Topic - 02
Classification & Related Knowledge
Part: 01

+88 0172 6867 984

tanvir@socian.ai

SOCIAN

Binary Classification

Like the name suggests, Binary Classification are problems where there can be two classes and the instances are classified in one of these Class.

It shouldn't be mixed up with **Binary-label** classification where two-labels are to be predicted for each instance.

For example:

- Support Vector Machine
- Linear Classifiers

Confusion Matrix – Where Decision is - True/False, Right/Wrong, Yes/No - Binary

Let’s say, we are developing an AI, which will replace the Umpires in Cricket. It will analyze all the Video Feeds and give decisions.

Now, at first, we will train our AI so that it can look into all the LBWs and based on that give decisions.

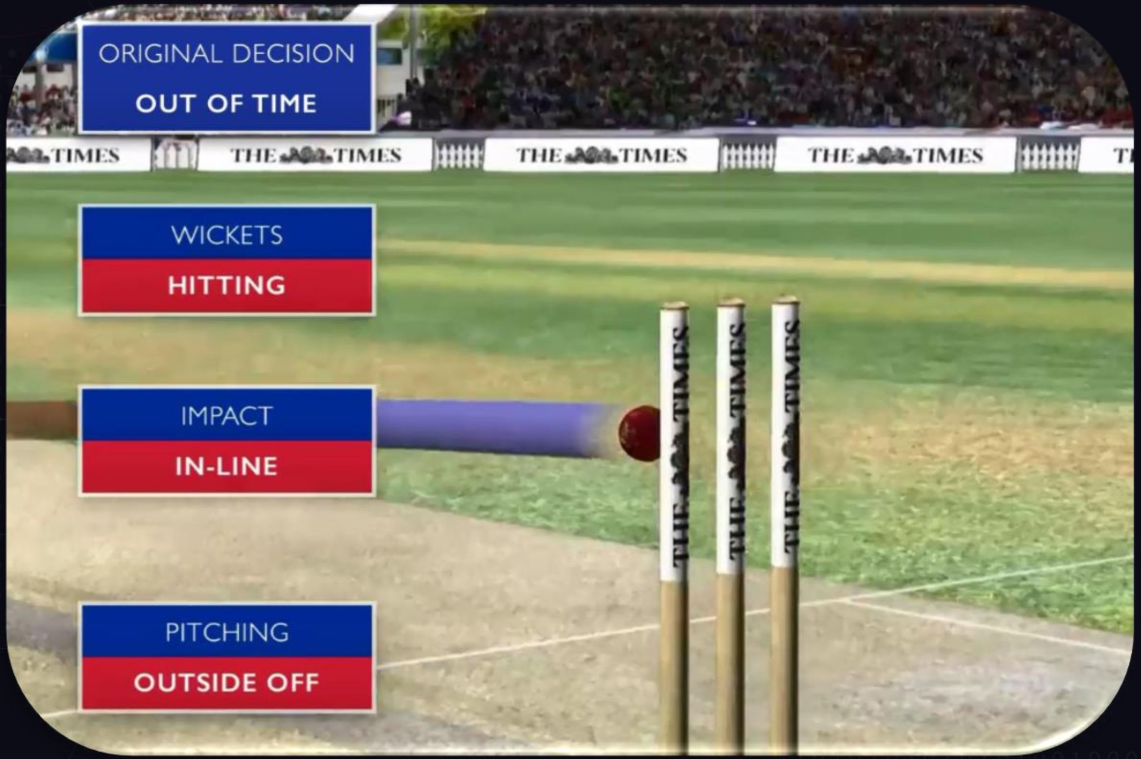
We will look into 1,00,000 cases, where the Umpires have given OUTS and NOT OUTS.

Among them, 50,000 will be OUTS and 50,000 will be NOT OUTS

80% Data (80,000) will be used for Training and 20% Data (20,000) will be used for Testing

True Positive (TP)

If the AI ***CORRECTLY/TRUELY*** gives the Decision as ***OUT***, which is ***OUT*** in reality, then the decision is ***TRUE POSITIVE***



