

Evaluating Model Performance

Evaluating Model Performance

- *The process of evaluating machine learning algorithms is needed
- *As algorithms have varying strengths and weaknesses, tests should distinguish among the learners.
- * It is also important to forecast how a learner will perform on future data.

Measuring performance for classification

- * We measured classifier accuracy by dividing the proportion of correct predictions by the total number of predictions.
- * This indicates the percentage of cases in which the learner is right or wrong.





Confusion matrix

- * A **confusion matrix** is a table that categorizes predictions according to whether they match the actual value.
- * One of the table's dimensions indicates the possible categories of predicted values, while the other dimension indicates the same for actual values.
- * Although we have only seen 2 x 2 confusion matrices so far, a matrix can be created for models that predict any number of class values.










Confusion matrix

- * The figure depicts the familiar confusion matrix for a two-class binary model as well as the 3 x 3 confusion matrix for a three-class model.

Two Classes

		Predicted Class	
		A	B
Actual Class	A		
	B		

Three Classes

		Predicted Class		
		A	B	C
Actual Class	A			
	B			
	C			

Confusion matrix

- * When the predicted value is the same as the actual value, it is a correct classification.
- * Correct predictions fall on the diagonal in the confusion matrix (denoted by **O**).
- * The off-diagonal matrix cells (denoted by **X**) indicate the cases where the predicted value differs from the actual value.
- * These are incorrect predictions.
- * The performance measures for classification models are based on the counts of predictions falling on and off the diagonal in these tables

Confusion matrix

- * The most common performance measures consider the model's ability to discern one class versus all others.
- * The class of interest is known as the **positive** class, while all others are known as **negative**.

Confusion matrix

- * The relationship between the positive class and negative class predictions can be depicted as a 2 x 2 confusion matrix that tabulates whether predictions fall into one of the four categories:
 - **True Positive (TP):** Correctly classified as the class of interest
 - **True Negative (TN):** Correctly classified as not the class of interest
 - **False Positive (FP):** Incorrectly classified as the class of interest
 - **False Negative (FN):** Incorrectly classified as not the class of interest

Confusion matrix

- * For the spam classifier, the positive class is spam, We can then imagine the confusion matrix as in the diagram:

		Predicted to be Spam	
		no	yes
Actually Spam	no	<div>TN</div> <div>True Negative</div>	<div>FP</div> <div>False Positive</div>
	yes	<div>FN</div> <div>False Negative</div>	<div>TP</div> <div>True Positive</div>

Using confusion matrices to measure performance

- * With the 2 x 2 confusion matrix, we can formalize our definition of prediction **accuracy** (sometimes called the **success rate**) as:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- * the terms *TP*, *TN*, *FP*, and *FN* refer to the number of times the model's predictions fell into each of these categories.
- * The accuracy is therefore a proportion that represents the number of true positives and true negatives, divided by the total number of predictions.

Using confusion matrices to measure performance

- * The **error rate** or the proportion of the incorrectly classified examples is specified as

$$\textit{error rate} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \textit{accuracy}$$

- * The error rate can be calculated as one minus the accuracy.
- * A model that is correct 95 percent of the time is incorrect 5 percent of the time.

The kappa statistic

- * Kappa is such a measure of “true” agreement.
- * It indicates the proportion of agreement beyond that expected by chance, that is, the *achieved* beyond-chance agreement as a proportion of the *possible* beyond-chance agreement. It takes the form:
- * $k = \text{observed agreement} - \text{chance agreement} / 1 - \text{chance agreement}$
- *
$$\kappa = \frac{P_o - P_c}{1 - P_c}$$

where P_o is the proportion of observed agreements and P_c is the proportion of agreements expected by chance.

The kappa statistic

- * Kappa values range from 0 to a maximum of 1, which indicates perfect agreement between the model's predictions and the true values.
- * Values less than one indicate imperfect agreement.

The kappa statistic

- * One common interpretation is shown as follows:
 - * Poor agreement = less than 0.20
 - * Fair agreement = 0.20 to 0.40
 - * Moderate agreement = 0.40 to 0.60
 - * Good agreement = 0.60 to 0.80
 - * Very good agreement = 0.80 to 1.00
- * It's important to note that these categories are subjective. While a "good agreement" may be more than adequate to predict someone's favorite ice cream flavor, "very good agreement" may not suffice if your goal is to identify birth defects.

The kappa statistic

$$P_o \leftarrow 1203 + 152 / 1390$$

$$= 0.974 \text{ (accuracy)}$$

$$P_c \leftarrow 0.868 * 0.888 + 0.132 * \\ 0.112[(1207/1390 * 1234/1390) + \\ (183/1390 * 156/1390)]$$

$$= 0.785568$$

P_c is 0.786, by chance alone, we would expect the observed and actual values to agree about 78.6 percent of the time.

$$k \leftarrow (P_o - P_c) / (1 - P_c)$$

$$0.8787494$$

	Ham	Spam	Total
Ham	1203	4	1207
Spam	31	152	183
Total	1234	156	1390

The kappa statistic

- * The probability of both choosing ham is:

$$Pr(actual\ type\ is\ ham) * Pr(predicted\ type\ is\ ham)$$

- * The probability of both choosing spam is:

$$Pr(actual\ type\ is\ spam) * Pr(predicted\ type\ is\ spam)$$

- * P_c is calculated as the sum of the probabilities that by chance the predicted and actual values agree that the message is either spam or ham.

Sensitivity and specificity

- * The **sensitivity** of a model (also called the **true positive rate**) measures the proportion of positive examples that were correctly classified.

