Artificial Learning Models

Lecture 6 : Ensemble Learning (Boosting and Bagging)

By : Pr. Lamri SAYAD

2025

Agenda

- Ensemble learning
- Bagging
- Boosting
- Handling Imbalanced datasets

Objectives

- Combining similar classifiers to improve performance
- Applying the AdaBoost algorithm
- Dealing with classification imbalance

Introduction

- Many candidate algorithms in machine learning for classification and regression
- Why should the problems you address with machine learning be solved by only one algorithm?

- Instead of using one isolated algorithm, improved performance can be obtained by combining different algorithms
  - Meta-algorithms
  - What is meta-algorithms ?
    - Meta-algorithms are a way of combining other algorithms.

Enemble learning (Boosting, bagging …)

Ensemble learning or meta-algorithms

- Methods that do this are known as *ensemble methods or meta-algorithms*.

- Potential approaches:
  - Apply different algorithms to the same dataset
  - Apply the same algorithm with different settings
  - Assign different parts of the dataset to different classifiers

What is Ensemble learning ?

- Ensemble learning is a technique where **multiple models (often called "base learners") are combined to solve a problem and improve overall performance** compared to a single model.
- The idea is that by aggregating the predictions of several models, you can :
  - reduce errors,
  - increase accuracy,
  - and make the system more robust.
- Key ides:
  - Even if individual models are weak or prone to errors, combining them often yields a stronger predictor.

Ensemble learning techniques

- Basic methods
  - Voting Ensemble (for classification): Combine predictions of several models via:
    - Majority voting (hard voting)
    - Weighted voting (soft voting)
  - Averaging Ensemble (for regression): Take the mean of predictions from different models.
    - Averaging
    - Weighted Averaging
- Advanced methods
  - Bagging (Bootstrap Aggregating)
  - Boosting
  - Stacking (Stacked Generalization)

Voting Ensemble learning

- A **voting ensemble** is one of the simplest **ensemble learning techniques**,
- Mainly used for **classification problems**.
- It combines the predictions of multiple models (called base learners) to make a **final decision**, based on a “voting” mechanism.

Voting Ensemble learning

How it works?

1. You train several models independently on the same dataset (e.g., Decision Tree, KNN, Logistic Regression).

2. Each model makes a prediction for a given instance.

3. The ensemble aggregates these predictions to produce a final prediction.

Voting Ensemble learning

Types of voting

1. **Hard Voting (**also called **Max voting)**

- Each model casts a “vote” for a class.
- The class with the **majority of votes** is chosen as the final prediction.
- Example:
  - classifier 1 predicts – class A
  - classifier 2 predicts – class B
  - classifier 3 predicts – class B

  } final prediction here would be class B

2. **Soft Voting**

- Each model predicts **probabilities** for each class.
- The probabilities are **averaged**, and the class with the **highest average probability** is selected.

Voting Ensemble learning

Key points

- Voting ensembles work best when the **base models are diverse** (different algorithms or different hyperparameters), because similar models will often make the same mistakes, reducing the benefit of voting.
- Soft voting is more flexible and usually performs better than hard voting if models can output probabilities.

Voting Ensemble learning

Example in Python (using sklearn)

```
from sklearn.ensemble     import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree         import DecisionTreeClassifier
from sklearn.svm          import SVC
        # Create base learners
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC(probability=True)
        # Create voting ensemble
ensemble = VotingClassifier(estimators=[('lr', model1), ('dt', model2), ('svc',
model3)],    voting='soft')  # 'hard' for majority vote
        # Train
ensemble.fit(X_train, y_train)
        # Predict
y_pred = ensemble.predict(X_test)
```

Averaging Ensemble learning

- An **Averaging Ensemble** is a simple but powerful **ensemble learning technique**,
- Mainly used for **regression problems** (though it can also be adapted for classification with probabilities).
- It works by **averaging the predictions** of multiple models to produce a more stable and accurate final output.