SOCIAN

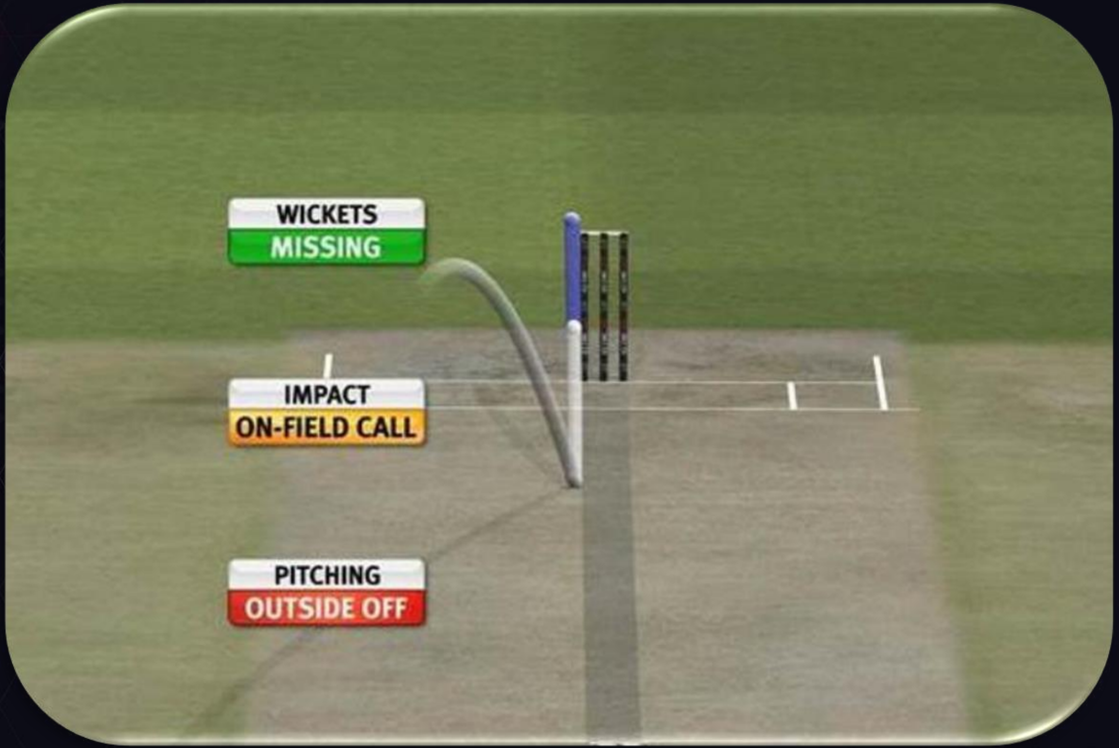
True Positive Cont.

In the time of Testing, if the AI Successfully recognizes 9,700 samples as OUT from all the samples which are OUT in reality, then –

True Positive = 9,700

True Negative (TN)

If the AI ***CORRECTLY/TRUELY*** gives the Decision as ***NOT OUT***, which is ***NOT OUT*** in reality, then the decision is ***TRUE NEGATIVE***



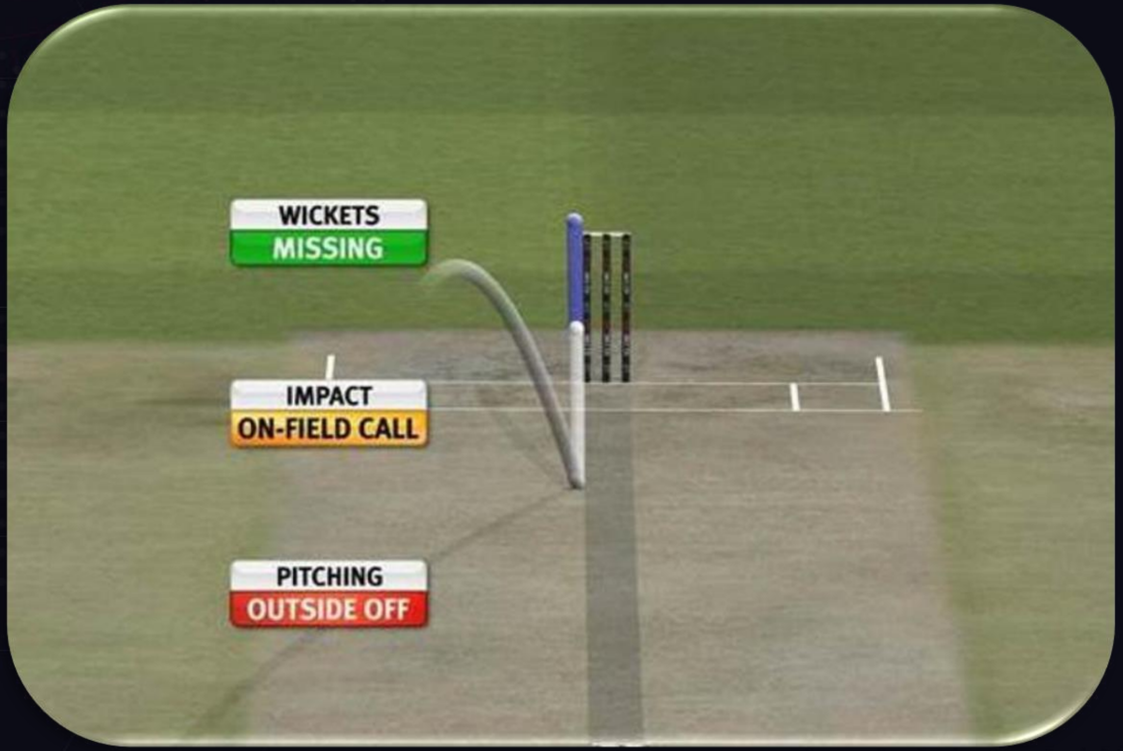
True Negative Cont.

