Machine Learning Notes

Amanjot Bhullar

December 22, 2017

This text is significantly influenced by the Coursera course "Machine Learning" by Andrew NG which is available on `https://www.coursera.org/learn/machine-learning`.

Blue text found in this report is quoted or paraphrased from another text.

1 Supervised Learning

The machine learns with the aid of correct answers.

1.1 Classification – Logistic Regression

A **classification** problem has discrete outcomes "0 or 1 or 2 ...". For example, does a given image best resemble a cat or dog (0 or 1).

1.1.1 The Data Set

The data set in a classification problem typically looks like the following:

x_0	x_1	x_2	x_3	x_4	...	x_n	y
1	2104	5	1	45	...	6	0
1	1416	3	2	40	...	4	1
1	1534	3	2	30	...	7	1
1	8152	2	1	36	...	5	0
1	...	...	...	...	...	...	...
1	5689	9	26	33	...	22	0

In matrix form: $\mathbf{X} = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 & ... & 6 \\ 1 & 1416 & 3 & 2 & 40 & ... & 4 \\ 1 & 1534 & 3 & 2 & 30 & ... & 7 \\ 1 & 8152 & 2 & 1 & 36 & ... & 5 \\ 1 & ... & ... & ... & ... & ... & ... \\ 1 & 5689 & 9 & 26 & 33 & ... & 22 \end{bmatrix}$, an `m`$\times$`(n+1)` matrix – the design matrix, and $\boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ ... \\ 0 \end{bmatrix}$, an `m`$\times$`1` matrix.

m is the number of training examples. The $x_1, ..., x_n$ parameters are the chosen features, the $x_0 = 1$ is simply for the bias term, and y is the class/label of the training example.

Note: Try to get roughly the same amount of training examples for each class. For example, No. of training examples for malignant tumors is same as No. of training examples of benign tumors.

1.1.2 The Goal

The goal of a classification problem is to find the best **decision boundary**.

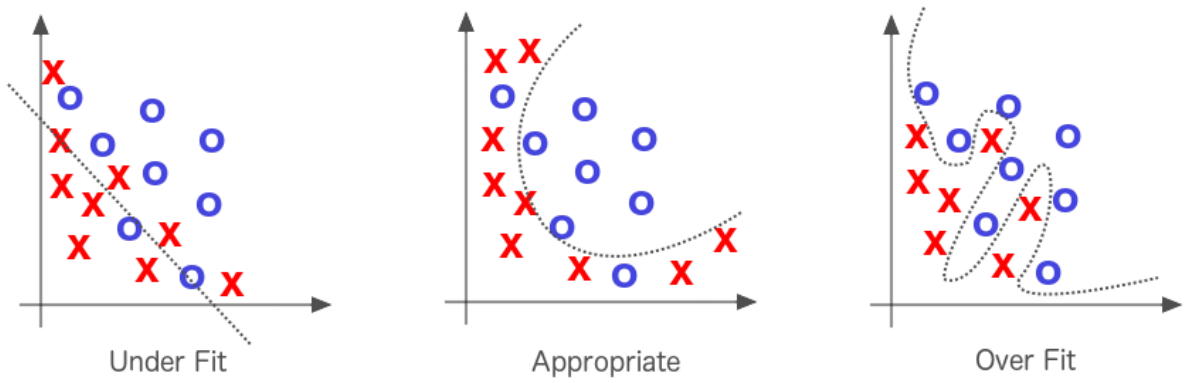
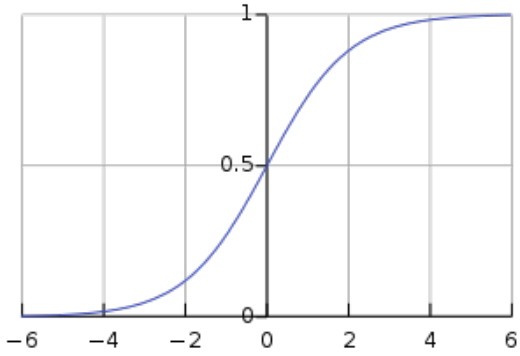


Illustration credit: [1]

The decision boundary is the hyperspace that separates the area of mostly 0's and the area of mostly 1's. It is created by our hypothesis function.

1.1.3 Hypothesis Function h_θ

The hypothesis function must tell us if the given features best correspond to a label of 0 or 1. A function that can do this is the sigmoid function.



