Advanced ML Classification

Unit 5

Syllabus

- Ensemble Classifiers: Introduction to Ensemble Methods, Bagging, Boosting, Random forests,
- Improving classification accuracy of Class-Imbalanced Data- Metrics for Evaluating Classifier Performance, Holdout Method and Random Subsampling, Cross-Validation, Bootstrap, Model Selection Using Statistical Tests of Significance, Comparing Classifiers Based on Cost–Benefit and ROC Curves.
- Self-learning Topics: Introduction to ML (Revision)

Introduction to Ensemble Methods

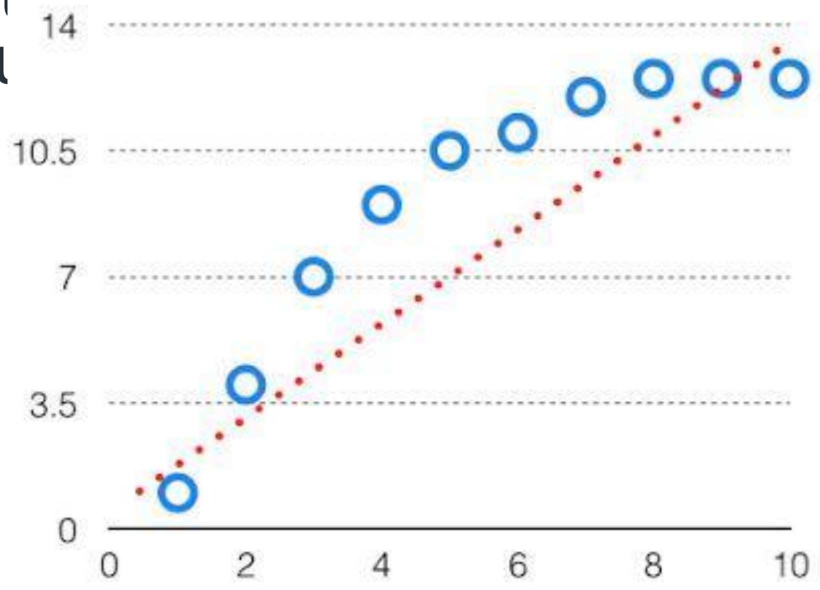
- “Unity is strength”. This old saying expresses pretty well the underlying idea that rules the very powerful “**ensemble method**s” in machine learning.
- Some **well known ensemble emthods** such as boostrapping, bagging, random forest, boosting, stacking and many others that are the basis of ensemble learning.
- **A system of beliefs, ideas, values, and habits** and the three most common paradigms are **positivism, constructivism or interpretivism and pragmatism**.
- Ensemble learning is a **machine learning paradigm** where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results.
- The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.

Introduction to Ensemble Methods

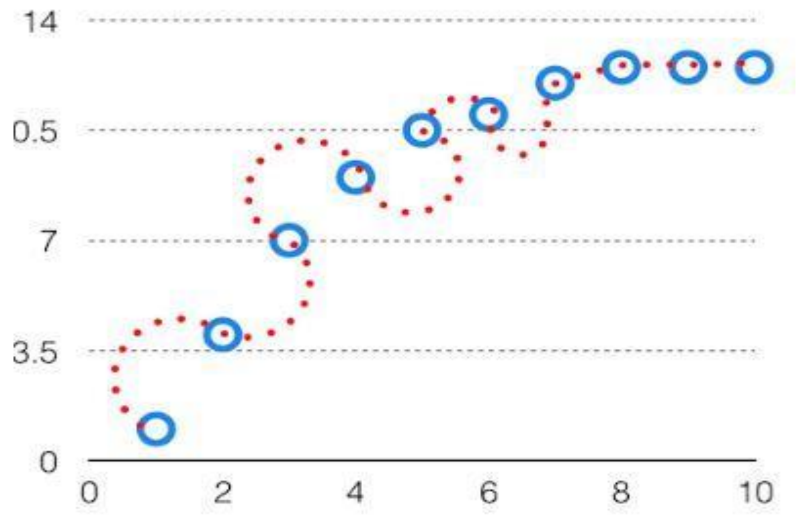
- In machine learning, if facing a classification or a regression problem, the choice of the model is extremely important to have any chance to obtain good results.
- This choice can depend on many variables of the problem: quantity of data, dimensionality of the space, distribution hypothesis…
- A low bias and a low variance, although they most often vary in opposite directions, are the two most fundamental features expected for a model.

- **Bias**
  The bias is known as the **difference between the prediction of the values by the ML model and the correct value**. Being high in biasing gives a large error in training as well as testing data. Its recommended that an algorithm should always be low biased to avoid the problem of underfitting.

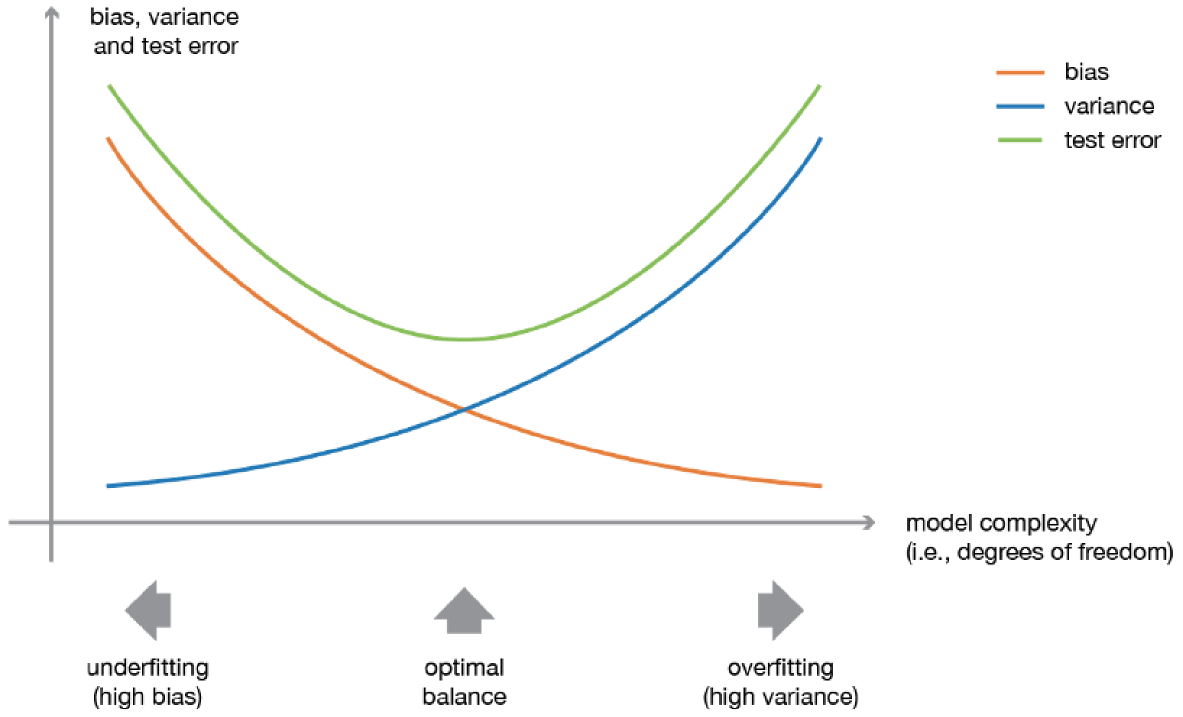
By high bias, the data predicted is in a straight line format, thus not fitting accurately in the data in the data set. Such fitting is known as **Underfitting of Data**. This happens when the hypothesis is too simple or linear in nature. Refer to the graph given below for an example of such a situ



- **Variance**
  The variability of model prediction for a given data point which tells us **spread of our data** is called the variance of the model. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before. As a result, such **models perform very well on training data but has high error rates on test data.**
  When a model is high on variance, it is then said to as **Overfitting of Data**. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high.
  While training a data model variance should be kept low.
- The high variance data looks like follows.