In the time of Testing, if the AI Successfully recognizes 9,500 samples as **NOT OUT** from all the samples which are NOT OUT in reality, then –

True Negative = 9,500

False Positive (FP)

If the AI ***WRONGLY/FALSELY*** gives the Decision as ***OUT***, which is ***NOT OUT*** in reality, then the decision is ***FALSE POSITIVE***



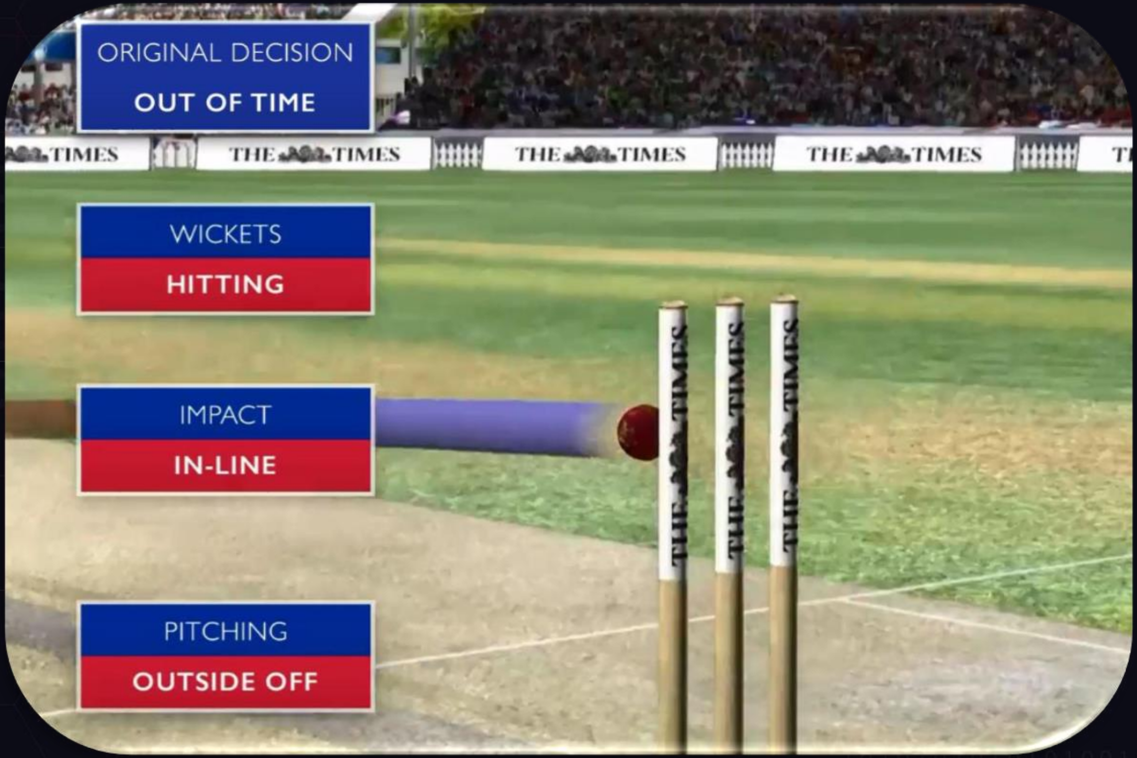
False Positive Cont.

In the time of Testing, if the AI Wrongly recognizes 500 samples as OUT from all the samples which are NOT OUT in reality, then –

False Positive = 500

False Negative (FN)

If the AI ***WRONGLY/FALSELY*** gives the Decision as ***NOT OUT***, which is ***OUT*** in reality, then the decision is ***FALSE NEGATIVE***



False Negative Cont.