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- * It is calculated as the number of true positives divided by the total number of positives, both correctly classified (the true positives) as well as incorrectly classified (the false negatives):

Sensitivity and specificity

- * The **specificity** of a model (also called the **true negative rate**) measures the proportion of negative examples that were correctly classified.

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- * As with sensitivity, this is computed as the number of true negatives, divided by the total number of negatives—the true negatives plus the false positives:

Sensitivity and specificity

* The calculation for sensitivity is:

* $152 / (152 + 31)$

	ham	spam
-ve ham	1203 (TN)	4 (FP)
+ve spam	31 (FN)	152(TP)

0.8306011

for specificity we can calculate:

$1203 / (1203 + 4)$

0.996686

Sensitivity and specificity

- * Sensitivity and specificity range from 0 to 1, with values close to 1 being more desirable.
- * For example, in this case, the sensitivity of 0.831 implies that 83.1 percent of the spam messages were correctly classified.
- * Similarly, the specificity of 0.997 implies that 99.7 percent of the nonspam messages were correctly classified or, alternatively, 0.3 percent of the valid messages were rejected as spam.

Sensitivity and specificity

- * The idea of rejecting 0.3 percent of valid SMS messages may be unacceptable, or it may be a reasonable trade-off given the reduction in spam.
- * Sensitivity and specificity provide tools for thinking about such trade-offs.
- * Typically, changes are made to the model and different models are tested until you find one that meets a desired sensitivity and specificity threshold.

Precision and recall

- * Used primarily in the context of information retrieval, these statistics are intended to provide an indication of how interesting and relevant a model's results are, or whether the predictions are diluted by meaningless noise.
- * The **precision** (also known as the **positive predictive value**) is defined as the proportion of positive examples that are truly positive; in other words, when a model predicts the positive class, how often is it correct?
- * A precise model will only predict the positive class in cases that are very likely to be positive. It will be very trustworthy.

Precision and recall

- * What would happen if the model was very imprecise.
- * Over time, the results would be less likely to be trusted.
- * In the case of the SMS spam filter, high precision means that the model is able to carefully target only the spam while ignoring the ham.

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

Precision and recall

- * **Recall** is a measure of how complete the results are.

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

- * This is defined as the number of true positives over the total number of positives.
- * You may have already recognized this as the same as sensitivity.
- * In this case, the interpretation differs slightly.

Precision and recall

- * A model with a high recall captures a large portion of the positive examples, meaning that it has wide breadth.
- * For example, a search engine with a high recall returns a large number of documents pertinent to the search query.
- * Similarly, the SMS spam filter has a high recall if the majority of spam messages are correctly identified.

Precision and recall

- * We can calculate precision and recall from the confusion matrix.

- * Assuming that spam is the positive class, the precision is:

$$152 / (152 + 4)$$

0.974359

- * The recall is:

$$152 / (152 + 31)$$

rec

0.8306011

Precision and recall

- * It is difficult to build a model with both high precision and high recall.
- * In contrast, having both high precision and recall at the same time is very challenging.
- * It is therefore important to test a variety of models in order to find the combination of precision and recall that will meet the needs of your project.

Precision and recall

The F-measure

- * A measure of model performance that combines precision and recall into a single number is known as the **F-measure** (also sometimes called the **F1 score** or **F-score**).
- * The F-measure combines precision and recall using the **harmonic mean**, a type of average that is used for rates of change.
- * The harmonic mean is used rather than the common arithmetic mean since both precision and recall are expressed as proportions between zero and one, which can be interpreted as

$$\text{F-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{recall} + \text{precision}} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}$$

Precision and recall

The F-measure

- * To calculate the F-measure, use the precision and recall values computed previously:

$$(2 * \text{prec} * \text{rec}) / (\text{prec} + \text{rec})$$

0.8967552

- * This comes out exactly the same as using the counts from the confusion matrix:

