Advanced Institute for Artificial Intelligence

Machine Learning - Classification

Raphael Cóbe

raphaelmcobe@gmail.com

Links and References

- Book: *Artificial Intelligence: A Modern Approach*
- Book: *An Introduction to Statistical Learning*
- Book: *The Elements of Statistical Learning*
- ROC and AUC Tutorial
- Logistic Regression Explained - Python Examples
- Intro to SVM
- KNN explained and applications

Classification

Logistic Regression

Introduction

There are problems for which we want to estimate the **class** (label) of a given sample based on its input data set, that is, the **features** of your problem.

Classification problems are similar to regression problems because we want to learn a function that, given an input, estimates an output value. The difference is that our output will be mapped to a **probability** (*soft classification*) or **label** (*hard classification*) of the sample.

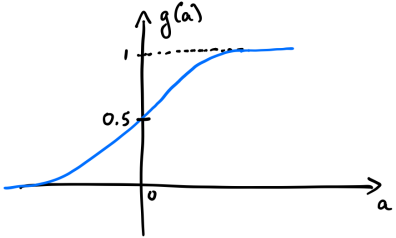
The Logistic Regression technique is one of the most used in the literature, mainly because it is simple and gives good results in various situations. It has this name because it estimates the probability of a given sample belonging to a specific class. Therefore, it is a *soft* classification technique.

Problem definition: let a dataset $\mathcal{X} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_z, y_z)\}$ such that $\boldsymbol{x}_i \in \mathbb{R}^{n+1}$ corresponds to the input data and $y_i \in [0, 1]$ denotes its respective output value. We also have that $\mathcal{X}$ can be **partitioned** as follows: $\mathcal{X} = \mathcal{X}^1 \cup \mathcal{X}^2$, where $\mathcal{X}^1$ and $\mathcal{X}^2$ denote the **training** and **test** datasets, respectively. Our goal is, given the training set, to learn a function $h : \mathbb{R}^{n+1} \to [0, 1]$ that can estimate the probability of a sample belonging to a given class. Why use a logistic (sigmoid) function?

The Logistic Function has some interesting properties with the following formulation:

$$g(a) = \frac{1}{1 + e^{-a}}, \tag{1}$$

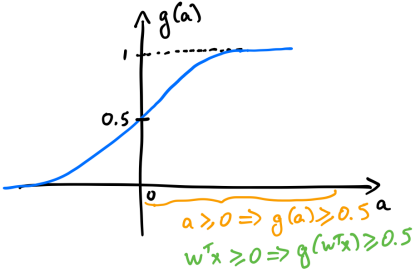
such that $g(a) \in [0, 1]$.



How does the Logistic Regressor work? Given that we want to obtain an output $h_{\boldsymbol{w}}(\boldsymbol{x}) \in [0, 1]$, we just need to modify our input in Equation 1 as follows:

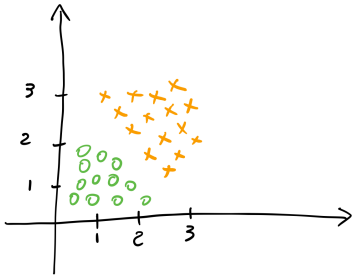
$$\begin{aligned} h_{\boldsymbol{w}}(\boldsymbol{x}) &= g(\boldsymbol{w}^T \boldsymbol{x}) \\ &= \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}. \end{aligned} \tag{2}$$

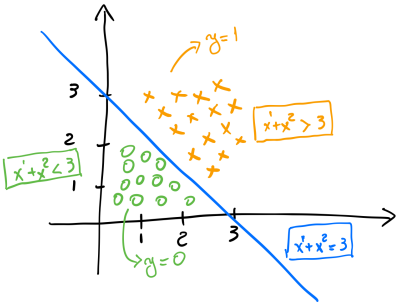
Now, the term $\boldsymbol{w}^T \boldsymbol{x}$ is called the **basis function**.