In the time of Testing, if the AI Wrongly recognizes 300 samples as NOT OUT from all the samples which are OUT in reality, then –

False Negative = 300

Precision (aka Positive Predictive Value)

Here, TP = 9,700, TN = 9,500, FP = 500, FN = 300

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}}$$

$$\text{Precision} = \frac{9700}{9700 + 500} = 0.951$$

Recall (aka Sensitivity aka True Positive Rate (TPR))

Here, TP = 9,700, TN = 9,500, FP = 500, FN = 300

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Actual Positives}}$$

$$\text{Recall} = \frac{9700}{9700 + 300} = 0.97$$

The higher the Recall (TPR), the more False Positives (FPR) the classifier produces

False Positive Rate (FPR)

Here, TP = 9,700, TN = 9,500, FP = 500, FN = 300

$$\text{FPR} = \frac{\text{False Positive}}{\text{False Positive + True Negative}}$$

$$\text{FPR} = \frac{\text{False Positive}}{\text{Actual Negatives}}$$

$$\text{FPR} = \frac{500}{500 + 9500} = 0.05$$

Accuracy

Here, TP = 9,700, TN = 9,500, FP = 500, FN = 300

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total}}$$

$$\text{Accuracy} = \frac{9700 + 9500}{20000} = 0.96$$

$$= 0.96 * 100\ \% = 96\%$$

F1-score

Here, TP = 9,700, TN = 9,500, FP = 500, FN = 300.

F1-score is the harmonic mean of precision and recall.

$$\text{F1-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{F1-score} = 2 * \frac{0.951 * 0.97}{0.951 + 0.97}$$

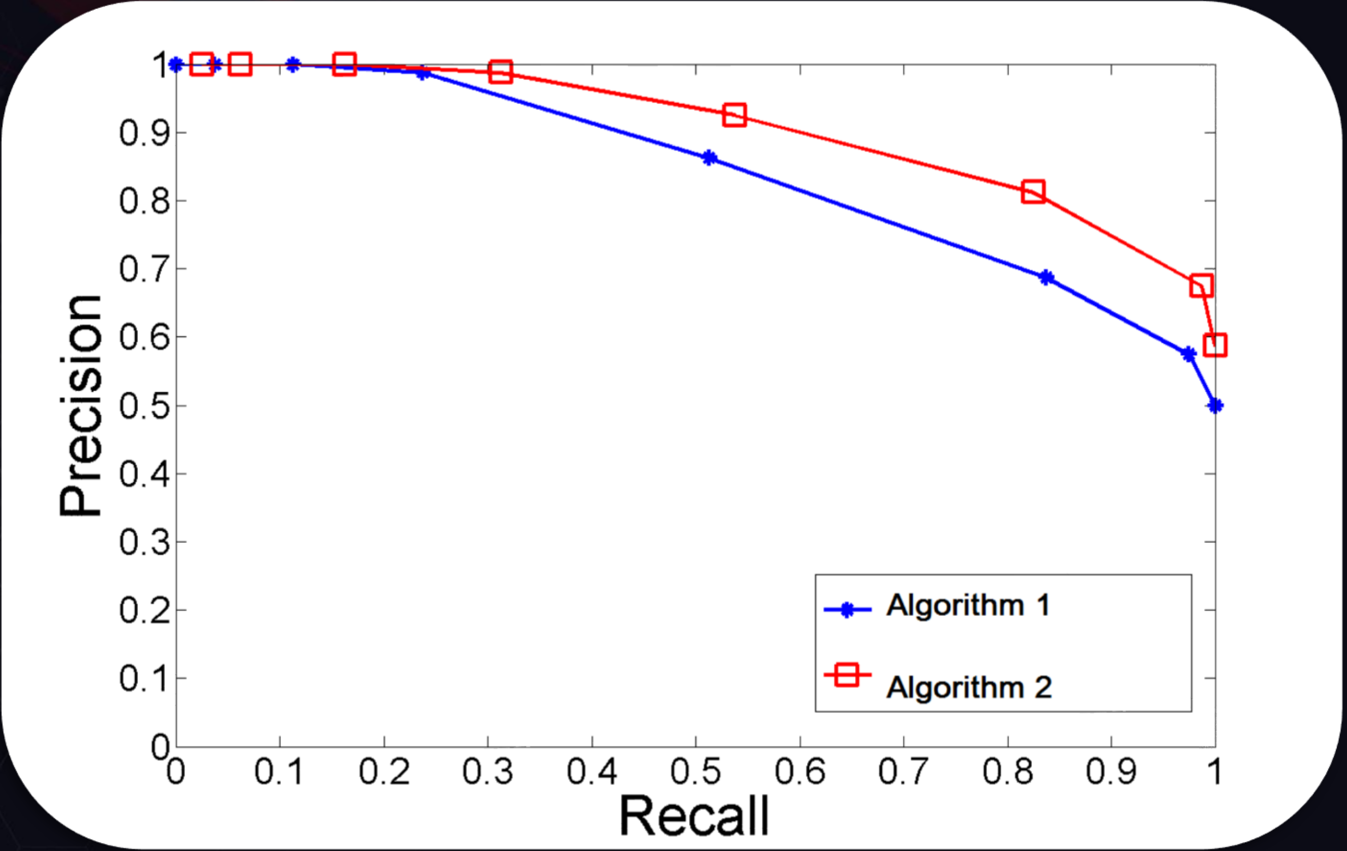
$$\text{F1-score} = 0.96 = 0.96 * 100\ \% = 96\%$$

Precision vs Recall Curve

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds.