- **Bias Variance Tradeoff:** If the algorithm is too simple (hypothesis with linear eq.) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree eq.) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well.



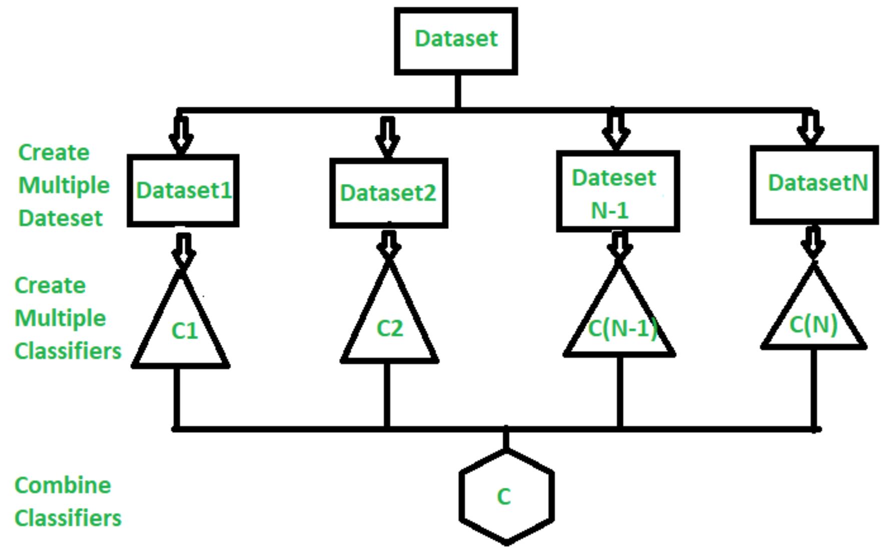
- One important point is that our choice of weak learners should be **coherent with the way we aggregate these models**. If we choose base models with low bias but high variance, it should be with an aggregating method that tends to reduce variance whereas if we choose base models with low variance but high bias, it should be with an aggregating method that tends to reduce bias.

1. This brings us to the question of how to combine these models. We can mention three major kinds of meta-algorithms that aims at combining weak learners: bagging, boosting, Stacking, Blending

Ensemble learning

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

Advantage : Improvement in predictive accuracy.
Disadvantage : It is difficult to understand an ensemble of classifiers.



Why Do Ensemble work

- **Statistical Problem –**
  The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!
- **Computational Problem –**
  The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis.
- **Representational Problem –**
  The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es

Main Challenge for Developing Ensemble Models?

- Is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassify different training examples, even if the base classifier accuracy is low.
- ***Methods for Independently Constructing Ensembles –***
  - *Majority Vote*
  - *Bagging and Random Forest*
  - *Randomness Injection*
  - *Feature-Selection Ensembles*
  - *Error-Correcting Output Coding*
- ***Methods for Coordinated Construction of Ensembles –***
  - *Boosting*
  - *Stacking*
- ***Reliable Classification:*** *Meta-Classifier Approach*
  Co-Training and Self-Training

Combine weak learners

- In ensemble learning theory, we call **weak learners** (or **base models**) models that can be used as building blocks for designing more complex models by combining several of them.
- In order to set up an ensemble learning method
- we first need to select our base models to be aggregated.
- **Most of the time** (including in the well known bagging and boosting methods) a single base learning algorithm is used so that we have **homogeneous weak learners** that are trained in different ways.
- The ensemble model we obtain is then said to be “homogeneous”. However, there also exist some methods that use different type of base learning algorithms: some heterogeneous weak learners are then combined into an “**heterogeneous ensembles model”.**

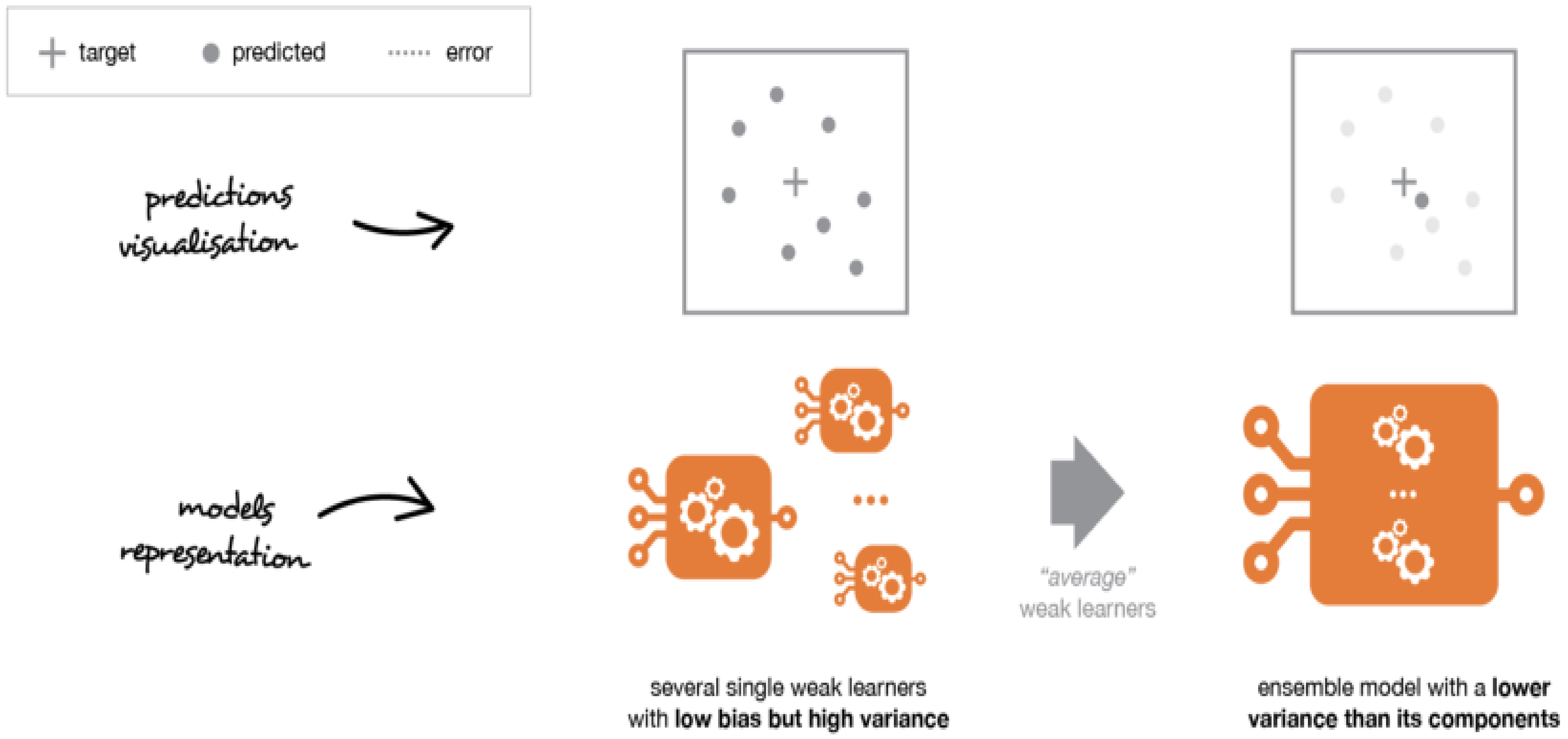
Introduction to Ensemble Methods

- **There are many ways to reduce the Bias Problem**

1. **Improve Data Collection and Preprocessing Techniques:** Collect more data and give much more time to preprocessing by detecting outliers and segment the data according to balanced class.
2. **Up Sampling and Down Sampling:** If you have less data, then this technique is quite useful. Up(Over) Sampling is increasing the number of classes which is less in number by considering the data points closer to that of the original class. Down Sampling is the inverse of oversampling, i.e. reducing the number of classes having a higher number of data points.
3. **Use Specific Algorithm Properties:** This is helpful for some Machine Learning Algorithms where you can give weights to a particular class, which may decrease bias. For example, you have 70% -A Class and 30% -B Class, where you can give more weight on A class because Algorithm may tend to be more biased towards Class B.