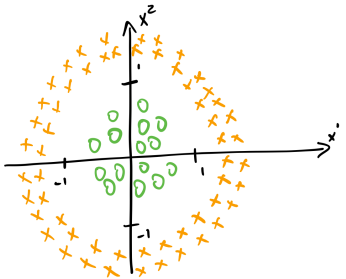
In practice, we want $h_{\boldsymbol{w}}(\boldsymbol{x}) \geq 0.5$ when $\boldsymbol{w}^T \boldsymbol{x} \geq 0$. Similarly, we have that $h_{\boldsymbol{w}}(\boldsymbol{x}) < 0.5$ when $\boldsymbol{w}^T \boldsymbol{x} < 0$.

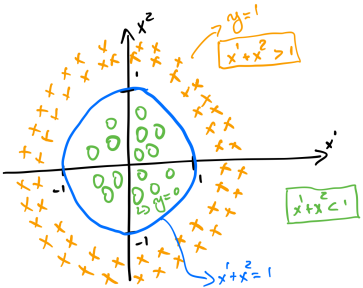
Suppose we have the following situation, where $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{w}^T\boldsymbol{x})$ and $\boldsymbol{w} = [-3\ 1\ 1]$. We want to assign $y = 1$ if $\boldsymbol{w}^T\boldsymbol{x} > 0$, that is, if $-3 + x^1 + x^2 > 0 \Rightarrow x^1 + x^2 > 3$. Similarly, we want to assign $y = 0$ if $\boldsymbol{w}^T\boldsymbol{x} < 0$, that is, if $-3 + x^1 + x^2 < 0 \Rightarrow x^1 + x^2 < 3$.





Now suppose another situation, where $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(w_0 + w_1 x^1 + w_2 x^2 + w_3 (x^1)^2 + w_4 (x^2)^2)$ and $\boldsymbol{w} = [-1\ 0\ 0\ 1\ 1]$. We want to assign $y = 1$ if $-1 + (x^1)^2 + (x^2)^2 > 0 \Rightarrow (x^1)^2 + (x^2)^2 > 1$. Similarly, we want to assign $y = 0$ if $1 + (x^1)^2 + (x^2)^2 < 0 \Rightarrow (x^1)^2 + (x^2)^2 < 1$.





In this way, we can note that, depending on the chosen basis function, different separation surfaces can be obtained. Usually, the higher the degree of the polynomial, the more complex its surface will be.

So, we have the following information so far:

- Hypothesis function: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \dfrac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}$.
- Basis function: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$ (default).
- Cost function: ?

Why is it not interesting to use MSE as a cost function for the Logistic Regressor?

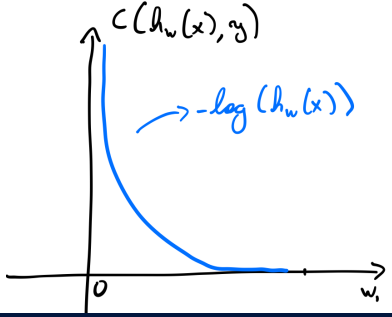
We have two main problems with MSE when using the Logistic Regression technique:

1. Suppose the true label of any sample $\boldsymbol{x} \in \mathcal{X}$ is $y = 1$, and our classifier outputs $h_{\boldsymbol{w}}(\boldsymbol{x}) = 0$. In this case (for $m = 1$), we have $J(\boldsymbol{w}) = \frac{1}{2}(1-0)^2 = 0.5$. Note that this is a very small penalty for a classification **error**. Therefore, MSE does not penalize classification errors very strongly and may lead to insufficient learning.
2. The MSE cost function is **not convex** for the Logistic Regressor (mathematically proven).