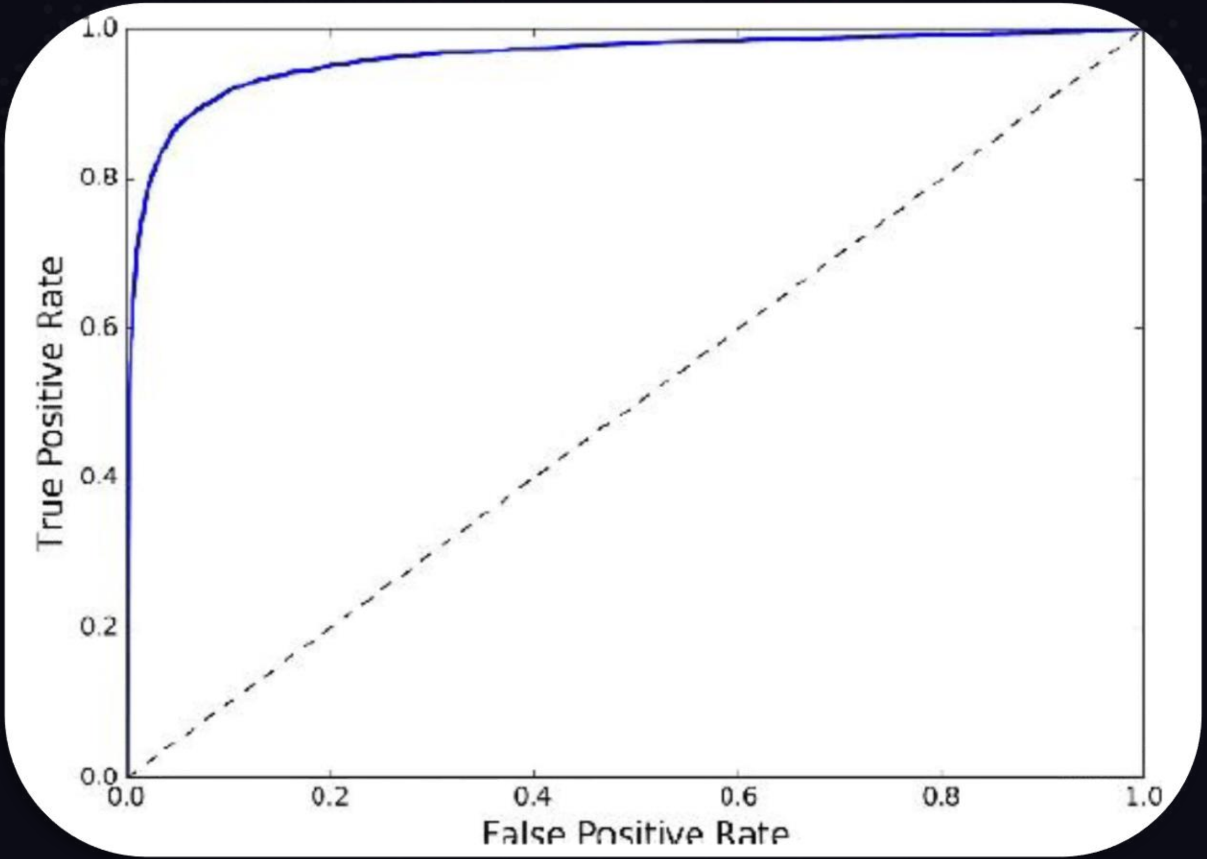
Precision-Recall curves should be used when there is a moderate to large class imbalance. A high-precision classifier is not very useful if its recall is too low.

If someone says "let's reach 99% precision," you should ask, "at what recall?"



ROC (Receiver Operating Characteristic) Curve

ROC Curve is very similar to the Precision vs Recall curve, but instead of plotting precision versus recall, the ROC curve plots the TPR against the FPR. The FPR is the ratio of negative instances that are incorrectly classified as positive.



SOCIAN

Multiclass Classification

Like the name suggests, Multiclass Classification are problems where can be more than two classes for a solution and the instances are classified into one of the Classes.

Multiclass classification should not be confused with Multi-label classification where Multiple Labels are to be predicted for each instance.

For example:

- Random Forest
- Naïve Bayes

One vs All (OvA) Strategy / One vs the Rest Strategy

There are different ways to perform Multi-class Classification operations using Multiple Binary Classifiers. One of the widely used way is One vs All approach.

For example: We will make a solution which can identify digits, starting from 0 to 9.