$$(2 * 152) / (2 * 152 + 4 + 31)$$

0.8967552

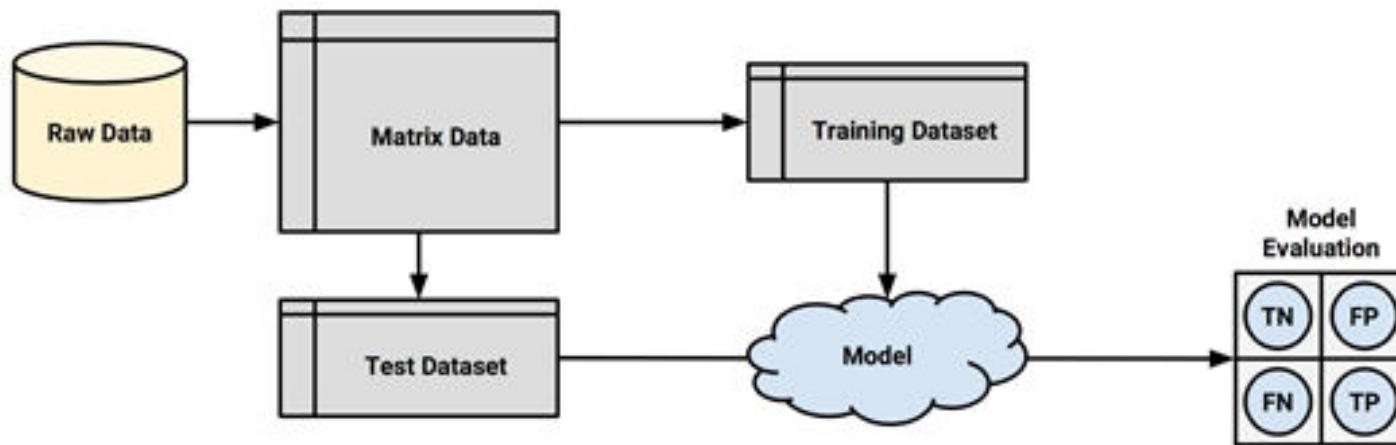
Precision and recall

The F-measure

- * F-measure describes the model performance in a single number, it provides a convenient way to compare several models side by side.
- * However, this assumes that equal weight should be assigned to precision and recall, an assumption that is not always valid.
- * It is possible to calculate F-scores using different weights for precision and recall, but choosing the weights could be tricky at the best and arbitrary at worst.
- * A better practice is to use measures such as the F-score in combination with methods that consider a model's strengths and weaknesses more globally, such as those described in the next section.

The holdout method

- * The procedure of partitioning data into training and test datasets is known as the **holdout method**.



- * As shown in the diagram, the **training dataset** is used to generate the model, which is then applied to the **test dataset** to generate predictions for evaluation.

The holdout method

- * Typically, about one-third of the data is held out for testing, and two-thirds is used for training, but this proportion can vary depending on the amount of available data.
- * To ensure that the training and test data do not have systematic differences, their examples are randomly divided into the two groups.
- * For the holdout method to result in a truly accurate estimate of the future performance, the performance on the test dataset should not be allowed to influence the model.

The holdout method

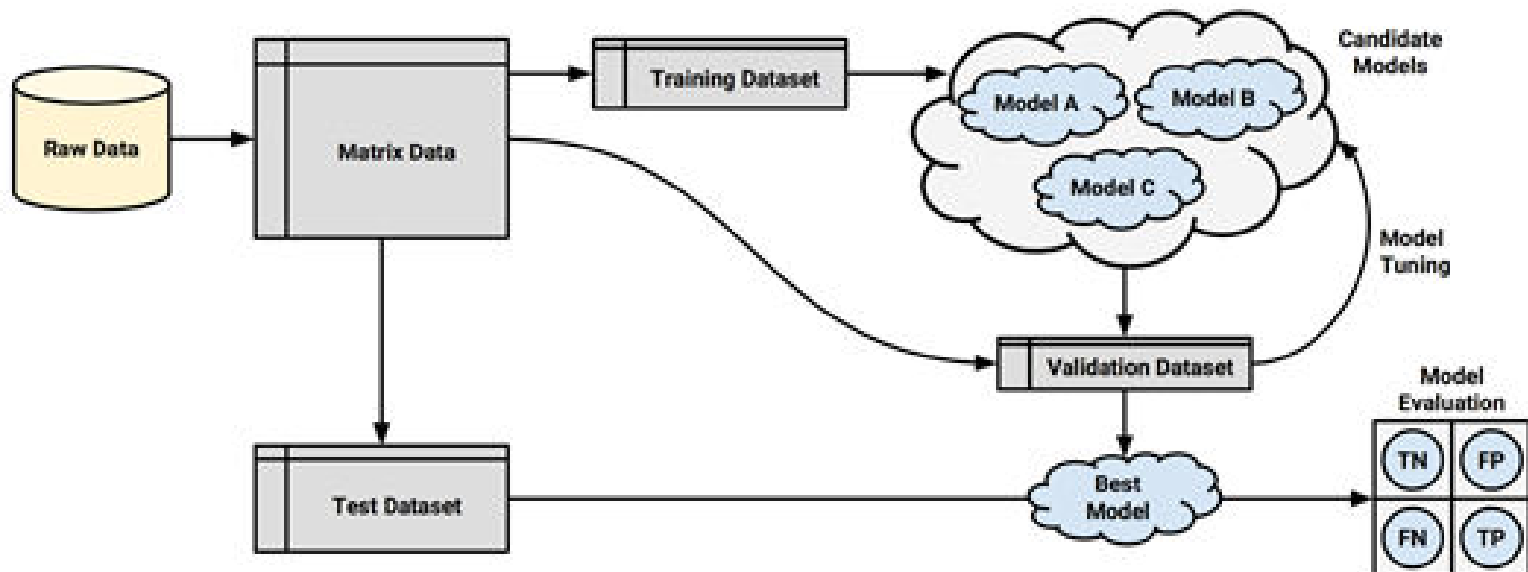
- * Some times the best model is selected based on the results of repeated testing.
- * For example, suppose we built several models on the training data, and selected the one with the highest accuracy on the test data.
- * Because we have cherry-picked the best result, the test performance is not an unbiased measure of the performance on unseen data.

The holdout method

- * To avoid this problem, it is better to divide the original data so that in addition to the training datasets and the test datasets, a **validation dataset** is available.
- * The validation dataset would be used for iterating and refining the model or models chosen, leaving the test dataset to be used only once as a final step to report an estimated error rate for future predictions.
- * A typical split between training, test, and validation would be 50 percent, 25 percent, and 25 percent, respectively.

The holdout method

- * A simple method to create holdout samples uses random number generators to assign records to partitions.



The holdout method

- * One problem with hold out sampling is that each partition may have larger or smaller proportion of some classes
- * Small proportion class may be omitted from the training and the model will not be able to learn from the class
- * In order to reduce the chance of this occurring, a technique called **stratified random sampling** can be used.
- * stratified random sampling guarantees that the random partitions have nearly the same proportion of each class as the full dataset, even when some classes are small.