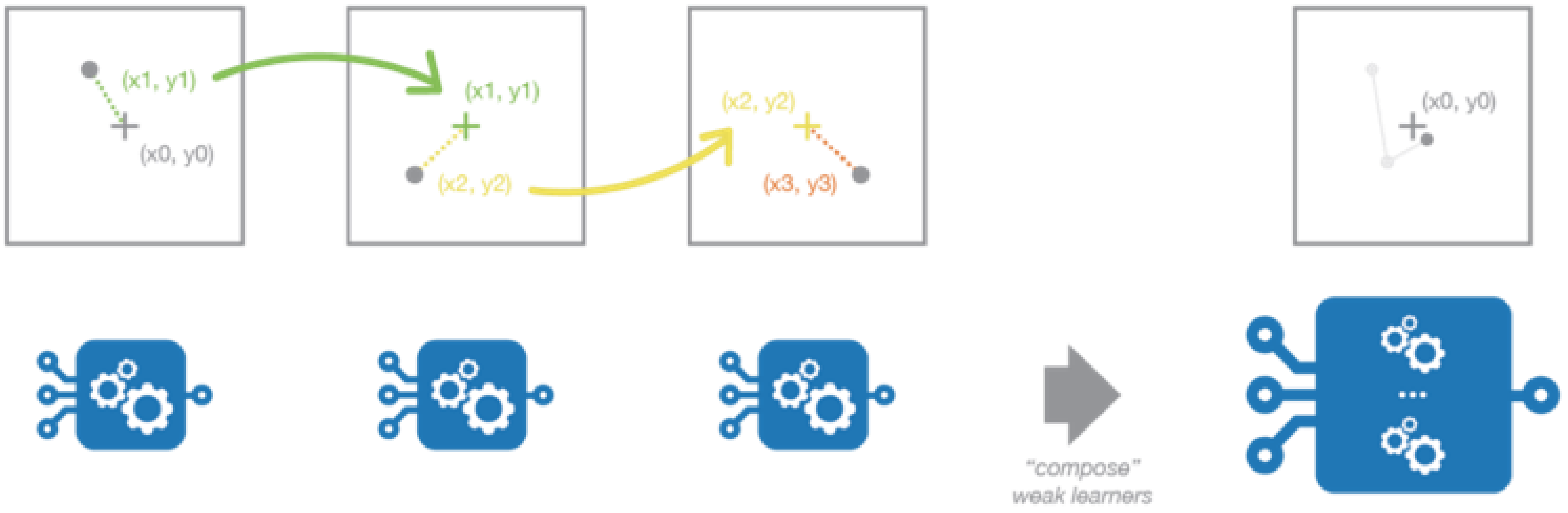
Ensemble Models

- **Bagging**, that often considers homogeneous weak learners, learns them independently from each other in **parallel** and combines them following some kind of **deterministic averaging process**
- **Boosting**, that often considers **homogeneous weak learners**, learns them **sequentially i**n a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy
- **Stacking**, that often considers **heterogeneous** weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions


+ target
● predicted
······ error
predictions visualisation
models representation
...
"average" weak learners
...
several single weak learners with **low bias but high variance**
ensemble model with a **lower variance than its components**

LOW BIAS HIGH VARIANCE WEAK LEARNERS

LOW VARIANCE HIGH BIAS WEAK LEARNERS

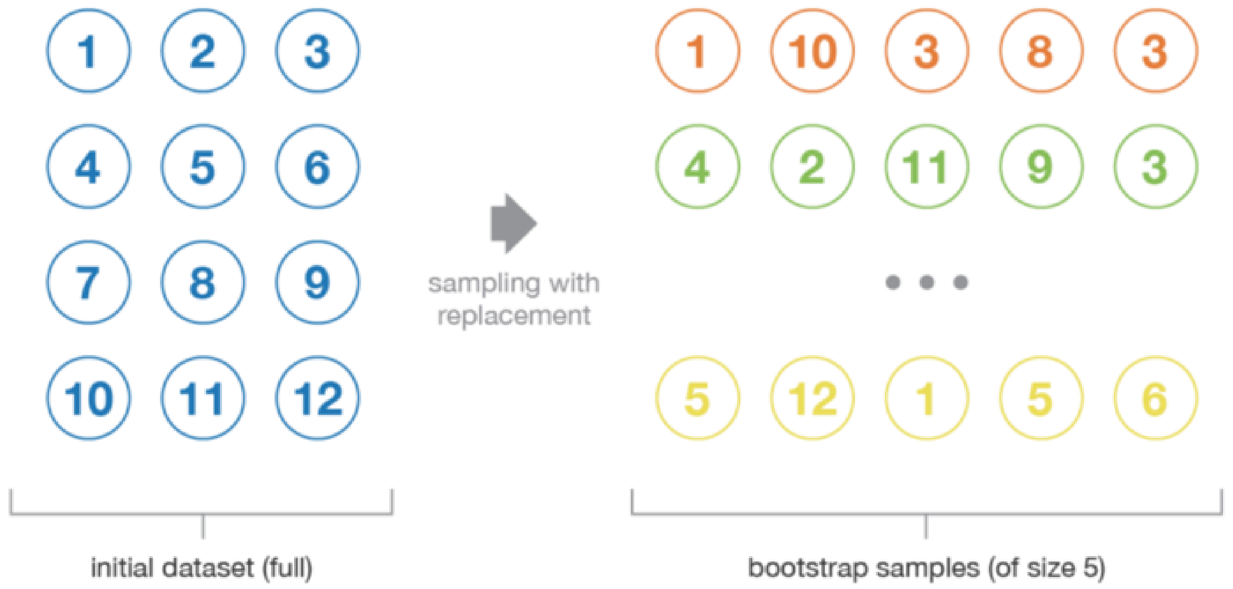


several single weak learners with **high bias but low variance**: each model target the error of the previous one