Normally, this can be done by applying Binary Classification for each of the Digits (10 Binary Classification for 10 Digits; a 0-detector, a 1-detector, a 2-detector, and so on) and based on that, when we will be classifying an image, we get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score.

One vs One (OvO) Strategy

Another approach to turn a Binary Classifier to a Multi-class Classifier is called One vs One Strategy.

In this case, we train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on.

If there are N classes, we need to train N × (N – 1) / 2 classifiers.

For our Handwritten Digit Recognition, this means training 45 binary classifiers! When we want to classify an image, we have to run the image through all 45 classifiers and see which class wins the most duels.

The main advantage of OvO is that each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.

Topic - 02
Classification & Related Knowledge
Part: 02

+88 0172 6867 984

tanvir@socian.ai

SOCIAN

One vs All against One vs One

Some algorithms (such as Support Vector Machine classifiers) scale poorly with the size of the training set, so for these algorithms OvO is preferred since it is faster to train many classifiers on small training sets than training few classifiers on large training sets.

For most binary classification algorithms, however, OvA is preferred.

Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs OvA (except for SVM classifiers for which it uses OvO).

If you want to force ScikitLearn to use one-versus-one or one-versus-all, you can use the ***OneVsOneClassifier*** or ***OneVsRestClassifier*** classes. Simply create an instance and pass a binary classifier to its constructor.

Confusion Matrix

A better way to evaluate the performance of a classifier is to look at the **Confusion Matrix**. The idea is to count the number of times instances of class A are classified as class B.

For example, to know the number of times the classifier confused images of 5s with 3s, you would look in the 5th row and 3rd column of the confusion matrix. To compute the confusion matrix, you first need to have a set of predictions, so they can be compared to the actual targets. Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

Scikit Learn has a function for Confusion Matrix. Just pass it the target classes (y_train_target) and the predicted classes (y_train_pred):