The equation for the sigmoid function is

$$h_\theta(z) = \frac{1}{1 + e^{-z}}$$

The z value in the above equation incorporates the essence of the features for a particular item whose label you would like to predict. The z value also determines the complexity of

the decision boundary (linear, polynomial, Gaussian, ...).

For instance, if we think that a line would be a good decision boundary than z will look like the following (assuming we only have 2 features):

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

More generally, if we have n features and we believe that the labels are linearly separable than the decision boundary would be a hyper-plane of the form:

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

If we want the decision boundary to be a cubic polynomial hyperspace instead of a hyper-plane just add higher order terms:

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + ... + \theta_{n-2} x_n + \theta_{n-1} x_n^2 + \theta_n x_n^3$$

In general,

$$z = \boldsymbol{\theta}^T \mathbf{x}$$

where $\mathbf{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \\ \cdots\cdots \\ x_n \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ .. \\ \theta_n \end{bmatrix}$

Note: $\boldsymbol{\theta}$ are the weights, and θ_0 is known as the bias term.

Hence,

$$h_\theta(z) = h_\theta(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}} \tag{1}$$

In summary, $h_\theta(\boldsymbol{z})$ is the estimated probability that $y = 1$ on input z.
Or $h_\theta(\boldsymbol{z}) = P(y = 1 | \boldsymbol{x}; \boldsymbol{\theta})$. This means probability that $y = 1$ given $\boldsymbol{x}$ and parametrized by $\boldsymbol{\theta}$.

1.1.4 Example of Decision Boundary and h_θ

For visualization purposes, only 2 features, x_1 and x_2, are used for the example below. In reality, classification problems typically involve > 2 features.

Additionally, the green $\times$'s represent positive training examples ($y = 1$), and the blue $\circ$'s represent negative training examples ($y = 0$).

Example 1

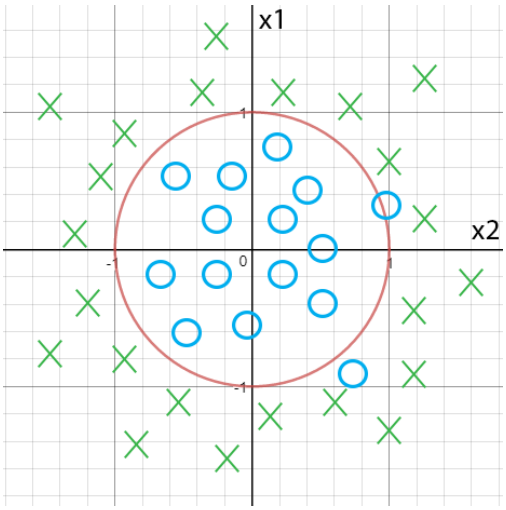


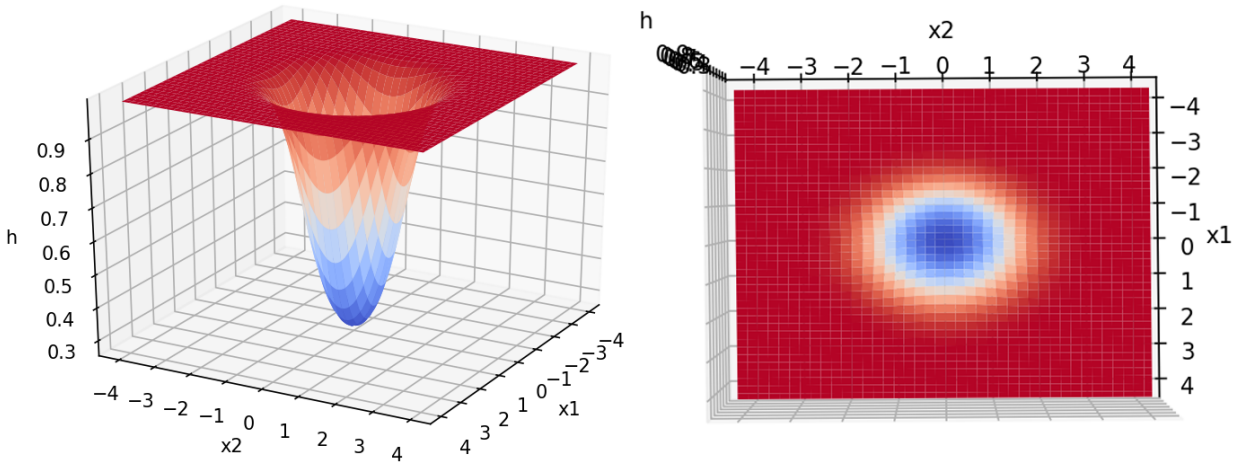
Figure 1

In the above example, it seems that a circle would be a good decision boundary. In this case, the equation for the decision boundary is

