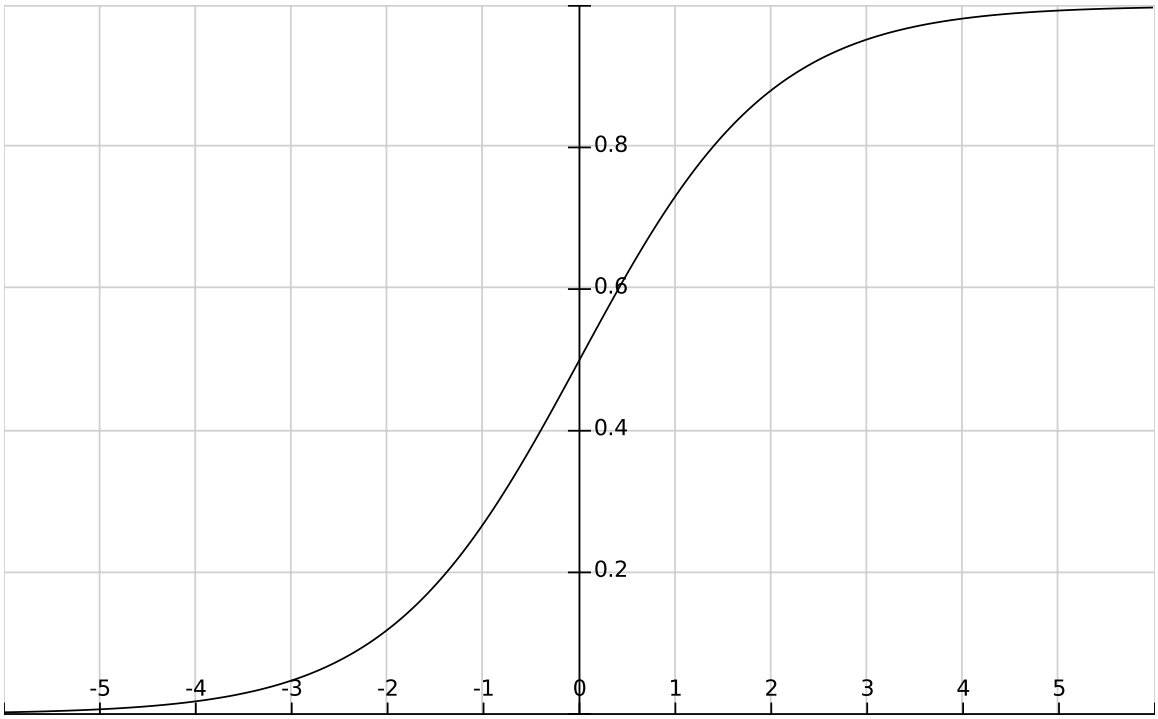
We can see that this maps our Score(x) range of $-\infty \; to \; +\infty$ to our range of probabilities by checking our extremes and the middle;

For Score(x) = $-\infty$, $\frac{1}{1+e^{-(-\infty)}} = \frac{1}{1+e^{\infty}} = \frac{1}{\infty} = 0$

For Score(x) = 0, $\frac{1}{1+e^{0}} = \frac{1}{1+1} = 0.5$

For Score(x) = ∞, $\frac{1}{1+e^{-(+\infty)}} = \frac{1}{1+e^{-\infty}} = \frac{1}{1+0} = 1$

We can see these results in the graph of the sigmoid function below (created using http:/fooplot.com);



Sigmoid Function; $f(x) = \frac{1}{1+e^{-x}}$

We can see that at Score(x) = 0, sigmoid(x) = 0.5. Everything to the left of Score(x) = 0 is a negative score and so it should have a probability less than 0.5 that it is a positive sentiment, which is true. Everything to the right of Score(x) = 0 has a positive score and so it should have a probability > 0.5 that is positive sentiment, which is also true.

Importantly, the Sigmoid function is symmetric. This maintains the property that the probabilities of mutually exclusive complementary events sum to 1.