Averaging Ensemble learning
How it works

- Instead of relying on one model, you train **several models** (e.g., Linear Regression, Decision Tree Regressor, Random Forest, etc.),
- And then combine their predictions by **taking the mean (average)**.

Averaging Ensemble learning

Types of averaging

1. **Simple (Uniform) Averaging**
   - All models contribute **equally** to the final prediction.

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} \hat{y}_i$$

2. **Weighted Averaging**
   - Some models are considered more reliable and given **higher weights**.

$$\hat{y} = \sum_{i=1}^{n} w_i \hat{y}_i, \quad \text{where} \sum w_i = 1$$

Averaging Ensemble learning
Example in Python (using sklearn)

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree         import DecisionTreeRegressor
from ensemble             import RandomForestRegressor
        # Train base models
model1 = LinearRegression().fit(X_train, y_train)
model2 = DecisionTreeRegressor().fit(X_train, y_train)
model3 = RandomForestRegressor().fit(X_train, y_train)
        # Get predictions
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)
        # Simple averaging
avg_pred = (pred1 + pred2 + pred3) / 3
        # Weighted averaging example
weights = [0.5, 0.3, 0.2]
weighted_avg_pred = np.average([pred1, pred2, pred3], axis=0, weights=weights)
```

Advanced ensemble learning techniques

- Boosting
- Bagging
- Stacking

Ensemble learning: Summary

- Differ in training strategy, and combination method
  - Parallel training with different training sets
    1. Bagging (bootstrap aggregation) { train separate models on overlapping training sets, average their predictions
  - Parallel training with objective encouraging division of labor: mixture of experts
  - Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting
- Notes:
  - Typically applied to weak models, such as decision stumps (single-node decision trees), or linear classifiers

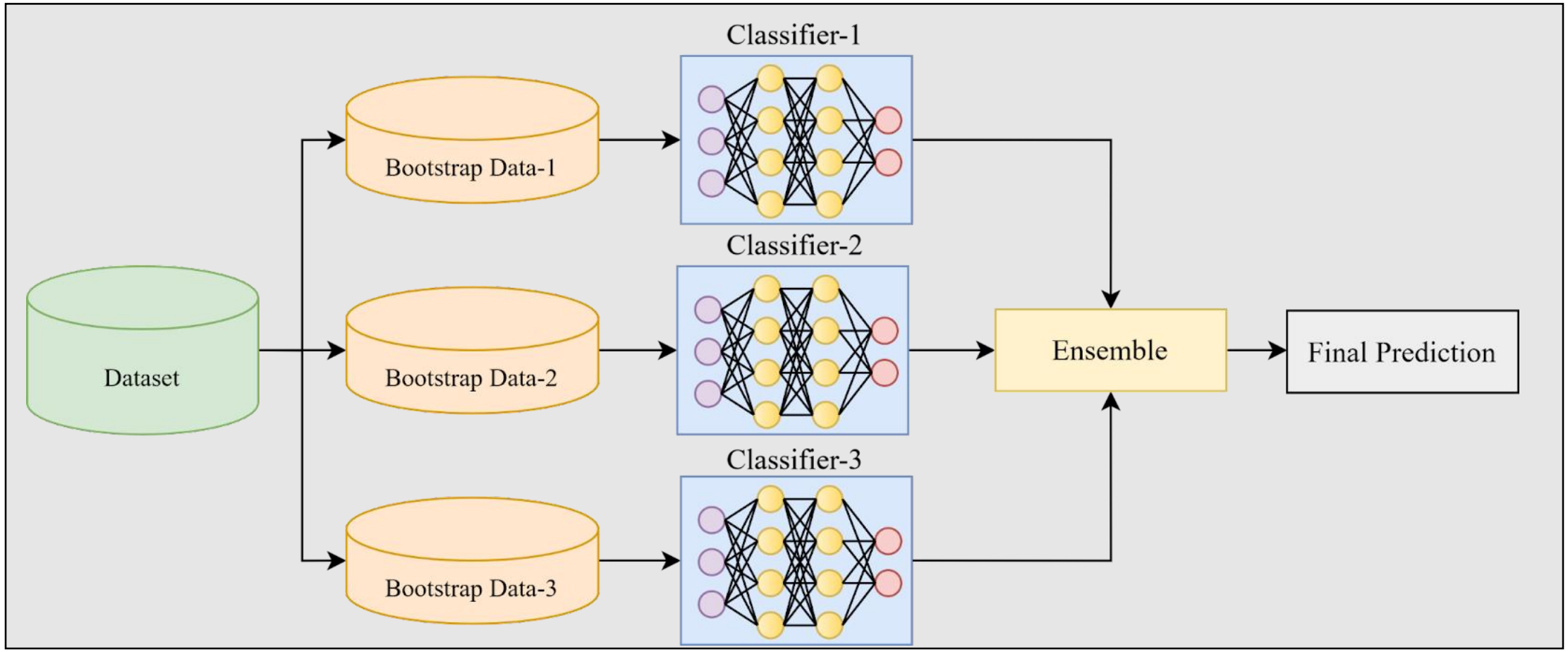