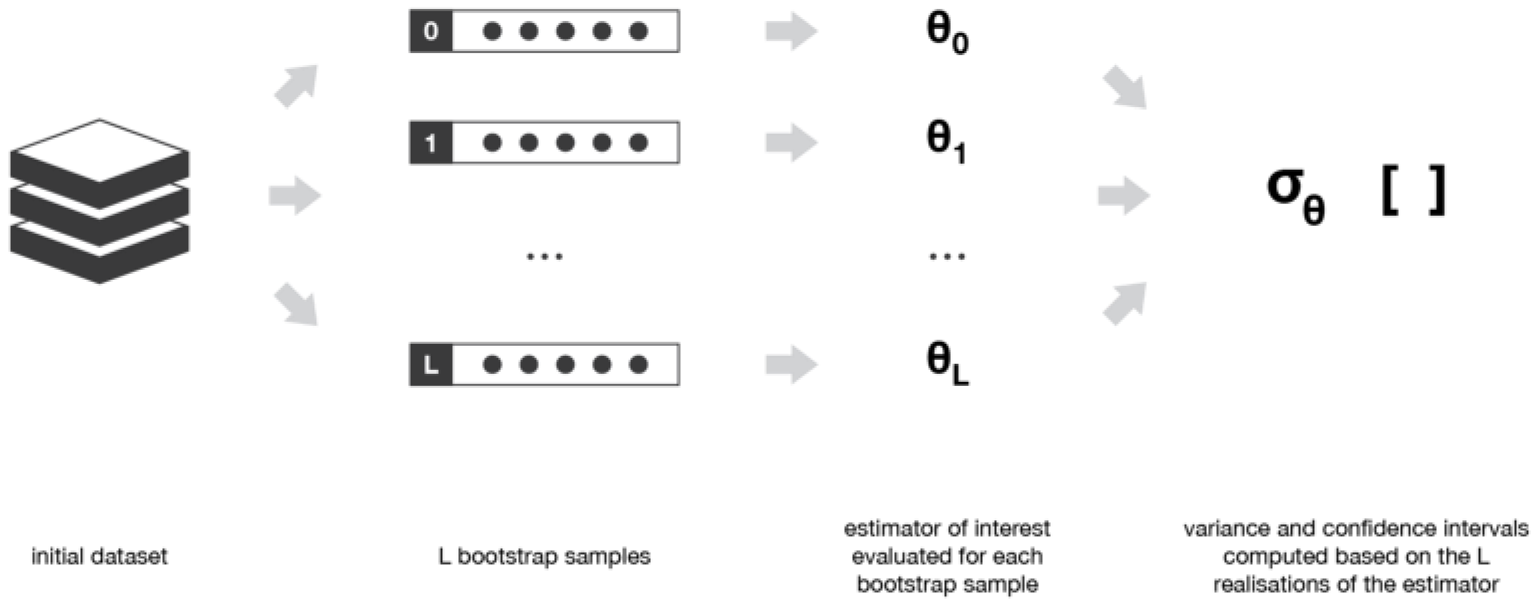
ensemble model with a **lower bias than its components**

Bagging" (standing for "bootstrap aggregating")

- This statistical technique consists in generating samples of size B (called bootstrap samples) from an initial dataset of size N by randomly drawing with replacement B observations.
- First, the size N of the initial dataset should be large enough to capture most of the complexity of the underlying distribution so that sampling from the dataset is a good approximation of sampling from the real distribution (**representativity**)



- Second, the size N of the dataset should be large enough compared to the size B of the bootstrap samples so that samples are not too much correlated (**independence**).
- Bootstrap samples are often used, for example, to evaluate variance or confidence intervals of a statistical estimators.


0
1
L
...
θ_0
θ_1
θ_L
σ_θ [ ]
initial dataset
L bootstrap samples
estimator of interest evaluated for each bootstrap sample
variance and confidence intervals computed based on the L realisations of the estimator

Bagging

- The idea of bagging is then simple: we want to fit several independent models and “average” their predictions in order to obtain a model with a lower variance.
- The **random forest** approach is a bagging method where **deep trees**, fitted on bootstrap samples, are combined to produce an output with lower variance.

Boosting

- Being **mainly focused at reducing bias**, the base models that are often considered for boosting are models with low variance but high bias.
- For example, if we want to use trees as our base models, we will choose most of the time shallow **decision trees** with only a few depths
- These two meta-algorithms differ on how they create and aggregate the weak learners during the sequential process. Adaptive boosting updates the weights attached to each of the training dataset observations whereas gradient boosting updates the value of these observations.
- In **adaptative boosting** (often called "adaboost"), we try to define our ensemble model as a weighted sum of L weak learners

$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.) \qquad \text{where } c_l\text{'s are coefficients and } w_l\text{'s are weak learners}$$

Boosting

- Finding the best ensemble model with this form is a **difficult optimisation problem**.
- First, it **updates the observations weights** in the dataset and train a new weak learner with a special focus given to the observations misclassified by the current ensemble model. Second, it **adds the weak learner to the weighted sum** according to an update coefficient that expresse the performances of this weak model: the better a weak learner performs, the more it contributes to the strong learner.

- Gradient Boosting : we make use of an **iterative optimisation process** that is much more tractable, even if it can lead to a sub-optimal solution.

$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.) \qquad \text{where } c_l\text{'s are coefficients and } w_l\text{'s are weak learners}$$

Stacking

- For example, for a classification problem, we can choose as weak learners a KNN classifier, a logistic regression and a SVM, and decide to learn a neural network as meta-model. Then, the neural network will take as inputs the outputs of our three weak learners and will learn to return final predictions based on it.
- **Then we have to follow the steps thereafter**:
  - split the training data in two folds
  - choose L weak learners and fit them to data of the first fold
  - for each of the L weak learners, make predictions for observations in the second fold
  - Fit the meta-model on the second fold, using predictions made by the weak learners as inputs

Multi-levels Stacking

- A possible extension of stacking is multi-level stacking. It consists in doing **stacking with multiple layers**. As an example, let’s consider a 3-levels stacking. In the first level (layer), we fit the L weak learners that have been chosen. Then, in the second level, instead of fitting a single meta-model on the weak models predictions (as it was described in the previous subsection) we fit M such meta-models. Finally, in the third level we fit a last meta-model that takes as inputs the predictions returned by the M meta-models of the previous level.

Improving classification accuracy of Class-Imbalanced Data.

Imblearn

Ways to reduce the Bias Problem

1. **Improve Data Collection and Preprocessing Techniques:** Collect more data and give much more time to preprocessing by detecting outliers and segment the data according to balanced class.
2. **Up Sampling and Down Sampling:** If you have less data, then this technique is quite useful. Up(Over) Sampling is increasing the number of classes which is less in number by considering the data points closer to that of the original class. Down Sampling is the inverse of oversampling, i.e. reducing the number of classes having a higher number of data points.
3. **Use Specific Algorithm Properties:** This is helpful for some Machine Learning Algorithms where you can give weights to a particular class, which may decrease bias. For example, you have 70% -A Class and 30% -B Class, where you can give more weight on A class because Algorithm may tend to be more biased towards Class B.

Weights in Scikit Learn Library

- *model=RandomForestClassifier(class_weights={1:0.71,2:1.67})*
- In the above case of the Random Forest Classifier, Class 1 will be given weights of 0.71 and Class 2, 1.67. So multiplying them will result in 50:50. This can be also used to make algorithms focused more on a particular class.

Imbalanced-learn: An Imbalance Dataset Library

- This library is used to balance the dataset in python. It consists of many resampling techniques, Under Sampling, Oversampling, and many more.
- Here's the documentation of Imblearn.
- For PyPI:

```
pip install -U imbalanced-learn
```

- For Anaconda:

```
conda install -c conda-forge imbalanced-learn
```

Under Sampling in Imbalanced-Learn Library

- Under Sampling in Imbalance, Learn Library is a group of techniques that **mainly focuses on reducing the data points** of the majority class in an imbalanced dataset.

1. Near Miss Under Sampling
2. Condensed Nearest Neighbours

Over Sampling in Imbalanced-Learn Library

- Over Sampling in Imbalance Learn Library is a group of techniques that **mainly focuses on increasing the data points** of the minority class in an imbalanced dataset.

1. RandomOverSampler
2. SMOTE- Synthetic Minority Over-sampling Technique

SMOTE

- **SMOTE is an Over Sampling technique**, used to increase the data points for minority classes by augmenting data with similar data points as of the minority class.
- Here, we are importing SMOTE from ‘imblearn.over_sampling’ and resampling the y[target] variable according to X, input variables.