Logistic regression model

So with the Sigmoid function as our link function, we now have a Logistic Regression model;

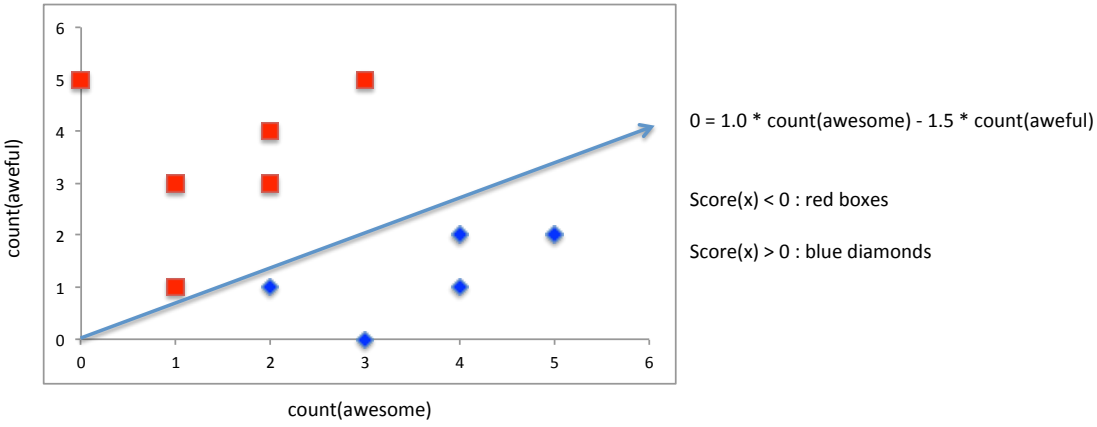
$$\hat{P}(y = +1|\boldsymbol{x_i}, \widehat{\boldsymbol{w}}) = sigmoid(\widehat{\boldsymbol{w}}^T h(\boldsymbol{x_i}))$$

Effect of coefficient values on predicted probabilities

We can further compare the Logisitic Regression model to the decision boundary model. Intuitively, we understand that we can have high confidence in the classification of points that are very far away from the boundary. Conversely, we have lower confidence in the classification of points that are close to the boundary. When a point is exactly on the boundary, we do not know it's class; it could be either positive or negative.

If we go back to our decision boundary data, we can plug in the Score(x) values to calculate the corresponding P(y = +1|x);

	score(x) = 1.0 + 1.0*count(awesome) - 1.5*count(awful)				
	count(awesome)	**count(awful)**	**score(x)**	**P(y = +1|x)**	**class**
sentence 1	0	5	-6.5	0.0015	-
sentence 2	1	3	-2.5	0.0759	-
sentence 3	2	4	-3.0	0.0474	-
sentence 4	2	3	-1.5	0.1824	-
sentence 5	3	5	-3.5	0.0293	-
sentence 6	1	1	0.5	0.6225	+
sentence 7	4	2	2.0	0.8808	+
sentence 8	3	0	4.0	0.9820	+
sentence 9	4	1	3.5	0.9707	+
sentence 10	5	2	3.0	0.9526	+
sentence 11	2	1	1.5	0.8176	+