Bagging (Bootstrap aggregating)

- It is short for *Bootstrap Aggregating*
- It is a technique for reducing generalization error by combining several models
  - The idea is to create several versions of a model using **different samples of the training data**, train each model independently, and then **aggregates their predictions** (by averaging or voting).
- This model combination works because different models will not make the same mistake
- A widely used and effective machine learning algorithm based on the idea of bagging is random forest.

Bagging (Bootstrap aggregating)
How it works

1. Bootstrap sampling:
   - From the original training set of size **N**, generate **k** new datasets, each also of size **N**, by sampling with replacement.
   - This means some samples may appear multiple times, others may not appear at all in a given dataset.
2. Model training:
   - Train one model (base learner) on each of these bootstrap samples.
   - Typically, all models are of the same type (e.g., Decision Trees).
3. Aggregation:
   - Combine the outputs of all models:
     - For classification → majority vote.
     - For regression → average of predictions.

Bagging (Bootstrap aggregating)



Bagging Algorithm

Training Phase

Initialize the parameters

D={Φ}

h=the number of classification

For k=1 to h

Take a bootstrap sample Sk from training set S

Build the classifier Dk using Sk as training set

D=DUDi

Return D

Classification Phase

Run D1,D2,……..Dk on the input k

The class with maximum number of vote is choosen as the label for X.

Bagging (Bootstrap aggregating)
Python Example

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree                    import DecisionTreeClassifier
        # Base learner
base_model = DecisionTreeClassifier()
        # Bagging ensemble
bag_model = BaggingClassifier(estimator=base_model, n_estimators=10,
random_state=42)
        # Train
bag_model.fit(X_train, y_train)
        # Predict
y_pred = bag_model.predict(X_test)
```