$$x_1^2 = 1 - x_2^2$$

And equation (1) for this case would be:

$$h_\theta(z) = \frac{1}{1 + e^{-(-1+x_1^2+x_2^2)}} \tag{2}$$



The above 3D plot is of equation (2). h_θ can be thought of as a probability distribution as it displays the probability that $y = 1$ given certain x_1 and x_2 combinations. The blue region in the 3D plot displays the x_1 and x_2 combinations that have a < 0.5 probability of being $y = 1$, and thus, they would be classified as $y = 0$.

It is also important to note that Figure 1 is a cross section of the 3D plot at $h_\theta = 0.5$.
Proof:

$$\begin{aligned} \frac{1}{1 + e^{-(-1+x_1^2+x_2^2)}} &= 0.5 \\ \Rightarrow 1 &= 0.5\big(1 + e^{-(-1+x_1^2+x_2^2)}\big) \\ \Rightarrow \ln(1) &= -(-1 + x_1^2 + x_2^2) \\ \Rightarrow 0 &= -1 + x_1^2 + x_2^2 \\ \Rightarrow x_1^2 &= 1 - x_2^2 \end{aligned}$$

In summary, the equation of the decision boundary can be found by solving $h_\theta = 0.5$. Also, for this example, $\mathbf{x} = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$. Hence, $z = \boldsymbol{\theta}^T\mathbf{x} = -1 + x_1^2 + x_2^2$.

1.1.5 The Cost Function and Gradient Descent

Suppose you have a 2D data-set where the positive and negative examples can be separated with an ellipse. The general equation for an ellipse is:

$$\phi_1(x_1 - \phi_2)^2 = -\phi_3(x_2 - \phi_4)^2 + \phi_5$$

Expanding the above, the general form of an ellipse is:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 = 0$$

Thus, for this problem equation (1) would be of the form:

$$h_\theta(z) = \frac{1}{1 + e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\theta_3 x_1^2+\theta_4 x_2^2)}} \tag{3}$$

How do we fit the parameters $\boldsymbol{\theta}$?

Gradient descent along with the **cost function** can be used to find the weights, $\boldsymbol{\theta}$, that minimize error in our classification. The training data is used to calculate $\boldsymbol{\theta}$.

The cost function for classification is:

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m} y^{(i)}\texttt{log}(h_\theta(x^{(i)})) + (1 - y^{(i)})\texttt{log}(1 - h_\theta(x^{(i)})) \tag{4}$$

where $y^{(i)}$ and $x^{(i)}$ are the label and features for a training example, respectively. This cost function can be derived from statistics using the principle of maximum likelihood estimation.

The cost function gives the error in the classification. We want to find the $\boldsymbol{\theta}$ that minimizes $J(\theta)$. This can be done using gradient descent.

Gradient Descent

1. Initialize $\boldsymbol{\theta}$. For example, $\theta_0 = 0, \theta_1 = 0, ...$
2. Update $\boldsymbol{\theta}$ in a way that reduces the error, $J(\boldsymbol{\theta})$. Specifically, update $\boldsymbol{\theta}$ by

$$\theta_j = \theta_j - \alpha \frac{d}{d\theta_j} J(\boldsymbol{\theta}) \tag{5}$$

$$= \theta_j - \alpha \sum_{i=1}^{m} \big(h_\theta(x^{(i)}) - y^{(i)}\big) x_j^{(i)}$$

3. Repeat until convergence.

Intuitively, equation (5) is updating θ_j by moving in the opposite direction of increasing error. Additionally, α is called the learning rate, and it determines how big of a step we take when updating θ_j. It is important to note that if α is too small than gradient descent will take a long time to converge, and if α is too big than gradient descent may overshoot the minimum.

Tips

1. Plot $J(\boldsymbol{\theta})$ vs. Number of Iterations to see when gradient descent has converged.
2. Declare convergence if subsequent iterations only decrease $J(\boldsymbol{\theta})$ by $\epsilon = 10^{-3}$.
3. If gradient descent is increasing $J(\boldsymbol{\theta})$ than lower α. If gradient descent is decreasing $J(\boldsymbol{\theta})$ to slowly than increase α.