The holdout method

- * A technique called **repeated holdout** is sometimes used to mitigate the problems of randomly composed training datasets.
- * The repeated holdout method is a special case of the holdout method that uses the average result from several random holdout samples to evaluate a model's performance.
- * As multiple holdout samples are used, it is less likely that the model is trained or tested on non representative data.

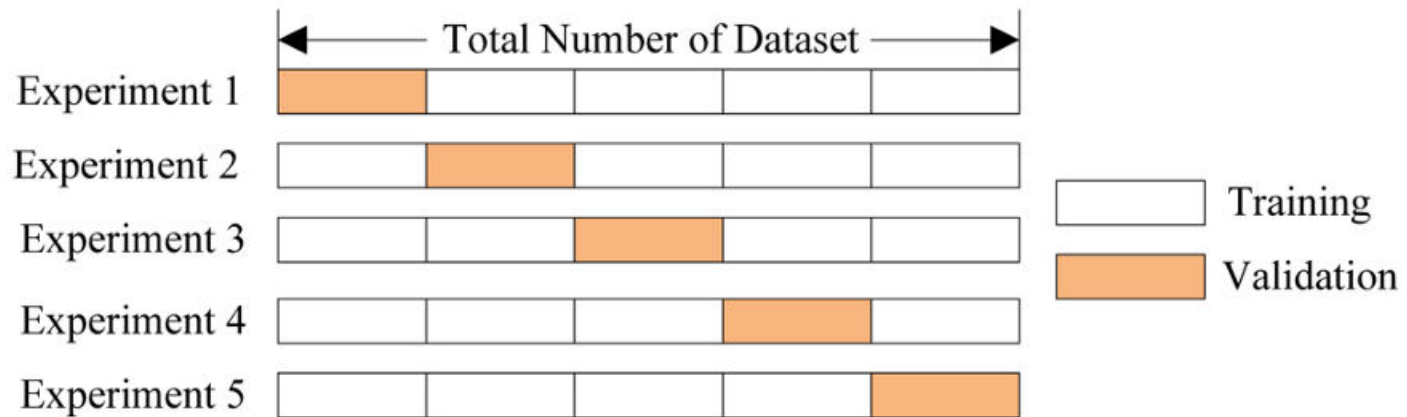
Cross-validation

- * The repeated holdout is the basis of a technique known as **k-fold cross-validation** (or **k-fold CV**), which has become the industry standard for estimating model performance.
- * Rather than taking repeated random samples that could potentially use the same record more than once, k-fold CV randomly divides the data into k to completely separate random partitions called **folds**.
- * Although k can be set to any number, by far, the most common convention is to use **10-fold cross-validation** (10-fold CV).
Why 10 folds?
- * The reason is that the empirical evidence suggests that there is little added benefit in using a greater number.

Cross-validation

- * For each of the 10 folds (each comprising 10 percent of the total data), a machine learning model is built on the remaining 90 percent of data.
- * The fold's matching 10 percent sample is then used for model evaluation.
- * After the process of training and evaluating the model has occurred for 10 times (with 10 different training/testing combinations), the average performance across all the folds is reported.

5 fold CV



Bootstrap sampling

- * A slightly less frequently used alternative to k-fold CV is known as **bootstrap sampling**, the **bootstrap** or **bootstrapping** for short.
- * These refer to the statistical methods of using random samples of data to estimate the properties of a larger set.
- * When this principle is applied to machine learning model performance, it implies the creation of several randomly selected training and test datasets, which are then used to estimate performance statistics.
- * The results from the various random datasets are then averaged to obtain a final estimate of future performance.

Bootstrap sampling

- * So, what makes this procedure different from k-fold CV?
- * Whereas cross-validation divides the data into separate partitions in which each example can appear only once, the bootstrap allows examples to be selected multiple times through a process of **sampling with replacement**.
- * This means that from the original dataset of n examples, the bootstrap procedure will create one or more new training datasets that will also contain n examples, some of which are repeated.
- * The corresponding test datasets are then constructed from the set of examples that were not selected for the respective training datasets.

Bootstrap sampling

- * Using sampling with replacement as described previously, the probability that any given instance is included in the training dataset is 63.2 percent.
- * Consequently, the probability of any instance being in the test dataset is 36.8 percent.
- * In other words, the training data represents only 63.2 percent of available examples, some of which are repeated.
- * In contrast to 10-fold CV, which uses 90 percent of the examples for training, the bootstrap sample is less representative of the full dataset.

Bootstrap sampling

- * Because a model trained on only 63.2 percent of the training data is likely to perform worse than a model trained on a larger training set, the bootstrap's performance estimates may be substantially lower than what would be obtained when the model is later trained on the full dataset.
- * A special case of bootstrapping known as the **0.632 bootstrap** accounts for this by calculating the final performance measure as a function of performance on both the training data (which is overly optimistic) and the test data (which is overly pessimistic).