Boosting

- Similar to bagging
- Different classifiers are trained sequentially.
- Each new classifier is trained based on the performance of those already trained.
- New classifiers focus on data that was previously misclassified by previous classifiers.
- The most popular boosting version, called AdaBoost.
- Boosting is different from bagging because the output is calculated from a weighted sum of all classifiers.
- It's one of the most effective methods in modern machine learning for both **classification** and **regression**.

Boosting
Main idea

- Unlike **bagging** (which trains models independently in parallel), **boosting trains models sequentially**, where:
  - The first model makes predictions.
  - The second model tries to fix the mistakes of the first one.
  - The third model corrects the errors of the second, and so on.

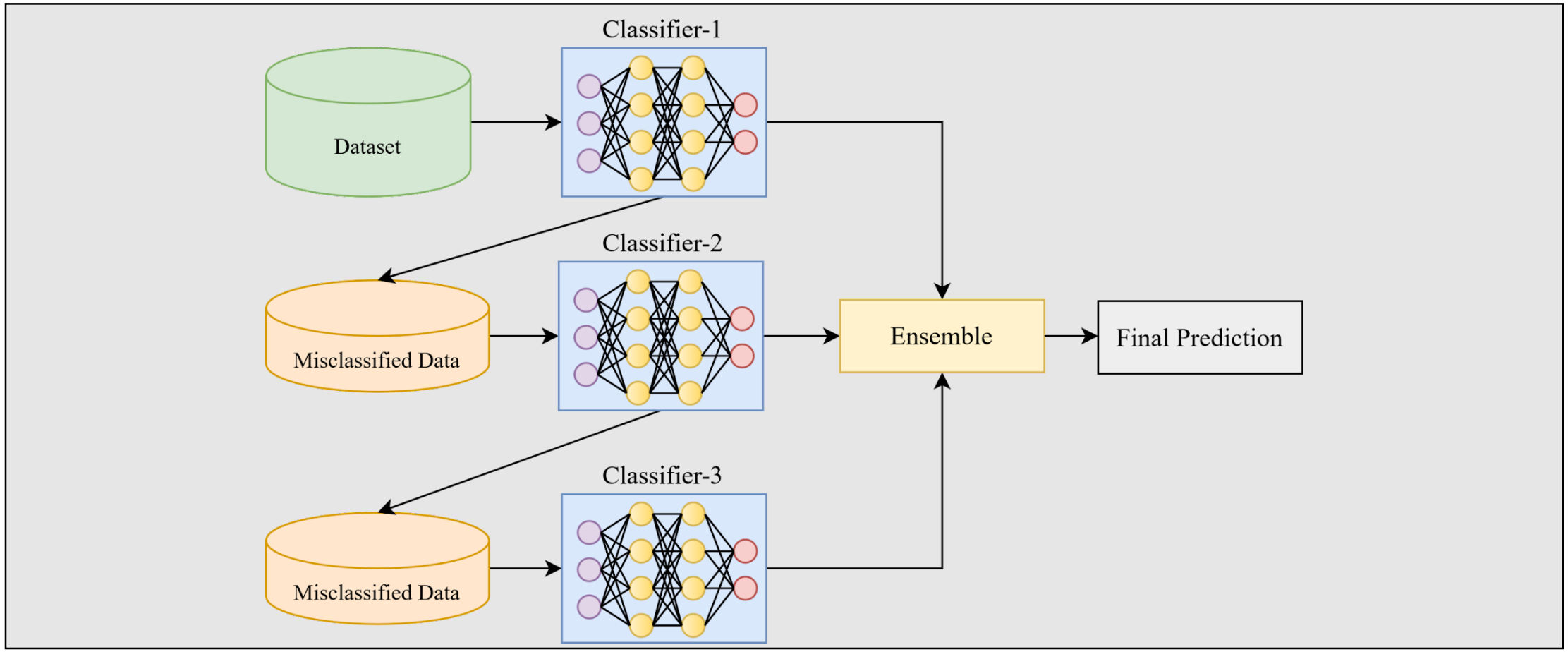
- At the end, all models are **combined (usually by weighted sum)** to make the final prediction.

Boosting

How it works ?

1. Start by training a weak model (e.g., a small decision tree).
2. Evaluate errors — find which samples were predicted incorrectly.
3. Increase the importance (weight) of those misclassified samples so the next model focuses more on them.
4. Train the next model on the updated (reweighted) dataset.
5. Repeat for several iterations.
6. Combine all models' predictions (weighted sum or vote).

Boosting


Classifier-1
Dataset
Classifier-2
Misclassified Data
Ensemble
Final Prediction
Classifier-3
Misclassified Data

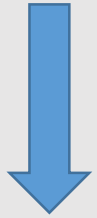
Boosting

Common boosting algorthims

- AdaBoost (Adaptive Boosting)
- Gradient Boosting
- XGBoost
- LightGBM
- CatBoost

AdaBoost (Adaptive Boosting)

- Question : can we take a weak classifier and use multiple instances of it to create a strong classifier?
  - Weak classifier : its error rate is greater than 50% in the two-class case.
  - Strong classifier : will have a much lower error rate.