1.1.6 Feature Scaling (For Fast Convergence)

Note: Do not scale $x_0 = 1$

Gradient Descent converges more quickly when the features are of the same magnitude. **Mean normalization** is computed to normalize each set of features and give them a mean of 0:

$$\frac{x_j^{(i)} - \mu_j}{\texttt{max value}}$$

or

$$\frac{x_j^{(i)} - \mu_j}{s_j}$$

where $x_j^{(i)}$ is the x_j value for the i^{th} training example, μ_j is the mean for column x_j, and s_j is the standard deviation for column x_j. Now $-0.5 \leq x_j \leq 0.5$.

1.1.7 Regularization - Make simpler hypothesis

Note: Don't regularize θ_0.
Suppose the decision boundary is a 3^{rd} order polynomial, $\theta_0+\theta_1x_1+\theta_2x_2+...+\theta_{n-1}x_1^3+\theta_nx_2^3$. However, it is slightly over-fitting to the data. Ergo, you decide to use a quadratic decision boundary, but you come to realize that it is slightly under-fitting to the data.

A solution to this problem is to penalize the higher order weights so that you get a more quadratic decision boundary. In other words, the cost function will look something like the following:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + 1000\theta_{n-1}^2 + 1000\theta_n^2$$

The extra terms, $1000\theta_{n-1}^2 + 1000\theta_n^2$, will result in the ideal model having smaller weights, θ_{n-1} and θ_n, as this will reduce the error in classification. Hence, the decision boundary will be more quadratic.

But what if you don't know which terms/weights to penalize?
In general, use the below cost function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

This will result in a simpler model that is more likely to generalize.

Note: If λ is too big then it results in under fitting because $\theta_1, ..., \theta_n \approx 0$ hence the model will just be equal to θ_0.

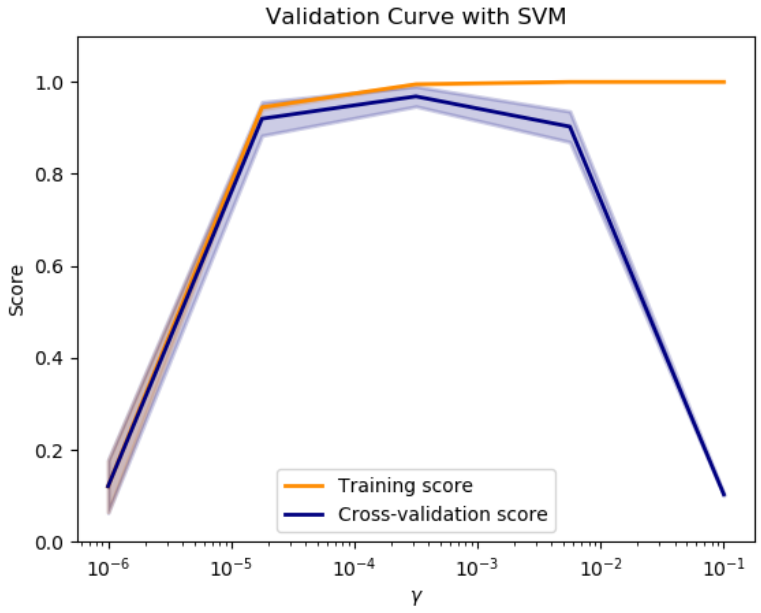
1.1.8 Validation Curve – Choosing the Best Hyper-Parameters

Split your data into

1. Training set (60%) – used to calculate h_θ.
2. Cross-validation set (20%) – used to determine best h_θ.
3. Test set (20%) – used to determine how good h_θ really is.

How do you determine the value of λ that will give the the best model?

Plot a validation curve.