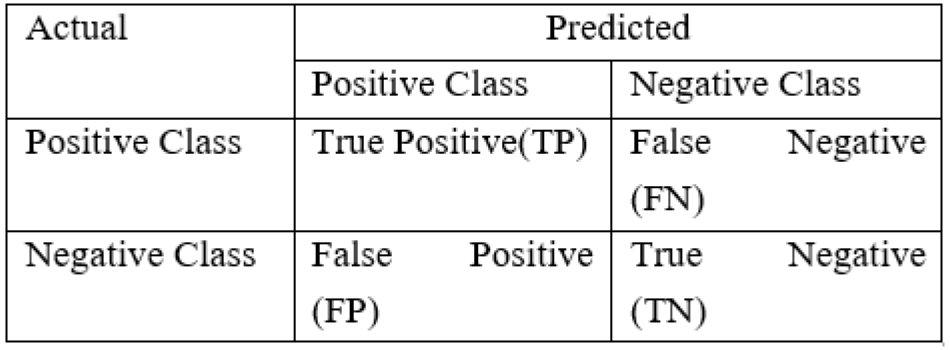
```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
X=data.drop(['Attrition','EmployeeCount','StandardHours','
Over18'],axis=1)
y=data['Attrition']
X_os,y_os=sm.fit_resample(X,y)
```

Example

- This problem is predominant in scenarios where anomaly detection is crucial like **electricity pilferage, fraudulent transactions in banks, identification of rare diseases, Natural Disaster like Earthquakes,** etc. In this situation, the predictive model developed using conventional machine learning algorithms could be biased and inaccurate.
- Example of imbalanced data

  Let’s understand this with the help of an example.
- **Ex:** In an utilities fraud detection data set you have the following data:
  - Total Observations = 1000
  - Fraudulent Observations = 20
  - Non Fraudulent Observations = 980
  - Event Rate= 2 %

- Standard classifier algorithms like **Decision Tree and Logistic Regression** have a bias towards classes which have number of instances.
- The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.ey tend to only predict the majority class data
- Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.
- **Accuracy of a model = (TP+TN) / (TP+FN+FP+TN)**



Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Approach to handling Imbalanced Data

- Data Level approach: Resampling Techniques
- Random Under-Sampling
- Random Over-Sampling
- Cluster-Based Over Sampling
- Informed Over Sampling: Synthetic Minority Over-sampling Technique for imbalanced data
- Modified synthetic minority oversampling technique (MSMOTE) for imbalanced data

Data Level approach: Resampling Techniques

- entails strategies such as improving classification algorithms or balancing classes in the training data (data preprocessing) before providing the data as input
- The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class

Random Under-Sampling

- Total Observations = 1000 , Fraudulent Observations =20
- Non Fraudulent Observations = 980 ,Event Rate= 2 %
- In this case we are taking 10 % samples without replacement from Non Fraud instances. And combining them with Fraud instances.
- Non Fraudulent Observations after random under sampling = 10 % of 980 =98
- Total Observations after combining them with Fraudulent observations = 20+98=118
- Event Rate for the new dataset after under sampling = 20/118 = 17%

Random Over-Sampling

- Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.
- Total Observations = 1000, Fraudulent Observations =20
- Non Fraudulent Observations = 980 ,Event Rate= 2 %
- In this case we are replicating 20 fraud observations 20 times.
- Non Fraudulent Observations =980
- Fraudulent Observations after replicating the minority class observations= 400
- Total Observations in the new data set after oversampling=1380
- Event Rate for the new data set after under sampling= 400/1380 = 29 %

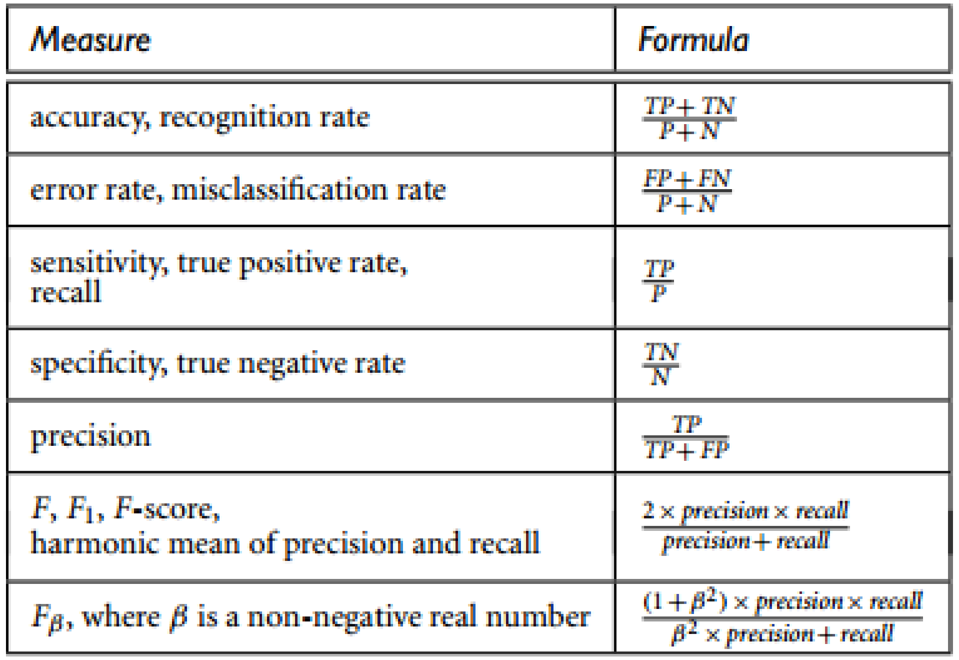
Cluster-Based Over Sampling

- This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.
- Total Observations = 1000, Fraudulent   Observations =20
- Non Fraudulent Observations = 980, Event Rate= 2 %
- **Majority Class Clusters**
    - Cluster 1: 150 Observations
    - Cluster 2: 120 Observations
    - Cluster 3: 230 observations
    - Cluster 4: 200 observations
    - Cluster 5: 150 observations
    - Cluster 6: 130 observations
- **Minority Class Clusters**
    - Cluster 1: 8 Observations
    - Cluster 2: 12 Observations
- **After oversampling of each cluster, all clusters of the same class contain the same number of observations.**
- **Majority Class Clusters**
    - Cluster 1: 170 Observations
    - Cluster 2: 170 Observations
    - Cluster 3: 170 observations
    - Cluster 4: 170   observations
    - Cluster 5: 170   observations
    - Cluster 6: 170   observations
- **Minority   Class Clusters**
    - Cluster 1: 250 Observations
    - Cluster 2: 250 Observations
- Event Rate post cluster based oversampling sampling = 500/ (1020+500) = 33 %

Informed Over Sampling: Synthetic Minority Over-sampling Technique for imbalanced data

- Total Observations = 1000
- Fraudulent Observations = 20
- Non Fraudulent Observations = 980
- Event Rate = 2 %
- A sample of 15 instances is taken from the minority class and similar synthetic instances are generated 20 times
- Post generation of synthetic instances, the following data set is created
- Minority Class (Fraudulent Observations) = 300
- Majority Class (Non-Fraudulent Observations) = 980
- Event rate= 300/1280 = 23.4 %