Bootstrap sampling


- * The final error rate is then estimated as:

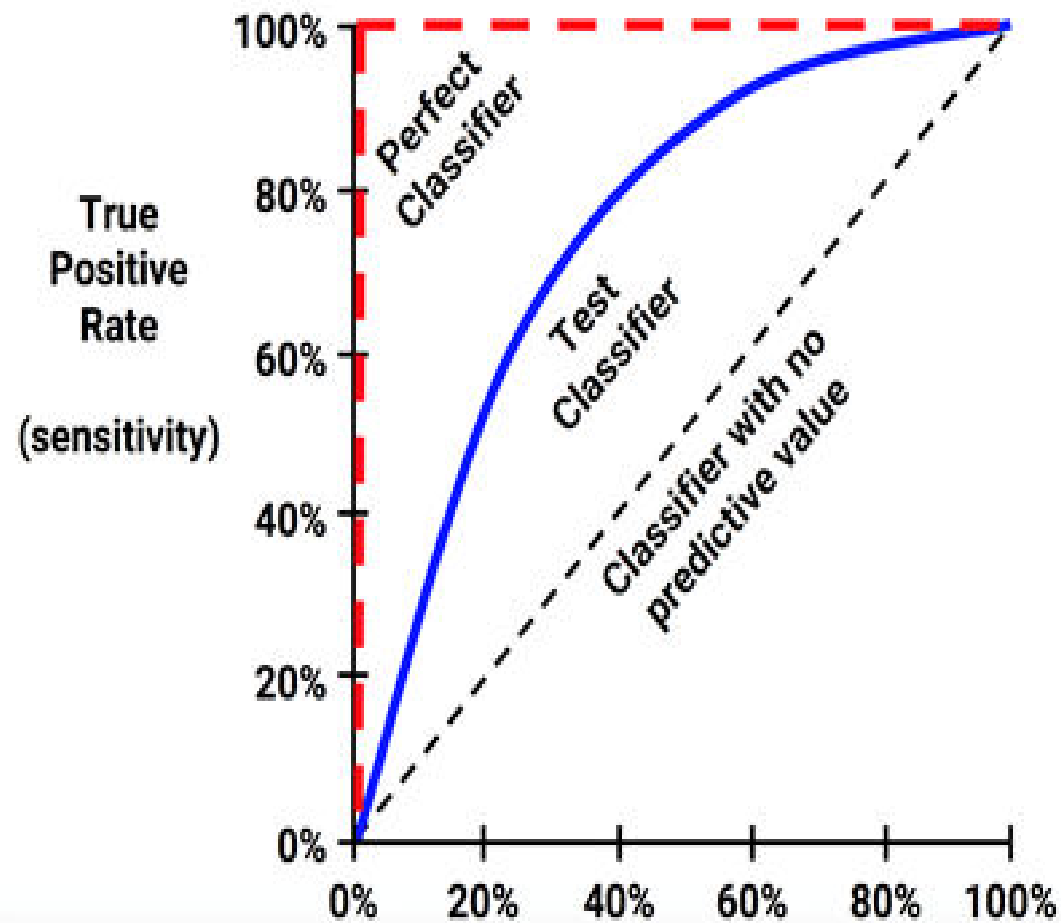
$$error = 0.632 * error_{test} + 0.368 * error_{train}$$


- * One advantage of bootstrap over cross-validation is that it tends to work better with very small datasets.
- * Additionally, bootstrap sampling has applications beyond performance measurement.

ROC Curve

- * A useful tool to examine the trade-off between the detection of true positives while avoiding false positives.
- * It is a plot of the false positive rate (x-axis) (1-sensitivity) versus the true positive rate (y-axis) (sensitivity) for a number of different candidate threshold values between 0.0 and 1.0. Put another way, it plots the false alarm rate versus the hit rate.
- * The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.
- *
$$\text{True Positive Rate} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$


- 
- * The false positive rate is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives.
 - * False Positive Rate = False Positives / (False Positives + True Negatives)
 - * it summarizes how often a positive class is predicted when the actual outcome is negative.
 - * Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives.
 - * Larger values on the y-axis of the plot indicate higher true positives and lower false negatives.
 - *






First, the diagonal line from the bottom-left to the top-right corner of the diagram represents a **classifier with no predictive value**. This type of classifier detects true positives and false positives at exactly the same rate, implying that the classifier cannot discriminate between the two. This is the baseline by which other classifiers may be judged. ROC curves falling close to this line indicate models that are not very useful. The **perfect classifier** has a curve that passes through the point at a 100 percent true positive rate and 0 percent false positive rate. It is able to correctly identify all of the positives before it incorrectly classifies any negative result. Most real-world classifiers are similar to the test classifier and they fall somewhere in the zone between perfect and useless.

The closer the curve is to the perfect classifier, the better it is at identifying positive values. This can be measured using a statistic known as the **area under the ROC curve** (abbreviated **AUC**).

- 
- * In a smog prediction system, we may be far more concerned with having low false negatives than low false positives. A false negative would mean not warning about a smog day when in fact it is a high smog day, leading to health issues in the public that are unable to take precautions. A false positive means the public would take precautionary measures when they didn't need to.

Improving Model Performance

- * How to automate model performance tuning by systematically searching for the optimal set of training conditions
- * The methods for combining models into groups that use teamwork to tackle tough learning tasks-Meta learning
- * How to apply a variant of decision trees, which has quickly become popular due to its impressive performance

- 
- * Model performance can be improved through hyperparameter tuning
 - * A hyperparameter is a parameter whose value is set before the learning process begins.
 - * Hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance
 - * Model *parameters* are learned during training — such as the slope and intercept in a linear regression but *hyperparameters* must be set before training
 - * The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering.