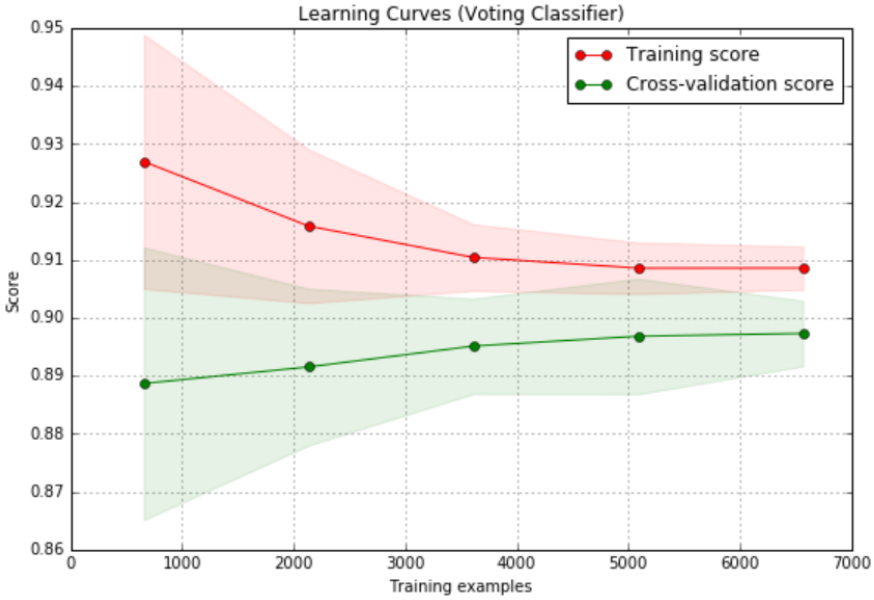


The score is $1 - \texttt{error}$. In this plot you can see the training scores and validation scores of an SVM for different values of the kernel hyper-parameter γ. For very low values of γ, you can see that both the training score and the validation score are low. This is called under-fitting. Medium values of gamma will result in high values for both scores, i.e. the classifier is performing fairly well. If gamma is too high, the classifier will overfit, which means that the training score is good but the validation score is poor. Illustration and caption credit: [4].

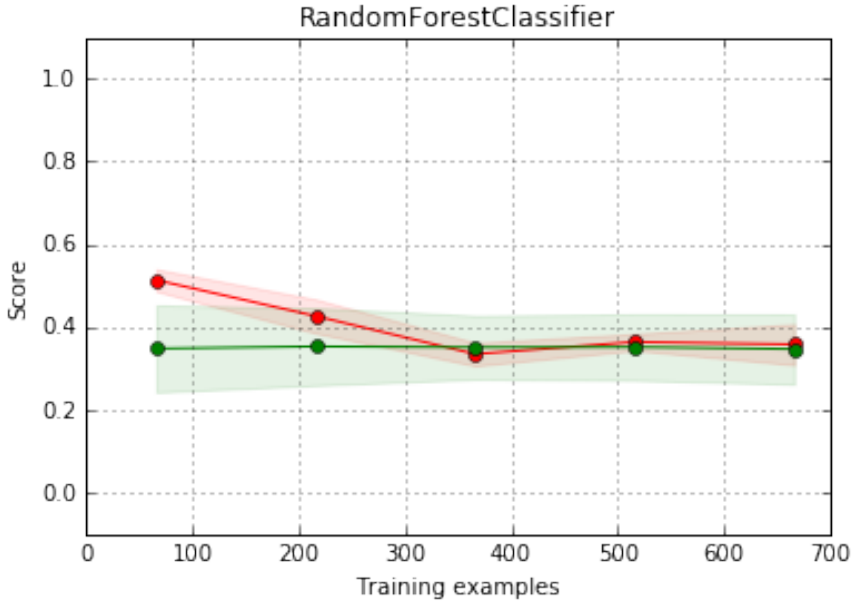
The technique can be generalized for any hyper-parameter.

1.1.9 Learning Curve

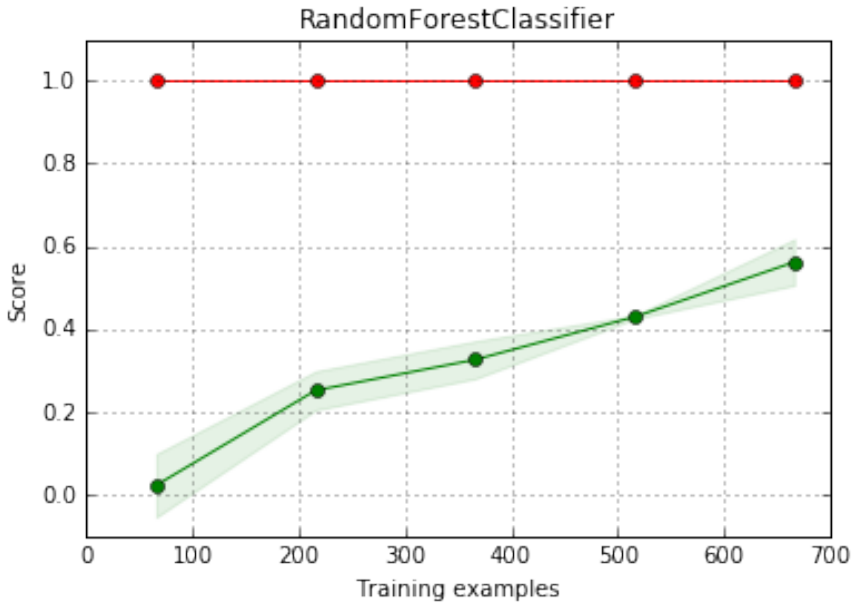
A learning curve shows the validation and training score of an estimator for varying numbers of training samples. It is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from over-fitting or under-fitting. If both the validation score and the training score converge to a value with increasing size of the training set, we will not benefit much from more training data [5].