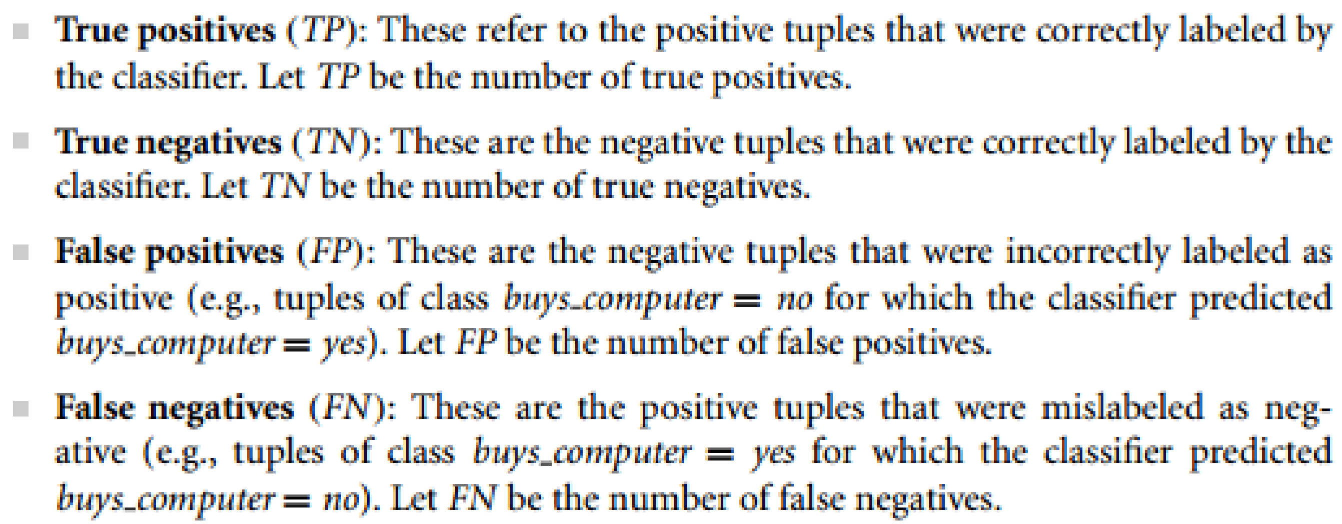
Model Evaluation and Selection



Measure	*Formula*
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$
F, F_1, F-score, harmonic mean of precision and recall	$\frac{2 \times precision \times recall}{precision + recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$

Evaluation measures. Note that some measures are known by more than one name. TP, TN, FP, P, N refer to the number of true positive, true negative, false positive, positive, and negative samples, respectively (see text).

Model Evaluation and Selection



- **True positives** (*TP*): These refer to the positive tuples that were correctly labeled by the classifier. Let *TP* be the number of true positives.
- **True negatives** (*TN*): These are the negative tuples that were correctly labeled by the classifier. Let *TN* be the number of true negatives.
- **False positives** (*FP*): These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class *buys_computer* = *no* for which the classifier predicted *buys_computer* = *yes*). Let *FP* be the number of false positives.
- **False negatives** (*FN*): These are the positive tuples that were mislabeled as negative (e.g., tuples of class *buys_computer* = *yes* for which the classifier predicted *buys_computer* = *no*). Let *FN* be the number of false negatives.

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem:**
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

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

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

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

- Perfect score is 1.0
- Inverse relationship between precision & recall

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F* measure (*F*$_1$ or *F*-score)**: harmonic mean of precision and recall,
- ***F*$_ß$**: weighted m

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

  - assigns ß ti

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

cross-validation procedure

- The k-fold cross-validation procedure involves splitting the training dataset into *k* folds. The first *k-1* folds are used to train a model, and the holdout *k*th fold is used as the test set. This process is repeated and each of the folds is given an opportunity to be used as the holdout test set. A total of *k* models are fit and evaluated, and the performance of the model is calculated as the mean of these runs.

Holdout Method

- The holdout method reserves a certain amount of data for testing and uses the remainder for training – so they are disjoint!

• Usually, one third (N/3) of data is used for testing, and the rest (N -N/3) = (2N/3) for training

• The choice of records for train and test data is essential We usually perform as repeated holdout Train-and-test; repeat

- Holdout can be made more reliable by repeating the process with different sub-samples - subsets of data selected for training and testing Repeated Holdout 1. In each iteration, a certain portion of data is randomly selected for training, the rest of data is used for testing 2. The error rates or predictive accuracy on diferent iterations are averaged to yield an overall error rate, or overall predictive accuracy
- • Repeated holdout still is not optimal: the different test sets overlap

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?

■ Use validation test set of class-labeled tuples instead of training set when assessing accuracy

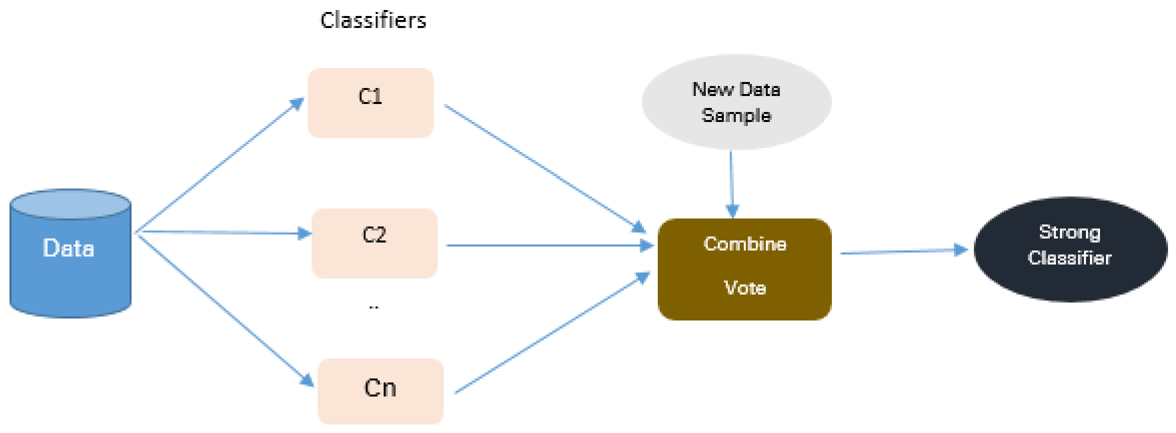
■ Methods for estimating a classifier’s accuracy:

  - ■ Holdout method, random subsampling
  - ■ Cross-validation
  - ■ Bootstrap

■ Comparing classifiers:

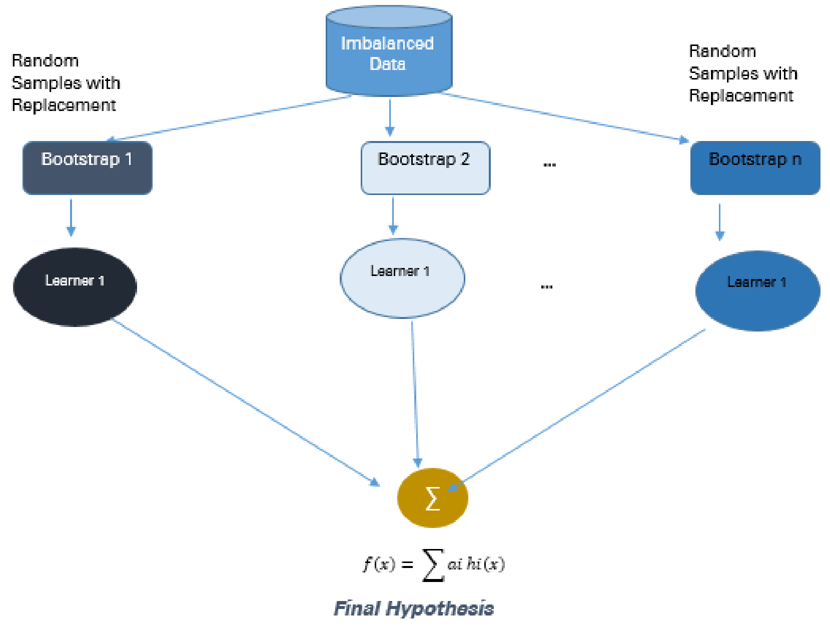
  - ■ Confidence intervals
  - ■ Cost-benefit analysis and ROC Curves