Let's then define a new cost function that is convex. Our function will be divided, initially, into two parts:

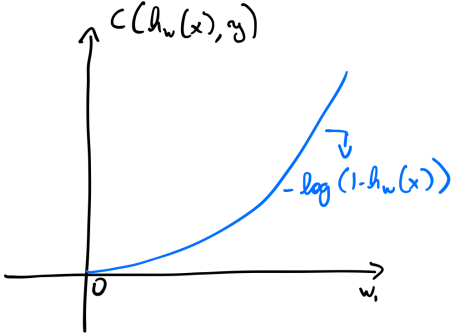
$$C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = \begin{cases} -\log(h_{\boldsymbol{w}}(\boldsymbol{x})) \text{ if } y = 1 \\ -\log(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \text{ if } y = 0. \end{cases} \tag{3}$$

Let's now analyze the two situations separately. As the first case, we have the situation where $y = 1$.



In this case, when we have an error, i.e., $h_{\boldsymbol{w}}(\boldsymbol{x}) = 0$, then $C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = -\log(0) = \infty$. In the case of a correct classification, i.e., $h_{\boldsymbol{w}}(\boldsymbol{x}) = 1$, we have $C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = -\log(1) = 0$

The next situation occurs when $y = 0$.



In this case, when we have an error, i.e., $h_{\boldsymbol{w}}(\boldsymbol{x}) = 1$, then $C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = -\log(1-1) = -\log(0) = \infty$. In the case of a correct classification, i.e., $h_{\boldsymbol{w}}(\boldsymbol{x}) = 0$, we have $C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = -\log(1-0) = -\log(1) = 0$.

We can write Equation 3 by grouping the two situations described earlier:

$$C(h_{\boldsymbol{w}}(\boldsymbol{x}), y) = -y \log(h_{\boldsymbol{w}}(\boldsymbol{x})) - (1 - y) log(1 - h_{\boldsymbol{w}}(\boldsymbol{x})). \tag{4}$$

When $y = 1$, the red term is zeroed and the above equation becomes Equation 3 for this case. On the other hand, when $y = 0$, the blue term is zeroed and the above equation becomes Equation 3 for this case.

Our final cost function for the Logistic Regressor is given by:

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{m}\sum_{i=1}^{m} C(h_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) \\
&= \frac{1}{m}\sum_{i=1}^{m} [-y_i \log(h_{\boldsymbol{w}}(\boldsymbol{x_i})) - (1-y_i)\log(1-h_{\boldsymbol{w}}(\boldsymbol{x}_i))]. \qquad (5)
\end{aligned}
$$

The function above is also known as **binary cross-entropy**.

Now, we just need to learn the set of parameters $\boldsymbol{w}$ similarly to Linear Regression, that is, we can use the gradient descent technique. The derivative of the cost function presented in Equation 5 is given as follows:

$$\frac{\partial J(\boldsymbol{w})}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_{\boldsymbol{w}}(\boldsymbol{x}_i) - y_i)x_i^j]. \tag{6}$$

Although the formulation above is identical to Linear Regression, we must remember that the hypothesis function $(h_{\boldsymbol{w}}(\boldsymbol{x})$ is different.

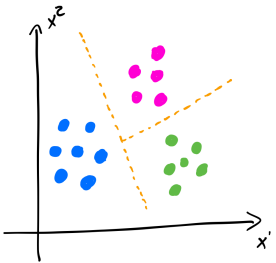
Thus, the gradient descent algorithm can be summarized as follows:

1. Assign random values to $\boldsymbol{w}$.
2. Evaluate the cost function $J(\boldsymbol{w})$.
3. If the **stopping criterion has been reached**, go to step 6.
4. $w_j^{(t+1)} = w_j^{(t)} - \alpha \frac{1}{m} \sum_{i=1}^{m} [(h_{\boldsymbol{w}}(x_i) - y_i)^2 x_i^j]$.
5. Return to Step 2.
6. End of algorithm.

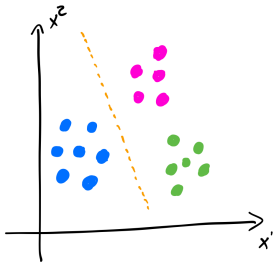
Note that the weights in $\boldsymbol{w}$ need to be updated simultaneously!