The classifier is performing well. This classifier will not benefit from more training data.



This classifier suffers from under-fitting. This is evident as the training score and validation score have converged to a low value. Additionally, this classifier will not benefit from more training data. Illustration credit: [3]



This classifier suffers from over-fitting. This is evident as the training score is very high, and the validation score is very low. Additionally, this classifier may benefit from more training data – as shown by the increasing validation score. Illustration credit: [3].

1.1.10 Debugging [2]

Over-fitting:High Variance and Under-Fit:High Bias

1. Get more training examples - Fixes high variance.
2. Try smaller sets of features - Fixes high variance.
3. Try getting additional features - Fixes high bias.
4. Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$ - Fixes high bias.
5. Try decreasing λ - Fixes high bias.
6. Try increasing λ - Fixes high variance.

1.1.11 Other Metrics for Measuring Goodness of Classifier [2]

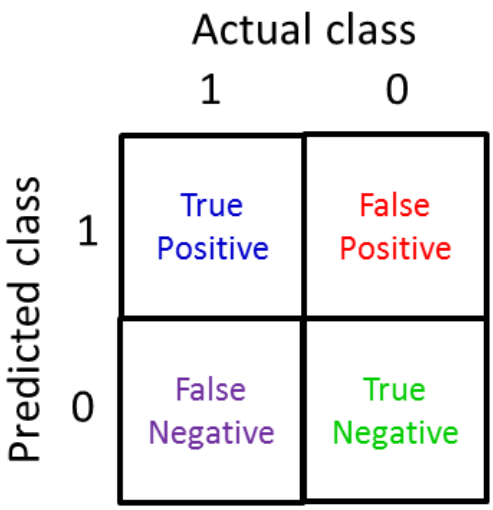


Illustration credit: [2]

Precision: *How certain is your predicted positive.*

$$\frac{\texttt{True Positive}}{\texttt{\#Predicted as Positive}} = \frac{\texttt{True Positive}}{\texttt{True Positive} + \texttt{False Positive}}$$

Recall: *How well can you find the positives in a data set.*

$$\frac{\texttt{True Positive}}{\texttt{\#Actual Positives}} = \frac{\texttt{True Positive}}{\texttt{True Positive} + \texttt{False Negative}}$$

You want high Precision and high Recall. This makes for a good classifier. i.e you want a high F_1 Score

$$F_1 = 2\frac{PR}{P+R}$$

2 Local Outlier Factor (LOF)

How the code works: `http://scikit-learn.org/stable/auto_examples/neighbors/plot_lof.html`

Extra – if you want to understand the math: `http://home.cse.ust.hk/~leichen/courses/mscbd-5002/lectures/LOF_Example.pdf`

3 Principle Component Analysis (PCA)

The math behind PCA: https://klevas.mif.vu.lt/~tomukas/Knygos/principal_components.pdf

What it does visually: http://setosa.io/ev/principal-component-analysis/

4 Support Vector Machine (SVM)

How it works: https://machinelearningmastery.com/support-vector-machines-for-machine-learning/

Intuition behind SVM (video):
https://www.coursera.org/learn/machine-learning/lecture/sHfVT/optimization-objective

References

[1] Josh Patterson Adam Gibson. Deep Learning. https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/ch01.html, 2017. [Online; accessed 12-Dec-2017].

[2] Andrew NG. Stanford University - Machine Learning. https://www.coursera.org/learn/machine-learning, 2017. [Online; accessed 12-Dec-2017].

[3] ritchieng.github.io. Learning Curve. http://www.ritchieng.com/machinelearning-learning-curve/, 2017. [Online; accessed 12-Dec-2017].

[4] Scikit-learn.org. 3.5. Validation curves: plotting scores to evaluate models scikit-learn 0.19.1. http://scikit-learn.org/stable/modules/learning_curve.html, 2017. [Online; accessed 12-Dec-2017].

[5] Scikit-learn.org. Plotting Learning Curves scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html, 2017. [Online; accessed 13-Dec-2017].