AdaBoost (Adaptive Boosting)

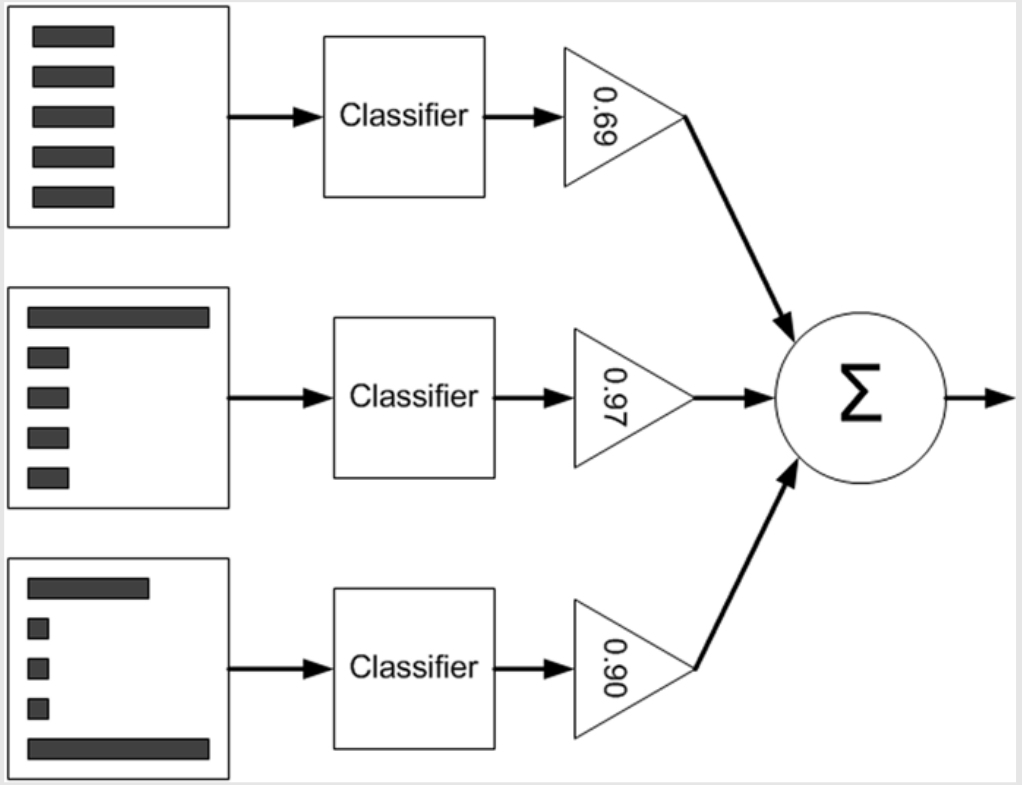
AdaBoost

- Most widely used form of boosting is the AdaBoost algorithm
- Boosting can give good results even if base classifiers have performance, only slightly better than random
  - Hence base learners are called weak learners
- Boosting can be extended to regression

AdaBoost

1. A weight is assigned to every example in the training data.
2. Initially, these weights are all equal.
3. The first weak classifier is trained on the training data.
4. The errors from the weak classifier are calculated,
5. The weights of the training set are adjusted so that the examples properly classified the first time are weighted less and the examples incorrectly classified in the first iteration are weighted more.
6. The second weak classifier is trained with the adjusted dataset.
7. This process is iterated until all classifiers are trained.
8. To get one answer from all of these weak classifiers, AdaBoost assigns α values to each of the classifiers.

AdaBoost



AdaBoost

- Error:

$$\varepsilon = \frac{number\ of\ incorrectly\ classified\ examples}{total\ number\ of\ examples}$$

- Alpha is given by

$$\alpha = \frac{1}{2} ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

- After you calculate α, the weight vector is updated

Boosting
Python Example

```
from sklearn.ensemble      import AdaBoostClassifier
from sklearn.tree          import DecisionTreeClassifier
        # Base learner
base_model = DecisionTreeClassifier(max_depth=1)
        # Boosting ensemble
boost_model = AdaBoostClassifier(estimator=base_model, n_estimators=50,
learning_rate=1.0, random_state=42)
        # Train and predict
boost_model.fit(X_train, y_train)
y_pred = boost_model.predict(X_test)
```