However, note that the Logistic Regressor is naturally a **binary classifier**, that is, $y_i \in \{0, 1\}$, $\forall i = 1, 2, \ldots, z$. So, how can we work on problems that have **multiple classes**?

Now, our labels can be represented as follows: $y_i \in \{0, 1, \ldots, c-1\}$, where c corresponds to the number of classes. Suppose the following situation where $c = 3$.

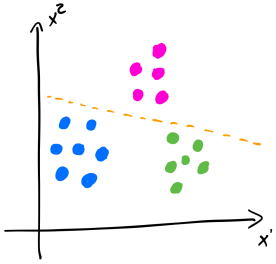


There are some approaches to handle this problem, such as OVO (*one-versus-one*) and OVA (*one-versus-all*). Let's consider the OVA approach. In this case, for a classification problem with c classes, we have to create c classifiers. Let's see the example below where $c = 3$.



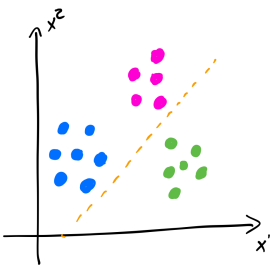
Blue versus all

$h^1_{\boldsymbol{w}}(\boldsymbol{x})$



Pink versus all

$h^2_{\boldsymbol{w}}(\boldsymbol{x})$



Green versus all

$h^3_{\boldsymbol{w}}(\boldsymbol{x})$

We just need to train each classifier $h^i_{\boldsymbol{w}}(\boldsymbol{x})$, $\forall i = 0, 1, \ldots, c-1$ on $\mathcal{X}^1$. Given a sample $\boldsymbol{x} \in \mathcal{X}^2$, its label y will be the one that satisfies the following decision rule:

$$y = \arg\max_i \{h^i_{\boldsymbol{w}}(\boldsymbol{x})\}. \tag{7}$$

Classification

k-Nearest Neighbors

Introduction

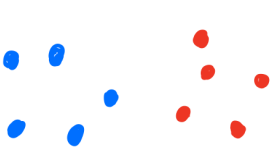
One of the most traditional techniques in machine learning is known as k-Nearest Neighbors (k-NN). This technique is a generalization of an older one known as Nearest Neighbors (NN). Both are quite simple approaches, as **there is no training stage**, even though we still have the training set.

Problem definition: given a labeled training set with m samples $\mathcal{X}^1 = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y$
we want to correctly classify a sample $\boldsymbol{x} \in \mathcal{X}^2$.

Objective: given any sample $\boldsymbol{x}$ from the test set, its label y will be the same as the closest sample from the training set, i.e., the one that satisfies the following equation:

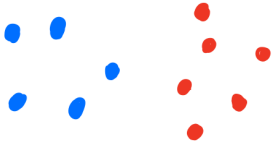
$$y = \underset{y_i | \boldsymbol{x}_i \in \mathcal{X}^1}{\arg\min} \|\boldsymbol{x} - \boldsymbol{x}_i\|_2. \tag{8}$$

Let's see an example of how the NN technique works.



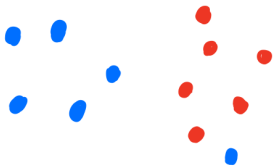
Training set

Sample to be classified

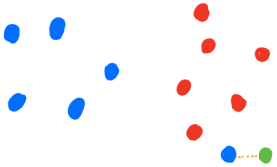


Classified sample

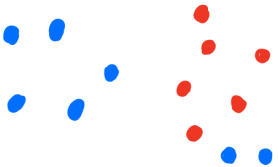
When the dataset is "well-behaved", NN is one of the best techniques to use. However, this doesn't always happen. Problem? "Noise" in the training dataset.