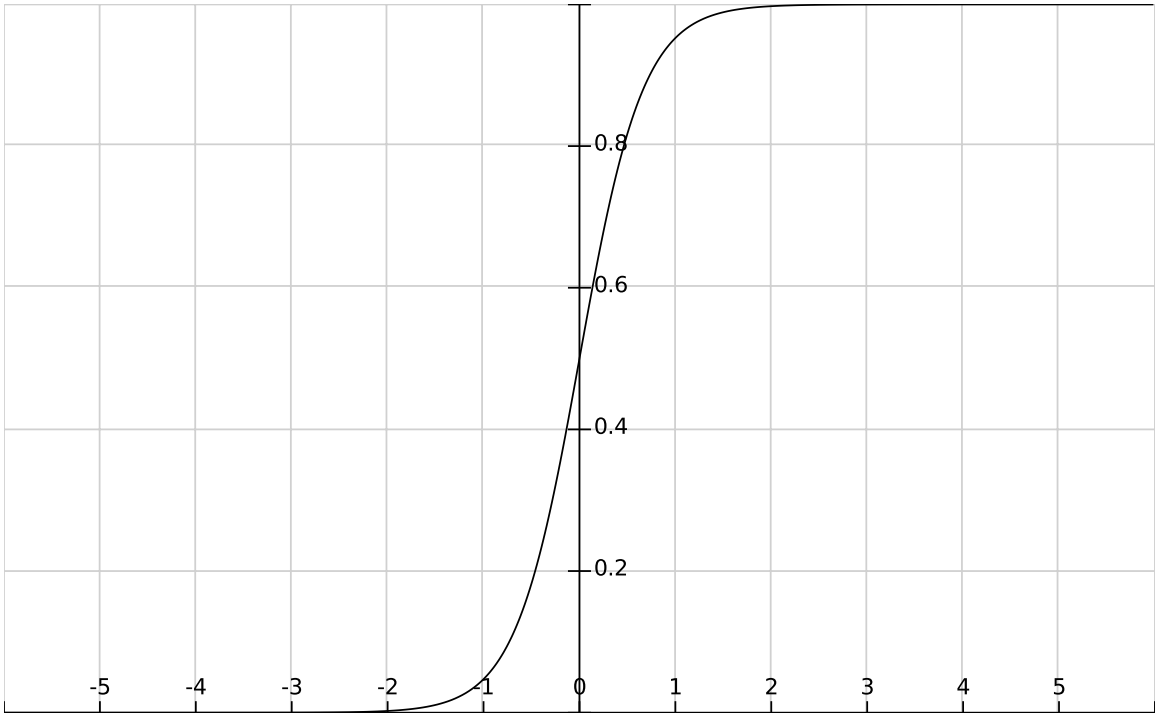


Note the point at (1,1), it is very close to the decision boundary and we can see that it's P(y = +1|x) is close to 0.5 indicating that we do not have high confidence in the prediction.

On the other hand, the point (0,5) is very far from the boundary and above it. It has no awesomes at all. It's P(y = +1) = 0.0015 is nearly zero, so it is very unlikely that it is a positive sentiment. We can calculate the complementary probability P(y = -1) = (1 - 0.0015) = 0.9985; it has a high probability of being negative sentiment.

Effect of w_1 and w_2.
These effectively decide how fast the distance from the decision boundary changes with a change in the feature, so they also decide how big the change in the probability is with a change in that feature. If the magnitude of the coefficients is small, the rate of change in the distance from the boundary and so the rate of change in the probability is small; the sigmoid curve is 'wider' and less steep.

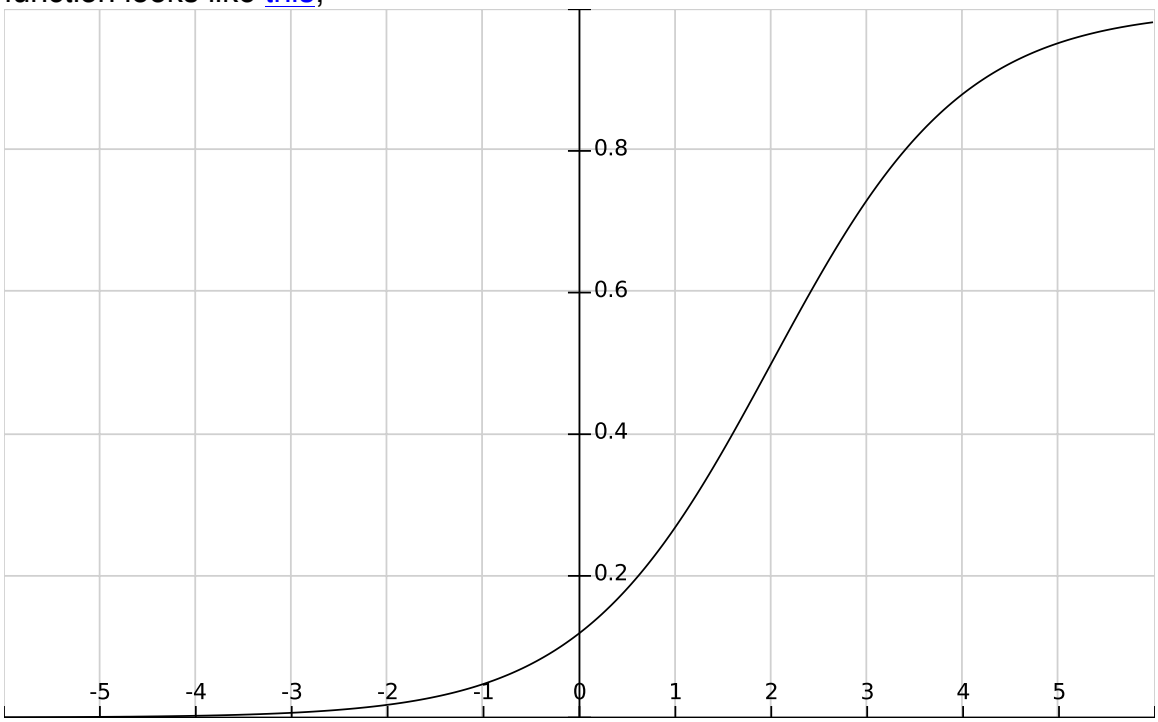
Conversely, if we made the magnitude of the coefficients greater, the rate of change of the distance from the boundary increases. This is seen as a steeper, narrower sigmoid. For instance, if we increase the magnitude of the non-constant coefficients by a factor of 3, then the sigmoid curve looks like this;