Approach to Ensemble based Methodologies



Bagging Based techniques-Imblearn

- Bagging is an abbreviation of Bootstrap Aggregating. The conventional bagging algorithm involves generating ‘n’ different bootstrap training samples with replacement. And training the algorithm on each bootstrapped algorithm separately and then aggregating the predictions at the end.
- Bagging is used for reducing Overfitting in order to create strong learners for generating accurate predictions

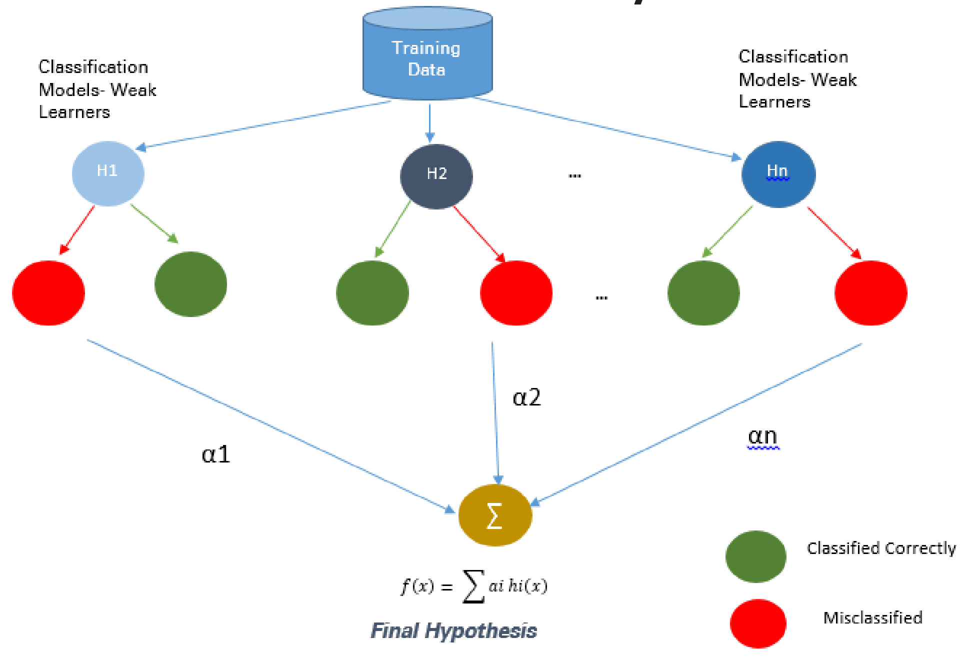


Bagging

- **Advantages**
  - Improves stability & accuracy of machine learning algorithms
  - Reduces variance
  - Overcomes overfitting
  - Improved misclassification rate of the bagged classifier
  - In noisy data environments bagging outperforms boosting
-
- **Disadvantages**
  - Bagging works only if the base classifiers are not bad to begin with. Bagging bad classifiers can further degrade performance

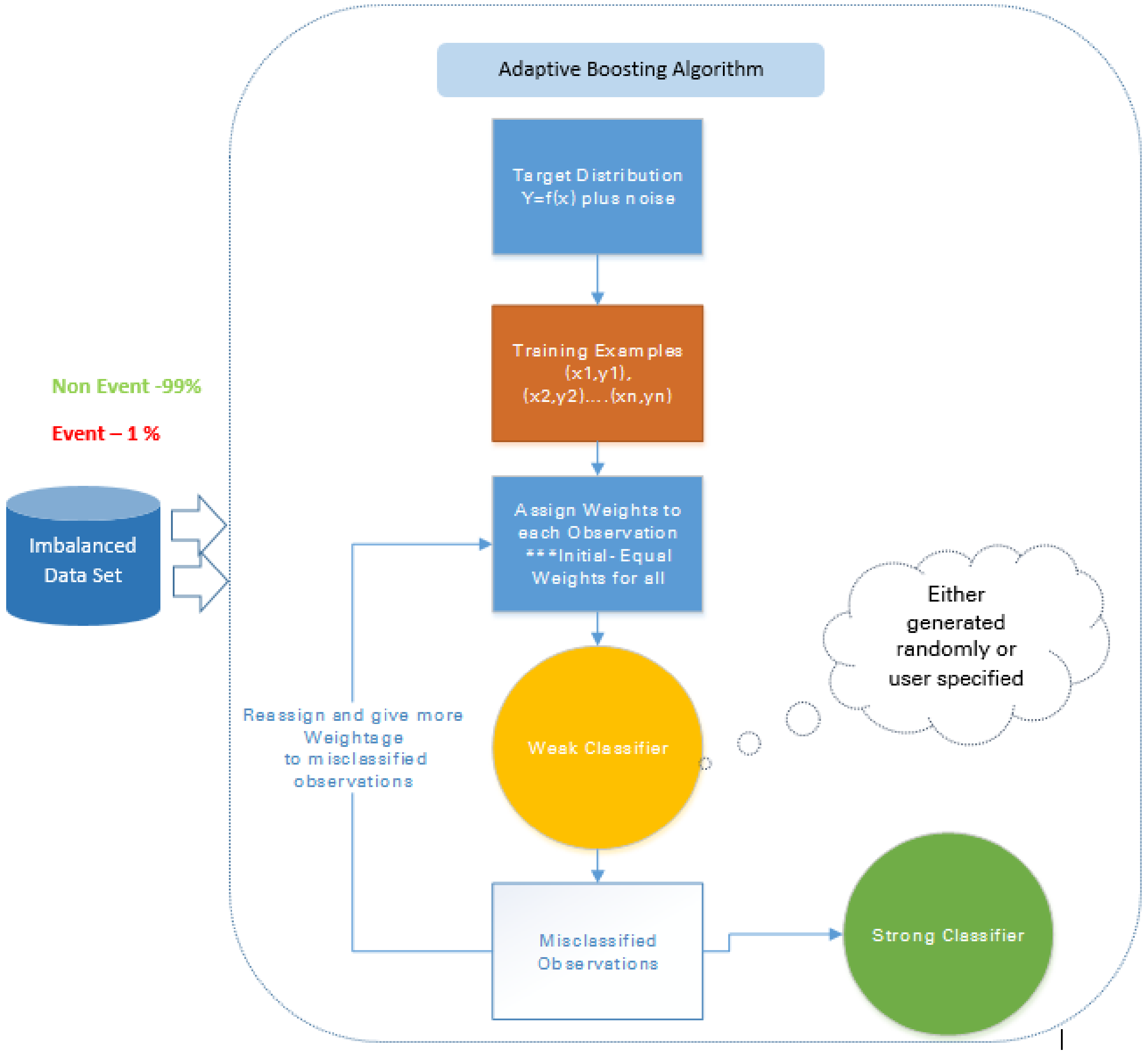
Boosting-Based techniques for Imblearn

- Boosting is an ensemble technique to combine weak learners to create a strong learner that can make accurate predictions.
- The base learners / Classifiers are weak learners i.e. the prediction accuracy is only slightly better than average.
- In the next iteration, the new classifier focuses on or places more weight to those cases which were incorrectly classified in the last round.