Training set

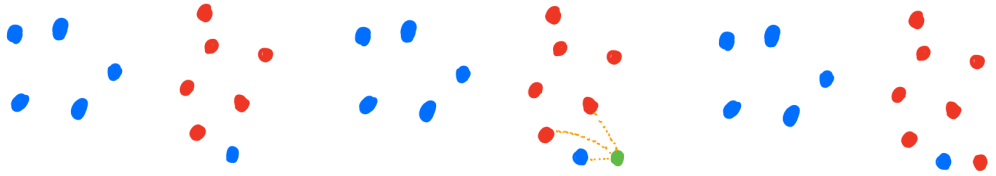


Sample to be classified



Classified sample

A generalization of the NN technique would then be to connect the test sample to its k nearest neighbors, giving rise to the k-NN classifier. Thus, considering the previous example, the sample would be correctly classified if we considered $k = 3$, for example (we usually use odd values for k to avoid ties).



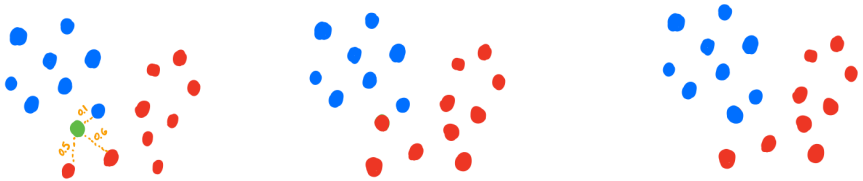
Training set Sample to be classified Classified sample

The k-NN technique is interesting for **recommendation** and **retrieval** problems, since it uses the closest samples for decision making. This data can then be used for recommendation purposes. Another important point concerns **regression** by k-NN, which is also quite simple. In this case, when connecting the test sample to its k nearest neighbors, just use, for example, the average value of their outputs as the value to be estimated.

Variants

A known variant of the k-NN technique is its **weighted** version, known as *weighted k-NN*. The idea consists of associating weights to each of the k nearest neighbors, which can be, for example, the **inverse of their distance** to the sample in question. These weights are normalized and used to weight the decision. The idea is that more distant samples have less influence during the decision process.

Let's see an example of weighted k-nn versus its traditional version.



Training set Classification by k-NN Classification by weighted k-NN

Let $\boldsymbol{w} \in \mathbb{R}^3$ be the weight vector, such that $w_1 = 1/0.1$ (blue class), $w_2 = 1/0.5$ (red class) and $w_3 = 1/0.6$ (red class). Normalizing them, we have $w_1 = 0.74$, $w_2 = 0.17$ and $w_3 = 0.09$. Even though the green sample is connected to the two samples of the red class, the weight of this class ($w_2 + w_3 = 0.26$) is less than that given by the sample of the blue class, i.e., $w_1 = 0.74$.

Classification

Evaluation Metrics and Data Division

Evaluation Metrics

Metrics are quantifiable measures used to analyze results of a specific process, action, or strategy.

When we bring this definition to the ML world, metrics are essential to inform the performance of a given model in a given application. Thus, the choice of the evaluation metric necessarily needs to be representative and consistent with the problem being addressed.

For classification problems, the literature provides us with some well-established and concise metrics that help in monitoring the performance of ML models, both in training and testing stages.

Let's take as a basis for the development of the topic, a binary classification problem whose objective is to separate nails from screws on an industrial conveyor belt.

Among the various metrics in the literature, some are more commonly used, such as:

- *Accuracy*;
- *Precision*;
- *Recall*;
- *F-Measure*;

The predictions of the ML model generate the so-called confusion matrix (or contingency matrix):

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

**FP $->$ overestimation (false alarm) - type I error;

**FN $->$ underestimation (serious error) - type II error;

Accuracy - Represents the measure of correctness of the ML model, i.e., how many examples were predicted correctly compared to the evaluated data population:

$$ACC = \frac{TP + TN}{P + N}$$
$$= \frac{TP + TN}{TP + TN + FP + FN}$$

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Precision - Represents the fraction of correctly labeled instances (TP) among all that were classified as correct, i.e., true positives and false positives:

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

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Recall - Represents the fraction of correctly labeled instances (TP) among all positives in the data population:

$$RC = \frac{TP}{P} = \frac{TP}{TP + FN}$$

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Example: Consider an ML model to recognize dogs (relevant element) in an image. When processing an image that contains ten cats and twelve dogs, the model identifies eight dogs.