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_target, y_train_pred)
Output: array([[TN, FP], [ FN, TP]])
```

Multilabel Classification

In some cases you may want your classifier to output multiple classes for each instance.

For example, let’s consider a face-recognition classifier: what should it do if it recognizes several people on the same picture? Of course it should attach one label per person it recognizes.

Say the classifier has been trained to recognize three faces, Alice, Bob, and Charlie; then when it is shown a picture of Alice and Charlie, it should output [1, 0, 1] (meaning “Alice yes, Bob no, Charlie yes”).

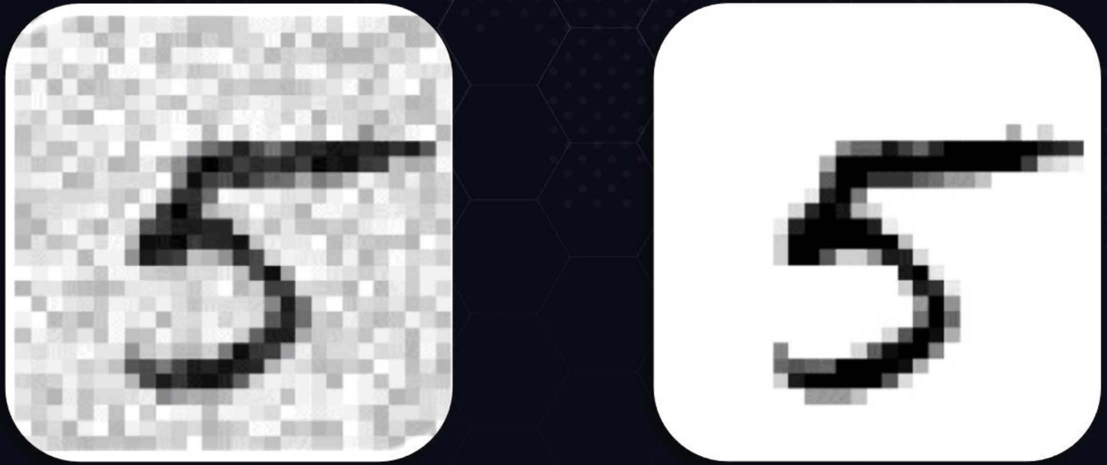
Such a classification system that outputs multiple binary labels is called a Multilabel Classification system.

Multioutput Classification

Multioutput Classification is simply a generalization of Multilabel Classification where each label can be multiclass (i.e., it can have more than two possible values).

For example: Let’s build a system that removes noise from images. It will take as input a noisy digit image, and it will (hopefully) output a clean digit image, represented as an array of pixel intensities.

Notice that the classifier’s output is multilabel (one label per pixel) and each label can have multiple values (pixel intensity ranges from 0 to 255).

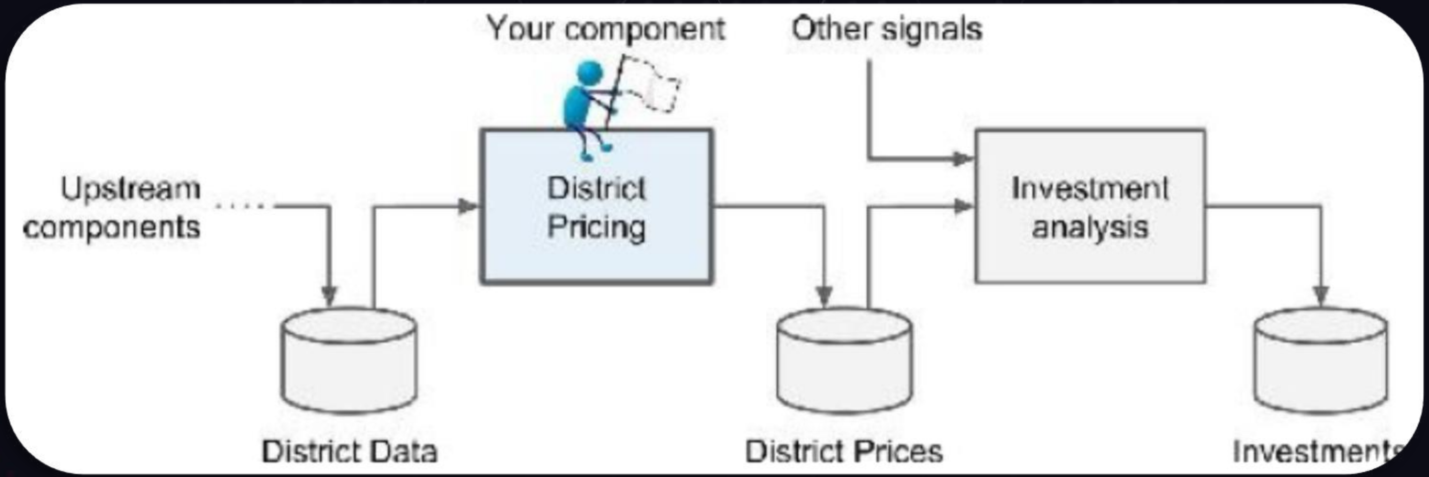


Framing the Problem

The first question to ask your boss is what exactly is the business objective; building a model is probably not the end goal. How does the company expect to use and benefit from this model? This is important because it will determine –

- How you frame the problem
- What algorithms you will select
- What performance measure you will use to evaluate
- How much effort you should spend tweaking it.

Your boss answers that your model's output (a prediction of a district's median housing price) will be fed to another Machine Learning system (see Figure 2-2), along with many other signals. This downstream system will determine whether it is worth investing in a given area or not. Getting this right is critical, as it directly affects revenue.



Data Pipeline

A sequence of data processing components is called a Data Pipeline. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many data transformations to apply.

Components typically run asynchronously. Each component pulls in a large amount of data, processes it, and spits out the result in another data store, and then some time later the next component in the pipeline pulls this data and spits out its own output, and so on. Each component is fairly self-contained: the interface between components is simply the data store. This makes the system quite simple to grasp (with the help of a data flow graph), and different teams can focus on different components.

Moreover, if a component breaks down, the downstream components can often continue to run normally (at least for a while) by just using the last output from the broken component. This makes the architecture quite robust.

On the other hand, a broken component can go unnoticed for some time if proper monitoring is not implemented. The data gets stale and the overall system's performance drops.

Performance Measure – Root Mean Square Error (RMSE)

A typical performance measure for regression problems is the Root Mean Square Error (RMSE). It gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors. Equation 2-1 shows the mathematical formula to compute the RMSE.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

Here, m is the number of instances in the dataset you are measuring the RMSE on.
For example, if you are evaluating the RMSE on a validation set of 2,000 districts, then m = 2,000.

x(i) is a vector of all the feature values (excluding the label) of the ith instance in the dataset, and y(i) is its label (the desired output value for that instance).

Computing the root of a sum of squares (RMSE) corresponds to the Euclidian norm: it is the notion of distance you are familiar with. It is also called the ℓ2 norm,

Performance Measure – Mean Absolute Error (MAE) / Avg. Absolute Deviation

Another Performance measure is Mean Absolute Error.

For example: In the house Price problem, there are many outlier districts. In that case, you may consider using the Mean Absolute Error (also called the Average Absolute Deviation)

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m} \left| h(\mathbf{x}^{(i)}) - y^{(i)} \right|$$

Computing the sum of absolutes (MAE) corresponds to the ℓ1 norm, noted · 1.

It is sometimes called the Manhattan norm because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.

Topic - 02
Classification & Related Knowledge
Part: 03

+88 0172 6867 984

tanvir@socian.ai

SOCIAN

RMSE vs MAE

The higher the norm index, the more it focuses on large values and neglects small ones.

This is why the RMSE is more sensitive to outliers than the MAE.

But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.

Checking, Double Checking

Before you go to production, it is always wise to check and double check your data.

For example, let’s say, in your system, you have kept one particular value in the non-desired format or your data contains more noise than the allowance limit.