Boosting vs Bagging

- The base classifiers are trained in sequence
- base classifier is trained using a weighted form of the data set in which the weighting coefficient associated with each data point depends on the performance of the previous classifiers
  - Misclassified points are given greater weight when used to train the next classifier
- Once all classifiers have been trained, their predictions are combined through a weighted majority weighting scheme

Handling Imbalanced Datasets

What is Imbalanced Data

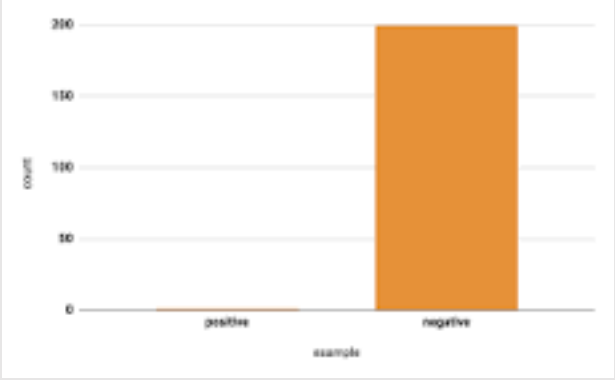
- datasets where the target class has an uneven distribution of observations
- one class label has a very high number of observations and the other has a very low number of observations.
- Example
  - A bank is concerned that some fraudulent transactions are going
  - when the bank checks their data they found that for each 2000 transaction there are only 30 of fraud recorded.
  - So, the number of fraud per 100 transactions is less than 2%, or we can say more than 98% transaction is “No Fraud” in nature.
  - Here, the class “No Fraud” is called the **majority class,**
  - and the much smaller in size “Fraud” class is called the **minority class**.

What is Imbalanced Data

- More such example of imbalanced data is:
  - Disease diagnosis
  - Customer churn prediction
  - Fraud detection
  - Natural disaster
- Class imbalanced is generally normal in classification problems.
- But, in some cases, this imbalance is quite critical where the majority class's presence is much higher than the minority class.

What is Imbalanced Data

- In a balanced dataset, the number of Positive and Negative labels is about equal.
- However, **if one label is more common than the other label, then the dataset is imbalanced**.
- The predominant label in an imbalanced dataset is called the majority class;
- The less common label is called the minority class.



Problems with Handling Imbalanced Data Classification

- Example of disease diagnosis
  - we are going to predict disease from an existing dataset where for every 100 records only 5 patients are diagnosed with the disease.
  - the majority class is 95% with no disease and the minority class is only 5% with the disease.
  - assume our model predicts that all 100 out of 100 patients have no disease.
  - the accuracy of the model from the confusion matrix is the total number of correct predictions by the model divided by the total number of predictions.

	Original	
Prediction	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

ACCURACY = TP + TN / TP+TN+FP+FN

  - In the above case it is (0+95)/(0+95+0+5)=0.95 or 95%. It means that the model fails to identify the minority class yet the accuracy score of the model will be 95%.

Resampling techniques

Modifies data to balance classes through oversampling and undersampling

Data augmentation

Creates dummy data by transforming existing data with operations like rotation and reflection.

SMOTE

Creates synthetic examples for the minority class in unbalanced data by interpolating between existing examples.

Ensemble techniques

Combines multiple models to improve overall performance.

One-class classification

Trains model to identify data points that don't belong to that class. Useful for identifying anomalies and outliers.

Cost-sensitive learning

Adjusts the cost of misclassifying data points to account for class imbalance.

Evaluation metrics

Evaluate model performance on imbalanced data.