Baseline model with default parameters:

```
* random_forest =  
  RandomForestClassifier(random_state=1).fit(X_train, y_train)  
random_forest.score(X_test, y_test)
```

GRID SEARCH

* One traditional and popular way to perform hyperparameter tuning is by using an Exhaustive Grid Search from Scikit learn. This method tries every possible combination of each set of hyper-parameters. Using this method, we can find the best set of values in the parameter search space. This usually uses more computational power and takes a long time to run since this method needs to try every combination in the grid size.


*



```
* parameters = {'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],  
  'criterion': ['gini', 'entropy'],  
  'max_features': [0.3, 0.5, 0.7, 0.9],  
  'min_samples_leaf': [3, 5, 7, 10, 15],  
  'min_samples_split': [2, 5, 10],  
  'n_estimators': [50, 100, 200, 400, 600]}
```

```
from sklearn.model_selection import GridSearchCV  
grid_search = RandomForestClassifier()  
grid_search = GridSearchCV(  
  grid_search,  
  parameters,  
  cv=5,  
  scoring='accuracy', n_jobs=-1)
```

```
grid_result= grid_search.fit(X_train, y_train)  
print('Best Params: ', grid_result.best_params_)  
print('Best Score: ', grid_result.best_score_)
```



Best Params: {'criterion': 'gini', 'max_depth': 90, 'max_features': 'log2', 'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 50}

Best Score: 0.8412587412587413

CPU times: user 3min 49s, sys: 6.52 s, total: 3min 56s


Wall time: 4h 49min 25s



Randomized Search

- * The main difference in the RandomizedSearch CV, when compared with GridCV, is that instead of trying every possible combination, this chooses the hyperparameter sample combinations randomly from grid space. Because of this reason, there is no guarantee that we will find the best result like Grid Search. But, this search can be extremely effective in practice as computational time is very less.
- * The computational time and model performs mainly depends on the n_iter value. Because this value specifies how many times the model should search for parameters. If this value is high, there is a better chance of getting better accuracy, but also this comes with more computational power.

*



```
* from sklearn.model_selection import RandomizedSearchCV
random_search=RandomizedSearchCV(estimator =
RandomForestClassifier(),
param_distributions=parameters,verbose=1, n_jobs=-1,
```

```
Best Score: 83.84221412390428
```

```
Best Params: {'n_estimators': 400, 'min_samples_split': 5, 'min_samples_leaf': 5, 'max_features': 0.3, 'max_depth': 50, 'criterion': 'entropy', 'bootstrap': True}
```

```
CPU times: user 5.5 s, sys: 178 ms, total: 5.68 s
```

```
Wall time: 5min 20s
```

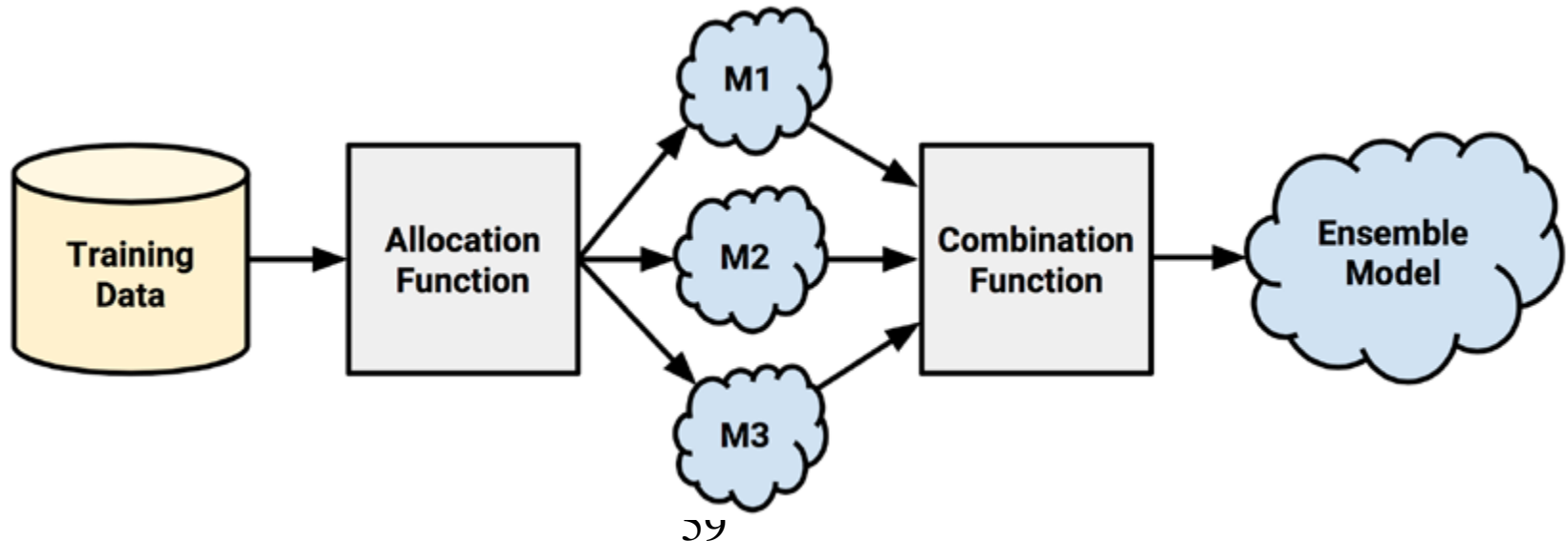
```
print('Best Params: ', random_result.best_params_)
```

Understanding ensembles

- * In order to improve the performance of learning models, several models can be combined to form a powerful team
- * Most of the models exhibits a unique bias to a learning task, it may readily learn one subset of examples, but have trouble with another
- * The ensemble methods are based on the idea that by intelligently combining the talents of several diverse team members, a strong team of multiple weak learner is created.