Of the eight elements identified as dogs, only five are actually dogs (TP), while the other three are cats (FP).

Seven dogs were missed (FN) and seven cats were correctly excluded (TN).

The model's *precision* is then 5/8 (TP/selected elements) while its *recall* is 5/12 (TP/relevant elements).

In summary: *Precision* can be seen as a measure of **quality** and *recall* as a measure of **quantity**.

Higher *precision* means that an algorithm returns more relevant results than irrelevant ones, and high *recall* means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

F-Measure - Combines *precision* and *recall* into a harmonic mean between them, represented by:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$
$$= 2 * \frac{TP}{TP + FP + FN}$$

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Quick Recap

- Classification problem:
  - Decide to predict class values
- Predict probabilities for each class instead
  - Ability to choose and calibrate the threshold of how to interpret predicted probabilities
  - For example, the 0.5 threshold:
    - Probability in [0.0, 0.49] is a negative result (0), and
    - Probability in [0.5, 1.0] is a positive result (1)
- The threshold can be adjusted to fine-tune the model's behavior for a specific problem:
  - One example would be reducing one type of error more than the other

Quick Recap

- Adjusting the threshold:
  - In a diagnostic prediction system, we might be much more concerned with having low false negatives than low false positives.
    - a false negative would mean not alerting about a particular disease, leading the patient not to worry about a certain condition
  - A false positive would mean starting a treatment that is not needed
- A common way to compare models that predict probabilities for two-class problems is to use an ROC curve.

ROC Curves

- An ROC curve plots TPR vs. FPR at different classification thresholds.
- Lowering the classification threshold classifies more items as positive
  - **Increases False Positives and True Positives**
- On the other hand, raising the classification threshold classifies more items as negative
  - Increases False Negatives and True Negatives
- Example: If it's a cancer classification application that you don't want your threshold to be as high as 0.5. Even if a patient has a 0.3 probability of having cancer you would classify them as 1

ROC Curves

- A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve
- Plot of the false positive rate (x-axis) versus the true positive rate (y-axis)
  - Try a number of different candidate threshold values between 0.0 and 1.0
- The true positive rate describes how good the model is at predicting the positive class when the actual outcome is positive
  - The true positive rate is also known as **sensitivity**
- The false positive rate is also called the **false alarm rate** as it summarizes how often a positive class is predicted when the actual outcome is negative.

Quick Recap

True Positive Rate (TPR) is synonymous with **recall** and is defined as follows:

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

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

ROC Curves

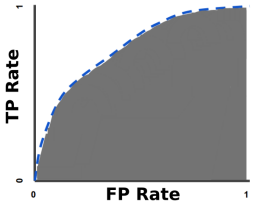
- Very useful tool
- Area Under the Curve (AUC) can be used as a summary of the model's capability
- Smaller values on the x-axis of the graph indicate fewer false positives and higher true negatives
- Larger values on the y-axis of the graph indicate more true positives and lower false negatives

ROC Curves

- Good models have curves that bow upward, toward the top left corner of the graph
- A skilled model will assign a higher probability to a randomly chosen real positive occurrence than to a negative one, on average

AUC: Area under the ROC Curve

- AUC stands for "Area under the ROC Curve". That is, AUC measures the entire two-dimensional area under the entire ROC curve (think of calculus integral) from (0,0) to (1,1).
- provides an aggregate measure of performance across all possible classification thresholds
- probability that the model ranks a random positive example higher than a random negative example



AUC: Area under the ROC Curve

- AUC ranges from 0 to 1:
  - 100% wrong has an AUC of 0.0; 100% correct has an AUC of 1.0.
- scale invariant
- in the cost of false negatives vs. false positives, it can be critical to minimize one type of classification error

ROC Curve - Python Code

```python
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y, scores, pos_label=2)
roc_auc = auc(fpr, tpr)
print("AUC: {}".format(roc_auc))
plt.plot(fpr, tpr)
```