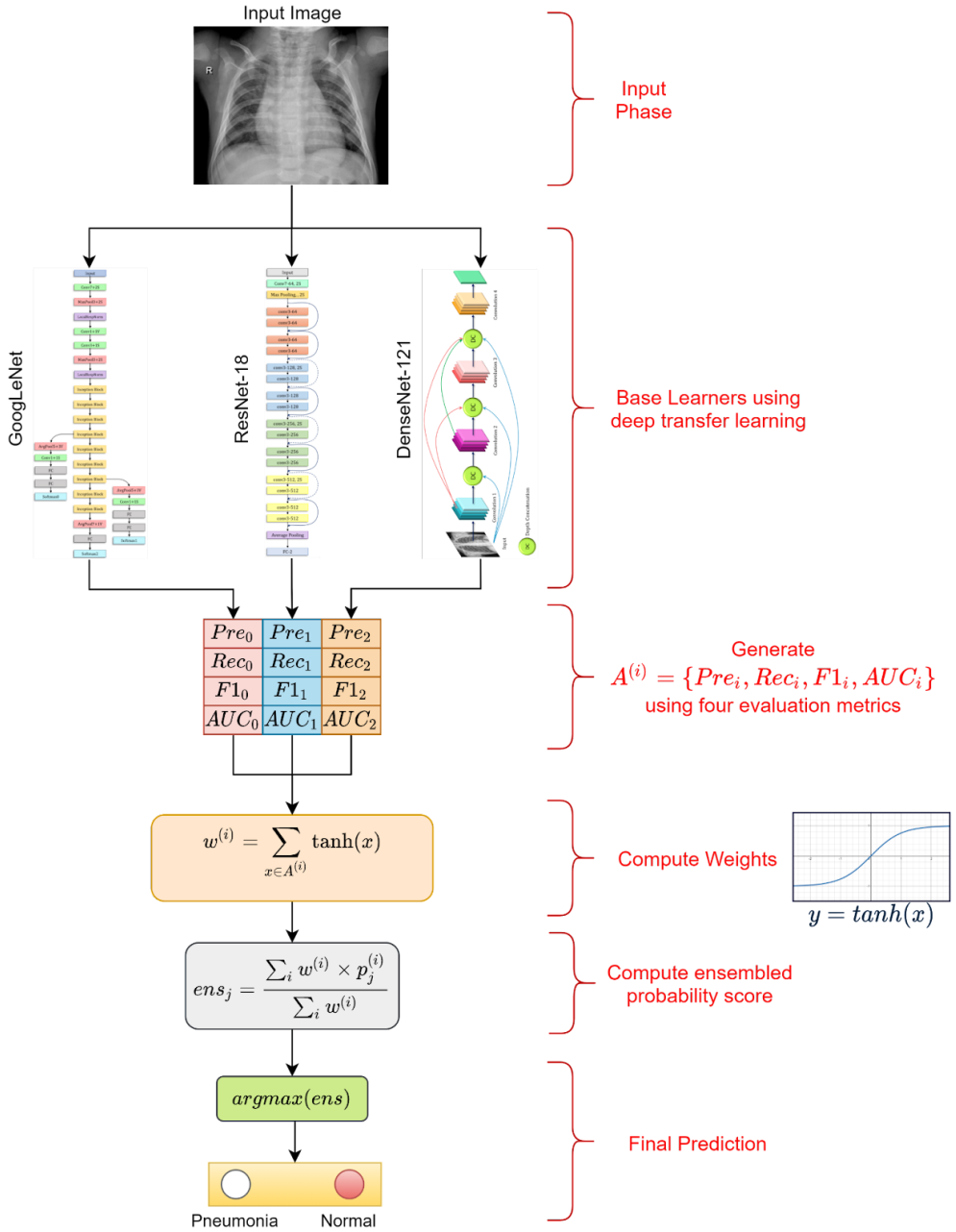
Handling Imbalanced Datasets

- Solutions:
  - Assign weights to classes
    - Assign a weight for every class.
    - The learning algorithm takes this information into account when looking for the best hyperplane.
  - Oversampling the minority class:
    - If a learning algorithm doesn't allow weighting classes, you can try the technique of over-sampling.
    - It consists of increasing the importance of examples of some class by making multiple copies of the examples of that class.
  - Undersampling the majority class
    - Randomly remove from the training set some examples of the majority class.
  - Synthetic oversampling (resampling):
    - SMOTE
    - ADASYN