Understanding ensembles

- * First, input training data is used to build a number of models.
- * The **allocation function** dictates how much of the training data each model receives.
- * Whether they each receive the full training dataset or merely a sample.
- * Whether they each receive every feature or a subset.



Understanding ensembles

- * the ideal ensemble includes a diverse set of models, the allocation function can increase diversity by artificially varying the input data to bias the resulting learners, even if they are the same type.
- * For instance, it might use bootstrap sampling to construct unique training datasets or pass on a different subset of features or examples to each model. On the other hand, if the ensemble already includes a diverse set of algorithms—such as a neural network, a decision tree, and a k-NN classifier—the allocation function might pass the data on to each algorithm relatively unchanged.

Understanding ensembles

- * After the models are constructed, they can be used to generate a set of predictions, and.
- * The **combination function** governs how disagreements among the predictions are reconciled.
- * For example, the ensemble might use a majority vote to determine the final prediction, or it could use a more complex strategy such as weighting each model's votes based on its prior performance.

Understanding ensembles

- * Some ensembles even utilize another model to learn a combination function from various combinations of predictions.
- * For example, suppose that when $M1$ and $M2$ both vote yes, the actual class value is usually no. In this case, the ensemble could learn to ignore the vote of $M1$ and $M2$ when they agree.
- * This process of using the predictions of several models to train a final arbiter model is known as **stacking**.

Understanding ensembles

- * One of the benefits of using ensembles is that they may allow you to spend less time in pursuit of a single best model. Instead, you can train a number of reasonably strong candidates and combine them.
- * Ensembles also offer a number of performance advantages over single models:
- * ***Better generalizability to future problems:*** *As the opinions of several learners are incorporated into a single final prediction, no single bias is able to dominate. This reduces the chance of over fitting to a learning task.*

Understanding ensembles

- * *Improved performance on massive or miniscule datasets:*
 - * *Many models run into memory or complexity limits when an extremely large set of features or examples are used,*
 - * *making it more efficient to train several small models than a single full model..*
 - * *Conversely, ensembles also do well on the smallest datasets because resampling methods such as bootstrapping are inherently a part of many ensemble designs.*
 - * *Perhaps most importantly, it is often possible to train an ensemble in parallel using distributed computing methods.*

Understanding ensembles

- * The ability to synthesize data from distinct domains: Since there is no one-size-fits-all learning algorithm, the ensemble's ability to incorporate evidence from multiple types of learners is increasingly important as complex phenomena rely on data drawn from diverse domains.*
- * A more nuanced understanding of difficult learning tasks: Real-world phenomena are often extremely complex with many interacting intricacies. Models that divide the task into smaller portions are likely to more accurately capture subtle patterns that a single global model might miss.*

Bagging

- * One of the first ensemble methods used a technique called **bootstrap aggregating** or **bagging** for short.
- * Bagging generates a number of training datasets by bootstrap sampling the original training data.
- * These datasets are then used to generate a set of models using a single learning algorithm. The models' predictions are combined using voting (for classification) or averaging (for numeric prediction).

Bagging

- * Bagging is a relatively simple ensemble, it can perform quite well with relatively **unstable** learners,
 - * that is, those generating models that tend to change substantially when the input data changes only slightly.
- * Bagging is often used with decision trees, which have the tendency to vary dramatically given minor changes in the input data.

Boosting

- * Another common ensemble-based method is called **boosting** because it boosts the performance of weak learners to attain the performance of stronger learners.
- * This method is based largely on the work of Robert Schapire and Yoav Freund, who have published extensively on the topic.
- * Similar to bagging, boosting uses ensembles of models trained on resampled data and a vote to determine the final prediction.
- * There are two key distinctions.
 - * First, the resampled datasets in boosting are constructed specifically to generate complementary learners.
 - * Second, rather than giving each learner an equal vote, boosting gives each learner's vote a weight based on its past performance.
- * Models that perform better have greater influence over the ensemble's final prediction.

Boosting

- * Boosting will result in performance that is often quite better and certainly no worse than the best of the models in the ensemble.
- * Since the models in the ensemble are built to be complementary, it is possible to increase ensemble performance to an arbitrary threshold simply by adding additional classifiers to the group, assuming that each classifier performs better.
- * Boosting is thought to be one of the most significant discoveries in machine learning.

Boosting

- * A boosting algorithm called **AdaBoost** or **adaptive boosting** was proposed by Freund and Schapire in 1997.
- * Though boosting principles can be applied to nearly any type of model, the principles are most commonly used with decision trees.
- * The algorithm is based on the idea of generating weak learners that iteratively learn a larger portion of the difficult-to-classify examples by paying more attention (that is, giving more weight) to frequently misclassified examples.
- * Beginning from an unweighted dataset, the first classifier attempts to model the outcome.

Boosting

- * Examples that the classifier predicted correctly will be less likely to appear in the training dataset for the following classifier, and conversely, the difficult-to-classify examples will appear more frequently.
- * As additional rounds of weak learners are added, they are trained on data with successively more difficult examples.
- * The process continues until the desired overall error rate is reached or performance no longer improves.
- * At that point, each classifier's vote is weighted according to its accuracy on the training data on which it was built.

Random forests

- * Another ensemble-based method called **random forests** (or **decision tree forests**) focuses only on ensembles of decision trees.
- * This method was championed by Leo Breiman and Adele Cutler, and combines the base principles of bagging with random feature selection to add additional diversity to the decision tree models.
- * After the ensemble of trees (the forest) is generated, the model uses a vote to combine the trees' predictions.