Adaptive Boosting- Ada Boost techniques

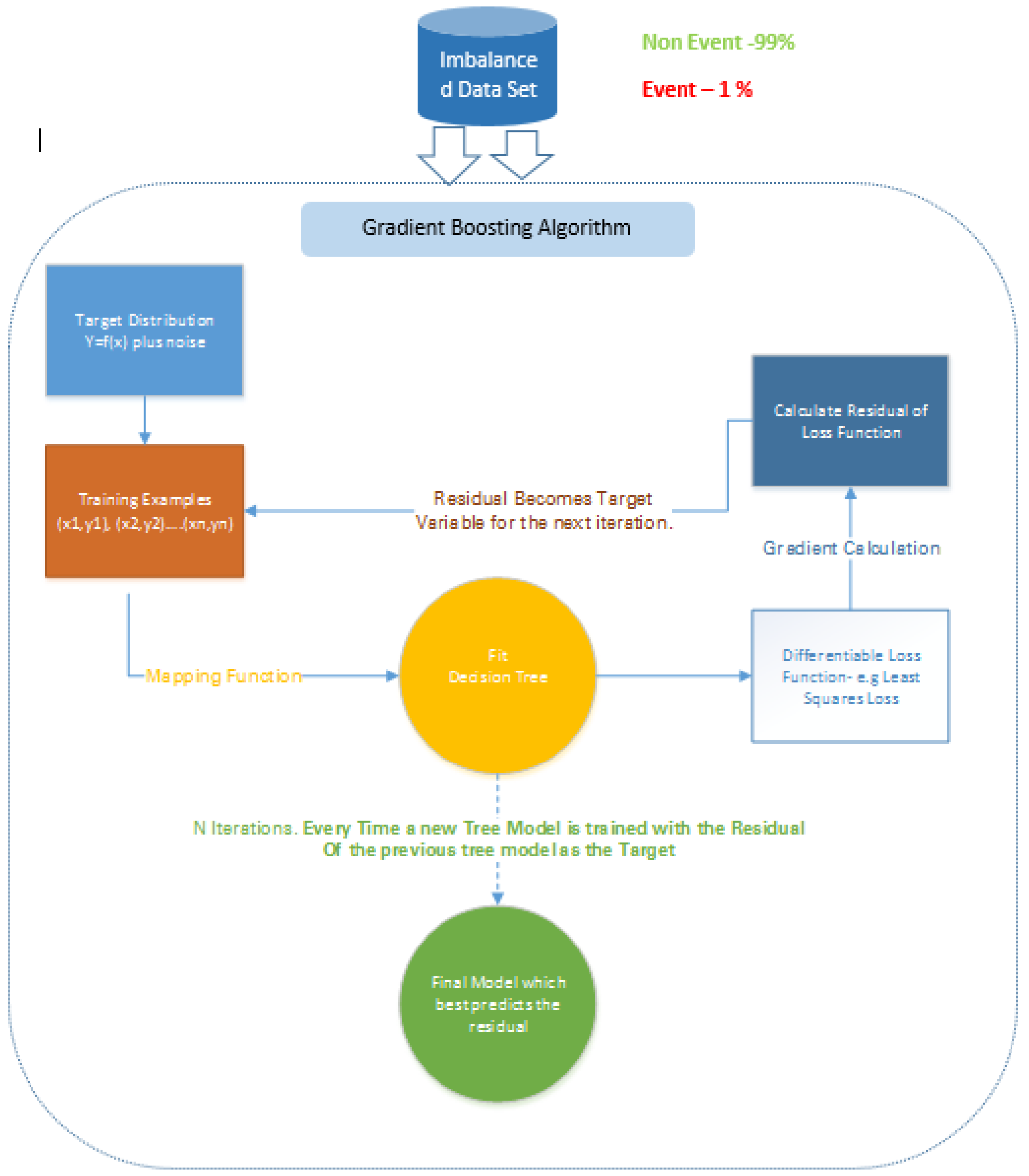
- Each classifier is serially trained with the goal of correctly classifying examples in every round that were incorrectly classified in the previous round.
- For a learned classifier to make strong predictions it should follow the following three conditions:
- The rules should be simple
- Classifier should have been trained on sufficient number of training examples
- The Classifier should have low training error for the training instances


Adaptive Boosting Algorithm
Non Event -99%
Event – 1 %
Imbalanced Data Set
Target Distribution Y=f(x) plus noise
Training Examples (x1,y1), (x2,y2)....(xn,yn)
Assign Weights to each Observation ***Initial- Equal Weights for all
Either generated randomly or user specified
Reassign and give more Weightage to misclassified observations
Weak Classifier
Misclassified Observations
Strong Classifier

- For example in a data set containing 1000 observations out of which 20 are labelled fraudulent. Equal weights W1 are assigned to all observations and the base classifier accurately classifies 400 observations.
- Weight of each of the 600 misclassified observations is increased to w2 and weight of each of the correctly classified observations is reduced to w3.
- In each iteration, these updated weighted observations are fed to the weak classifier to improve its performance. This process continues till the misclassification rate significantly decreases thereby resulting in a strong classifier.
-
- **Advantages**
  - Very Simple to implement
  - Good generalization- suited for any kind of classification problem ü Not prone to overfitting
-
- **Disadvantages**
  - Sensitive to noisy data and outliers

Gradient Tree Boosting techniques for imbalanced data

- In Gradient Boosting many models are trained sequentially. It is a numerical optimization algorithm where each model minimizes the loss function, **y = ax+b+e**, using the Gradient Descent Method.
- Decision Trees are used as weak learners in Gradient Boosting.
- While both Adaboost and Gradient Boosting work on weak learners / classifiers. And try to boost them into a strong learner
- On the other hand, Gradient Boosting builds the first learner on the training dataset to predict the samples, calculates the loss (Difference between real value and output of the first learner). And use this loss to build an improved learner in the second stage.



Imbalanced Data Set
Non Event -99%
Event – 1 %
Gradient Boosting Algorithm
Target Distribution Y=f(x) plus noise
Training Examples (x1,y1), (x2,y2)…..(xn,yn)
Residual Becomes Target Variable for the next iteration.
Calculate Residual of Loss Function
Gradient Calculation
Mapping Function
Fit Decision Tree
Differentiable Loss Function- e.g Least Squares Loss
N Iterations. Every Time a new Tree Model is trained with the Residual Of the previous tree model as the Target
Final Model which best predicts the residual

Disadvantage

- Gradient Boosted trees are harder to fit than random forests
- Gradient Boosting Algorithms generally have 3 parameters which can be fine-tuned, Shrinkage parameter, depth of the tree, the number of trees. Proper training of each of these parameters is needed for a good fit. If parameters are not tuned correctly it may result in over-fitting.

•

XG Boost techniques for imbalanced data

- XGBoost (Extreme Gradient Boosting) is an advanced and more efficient im
- Advantages over Other Boosting Techniques
    - It is 10 times faster than the normal Gradient Boosting as it implements parallel processing. It is highly flexible as users can define custom optimization objectives and evaluation criteria, has an inbuilt mechanism to handle missing values.
    - Unlike gradient boosting which stops splitting a node as soon as it encounters a negative loss, XG Boost splits up to the maximum depth specified and prunes the tree backward and removes splits beyond which there is an only negative loss.
    - Implementation of Gradient Boosting Algorithm

10 Techniques to deal with Imbalanced Classes in Machine Learning

- Random Under-Sampling
- Random Over-Sampling
- Random under-sampling with imblearn
- Random over-sampling with imblearn
- Under-sampling: Tomek links
- Synthetic Minority Oversampling Technique (SMOTE)
- NearMiss
- Change the performance metric
- Penalize Algorithms (Cost-Sensitive Training)
- Change the algorithm

References

- https://www.geeksforgeeks.org/ml-bias-variance-trade-off/
- https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205
- https://medium.com/swlh/imbalance-dataset-increasing-accuracy-in-machine-learning-using-imblearn-9cf1399e2319
- https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/
- https://www.youtube.com/watch?v=WtWxOhhZWX0
- https://www.youtube.com/watch?v=KIOeZ5cFZ50
- https://www.youtube.com/watch?v=NLRO1-jp5F8&t=1s
- https://www.youtube.com/watch?v=NoI1hVtLOSg&t=548s
- https://www.youtube.com/watch?v=gPciUPwWJQQ
- https://sebastianraschka.com/blog/2018/model-evaluation-selection-part4.html