Sigmoid Function; $f(x) = \frac{1}{1+e^{-3x}}$

Notice that nothing has changed at the extremes or at x = 0. However, the curve is steeper in the middle and approaches y=0 and y=1 faster (it is narrower). So a small change in Score(x) results in a larger change in P(y = +1|x).

Effect of w_0, the constant feature.
If we choose 2 rather than zero for our constant feature, then our sigmoid function looks like this;



Sigmoid function with $w_o = 2$; $f(x) = \frac{1}{1+e^{(2-x)}}$

Notice that the curve is shifted such that y = 0.5 is now at x = 2. The rate of change in the probability relative to our original curve has not change, but the values are offset.

Overview of learning logistic regression models
Training a classifier means that we are learning the coefficients (weights for each feature), $\widehat{\boldsymbol{w}}$ for the logistic regression model;

$$\hat{P}(y = +1|\boldsymbol{x_i}, \widehat{\boldsymbol{w}}) = sigmoid\left(\widehat{\boldsymbol{w}}^T h(\boldsymbol{x_i})\right) = \frac{1}{1 + e^{-\widehat{\boldsymbol{w}}^T h(\boldsymbol{x_i})}}$$

Finding the best classifier involves testing our estimated coefficients in the model against known output values using a quality metric. In our case, the **quality metric** is the **Likelihood function**, $l(w)$. We will use a Gradient Ascent algorithm to find the set of coefficients with the highest likelihood.

Encoding categorical inputs
Categorical inputs are not continuous numbers, so how do we apply coefficients to them? Sometimes even numbers are categorical variables, like postal code.

Hot-One Encoding
One way to handle this is to use a method called “One-hot Encoding”. Here we give each category it’s own feature which then can have the value either 0 or 1. For instance, country of origin can be encoded in this way. There are 196 countries. Each one would get a column in the H matrix, but in each row of H only one of those columns would have the value 1 and the other 195 would have the value 0, since they are mutually exclusive features. That is why it is called one-hot; it means one-active. This then allows us to apply a weight to the individual country features for for each row, only the country column whose value is 1 will end up contributing to the overall score.

Bag-of-Words Encoding
Here each word is a category. Each word get’s it’s own feature whose value is the word count for that word. In this case, these are NOT mutually exclusive features (as contrasted with one-hot encoding) because the bag for any row can contain any set of words.

In both of these encodings, we end up with sparse data in the feature vector because many of the features will have zeroes in them.

Multiclass classification with 1 versus all
How do we handle classification for more than two classes? For two classes, we essentially find the boundary between the two classes. How do we do this for 3 classes? It is actually very simple. We run 3 separate classifiers. In each case, we pick on class as the positive class and the other classes are lumped into the negative class (the not-positive class). To do the prediction, we run each of the 3 classifiers and pick the class with the highest probability.

We train a classifier for each of C classes. So we can determine the probability that a input is of class C using;

$$\hat{P}_C(y = +1|x)$$

And so the probability that it is not class C is

$$\hat{P}_C(y = -1|x) = 1 - \hat{P}_C(y = +1|x)$$

We do the prediction with each of the C classifiers then pick the one that has the highest probability.