Again, let’s say your task was to give output the CATEGORIES of flats as LOW, MEDIUM or HIGH. But you ended up getting outputting the PRICES of the flats.

If you output the CATEGORIES, then it is a Regression Task. If you output the PRICES, then it is a Classification Task.

You don’t want to find this out after working on a Classification system for months.

Fortunately, after talking with the team or friends, you are confident that you do indeed need the CATEGORIES as well, not just PRICES. You output the CATEGORIES as well!

Awesome! You’re all set, the lights are green, and you can start coding now!

Feature Scaling

Before you start training, in some of the cases, you have to transform your data.

One of the most important transformations you need to apply to your data is FEATURE SCALING. With few exceptions, Machine Learning algorithms don’t perform well when the input numerical attributes have very different scales.

For example, in the case of House Pricing problem, if we look into the Housing Data:

The ***TOTAL NUMBER OF ROOMS*** ranges from about 6 to 39,320, while the ***MEDIAN INCOMES*** only range from 0 to 15.

Scaling the ***TARGET VALUES*** is generally not required.

There are two common ways to get all attributes to have the same scale:
- Min-Max Scaling (Used in most cases)
- Standardization

Min-Max Scaling (aka Normalization)

Min-Max Scaling is quite simple. In Min-Max Scaling, values are shifted and rescaled so that they end up ranging from ***0 to 1***.

We do this by subtracting the ***MIN VALUE*** from the ***CURRENT VALUE*** and dividing by the ***MAX VALUE*** minus the ***MIN VALUE***.

$$\text{Min Max Scaling} = \frac{\text{CURRENT VALUE} - \text{MIN VALUE}}{\text{MAX VALUE} - \text{MIN VALUE}}$$

Scikit-Learn provides a transformer called ***MinMaxScaler*** for this operation.

It has a ***feature_range hyperparameter*** that lets you change the range if you don't want 0–1 for some reason.

Better Evaluation using Cross-Validation

One way to evaluate the any model would be to use the ***train_test_split*** function to split the training set into a smaller training set and a validation set, then train your models against the smaller training set and evaluate them against the validation set. It's a bit of work, but nothing too difficult and it would work fairly well.

A great alternative is to use Scikit-Learn's ***cross-validation*** feature. The following code performs K-fold cross-validation: it randomly splits the training set into 10 distinct subsets called folds, then it trains and evaluates the Model 10 times, picking a different fold for evaluation every time and training on the other 9 folds.

Grid Search

When you will be finding the most suitable value of Hyperparameter, one way is manually inputting different values for Testing, which is not very helpful. You may feel tired after some time.

Instead of doing that you can use the Scikit-Learn’s GridSearchCV to search for you.

All you need to do is tell it which hyperparameters you want it to experiment with, and what values to try out, and it will evaluate all the possible combinations of hyperparameter values, using cross-validation.

Randomized Search

When the hyperparameter search space is large, it is often preferable to use RandomizedSearchCV instead of GridSearchCV. This class can be used in much the same way as the GridSearchCV class.

But instead of trying out all possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration.

This approach has two main benefits: If you let the randomized search run for, say, 1,000 iterations, this approach will explore 1,000 different values for each hyperparameter (instead of just a few values per hyperparameter with the grid search approach). You have more control over the computing budget you want to allocate to hyperparameter search, simply by setting the number of iterations.

Correlation Coefficient / Pearson's R

If the data you are working with is not too large, you can easily compute the standard Correlation Coefficient (also called Pearson's R) between every pair of attributes using the ***corr()*** method.

corr_matrix = housing.corr()

The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation. For example, the ***MEDIAN HOUSE VALUE*** tends to go up when the ***Median Income*** goes up.

When the coefficient is close to –1, it means that there is a strong negative correlation. For example: There is a small negative correlation between the ***latitude*** and the ***Median House Value***.

Finally, coefficients close to zero mean that there is no linear correlation

Attribute Combinations

One final step before starting your Training is to try out various ***Attribute Combinations***.

For example, the ***Total Number of Rooms in a District*** is not very useful if you don't know how many ***Households*** there are. What you really want is the **Number of Rooms per Household**.

Similarly, the ***Total Number of Bedrooms*** by itself is not very useful: you probably want to compare it to the ***Number of Rooms***. And the **Population Per Household** also seems like an interesting attribute combination to look at.

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

Doing this can improve the Accuracy later on.