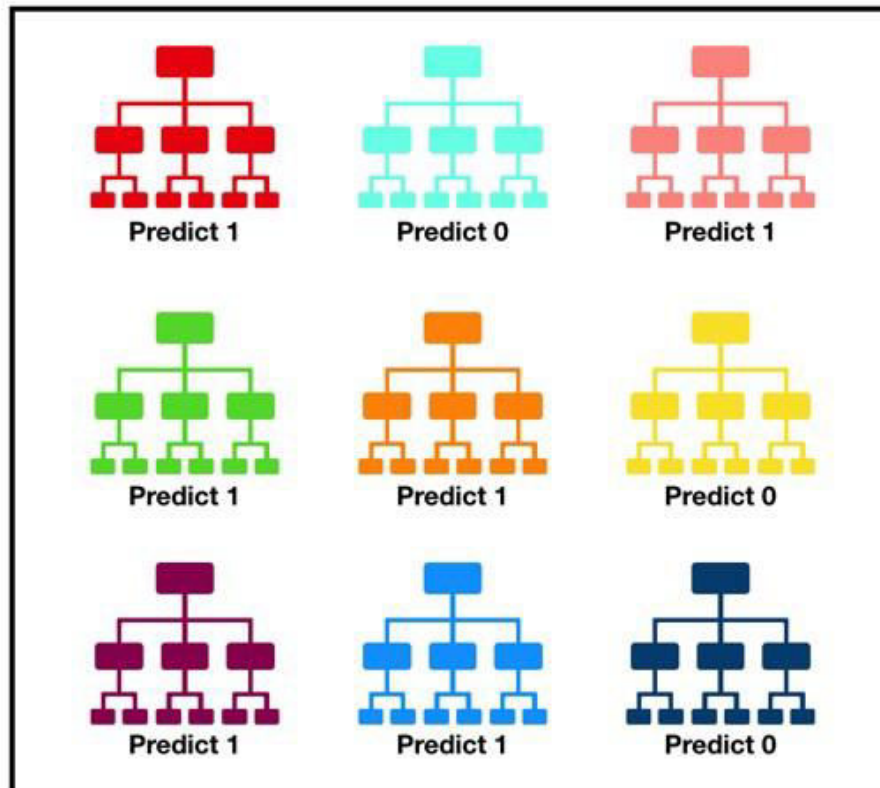
Random forests

- * Random forests combine versatility and power into a single machine learning approach.
- * As the ensemble uses only a small, random portion of the full feature set, random forests can handle extremely large datasets, where the so-called "curse of dimensionality" might cause other models to fail.
- * At the same time, its error rates for most learning tasks are on par with nearly any other method.


Random forests

- * Due to their power, versatility, and ease of use, random forests are quickly becoming one of the most popular machine learning methods.

Strengths	Weaknesses
<ul style="list-style-type: none">• An all-purpose model that performs well on most problems• Can handle noisy or missing data as well as categorical or continuous features• Selects only the most important features• Can be used on data with an extremely large number of features or examples	<ul style="list-style-type: none">• Unlike a decision tree, the model is not easily interpretable• May require some work to tune the model to the data



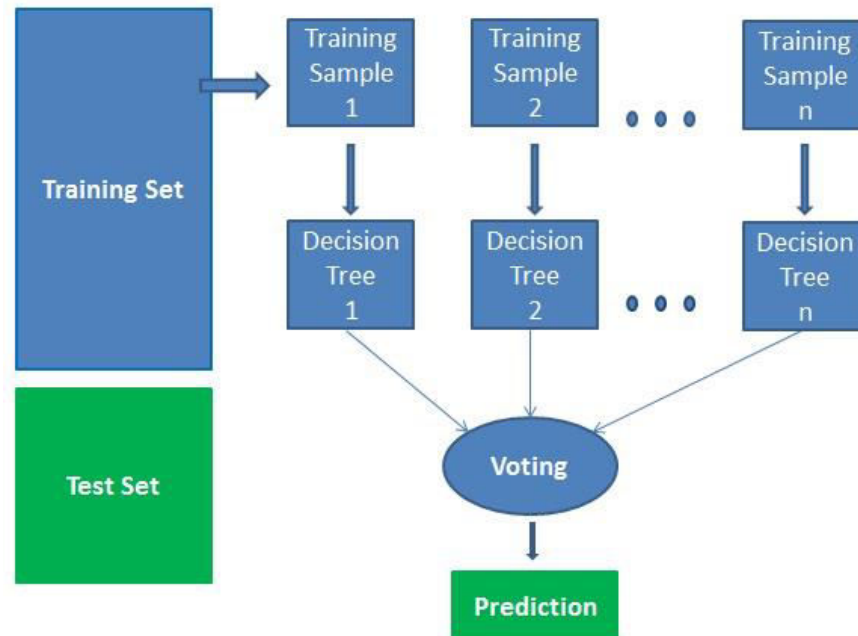
Tally: Six 1s and Three 0s
Prediction: 1

- 
- * Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.
 - * Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.
 - * In a classification problem, each tree votes and the most popular class is chosen as the final result. In the case of regression, the average of all the tree outputs is considered as the final result. It is simpler and more powerful compared to the other non-linear classification algorithms.

How does the algorithm work?

- * Select random samples from a given dataset.
- * Construct a decision tree for each sample and get a prediction result from each decision tree.
- * Perform a vote for each predicted result.
- * Select the prediction result with the most votes as the final prediction.

*





Advantages:

- * Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- * It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- * The algorithm can be used in both classification and regression problems.
- * Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
- * You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

Disadvantages:

- * Random forests is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
- * The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.