Handling Imbalanced Datasets

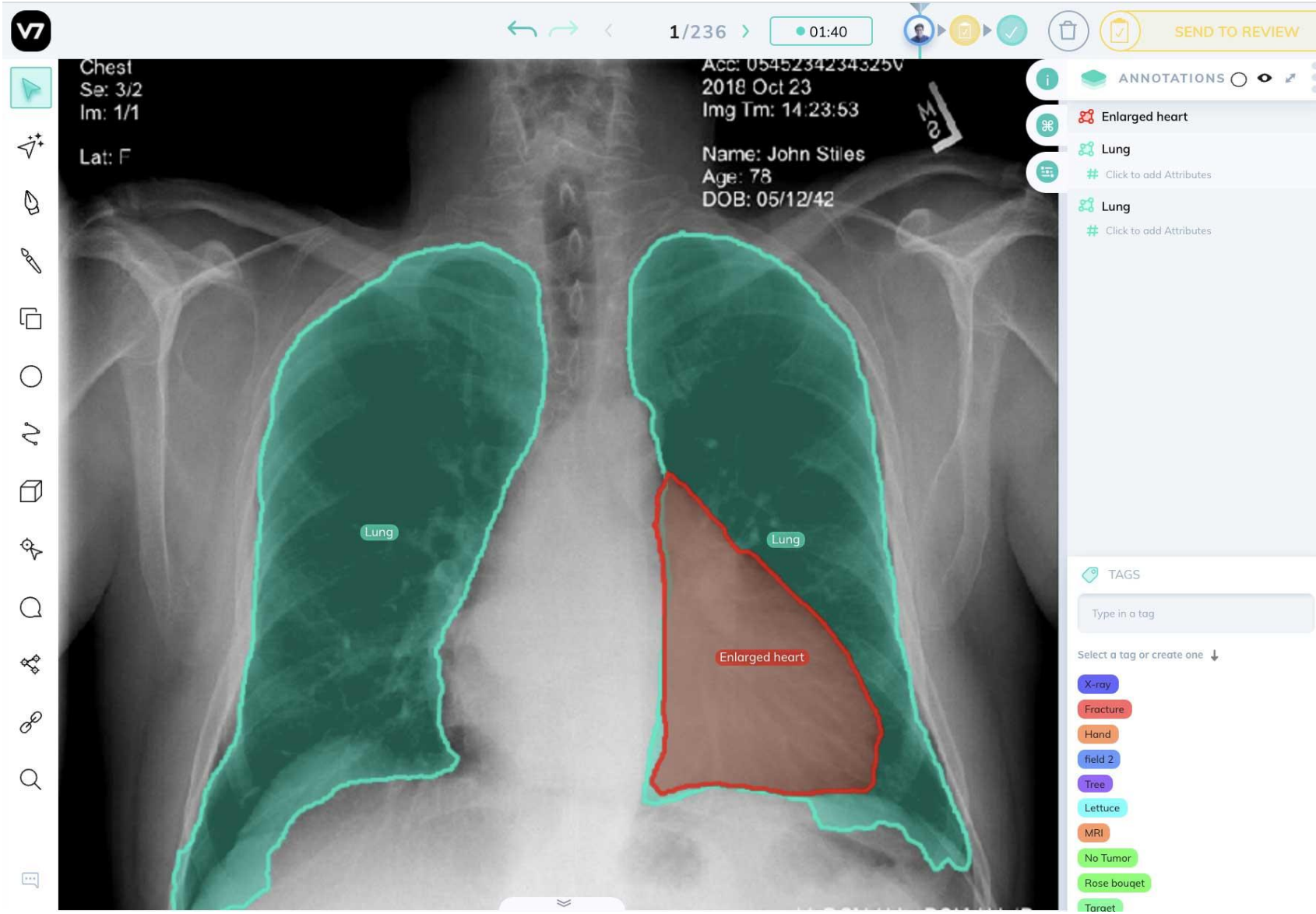
- **Data augmentation**
  - applying various transformations such as rotations, translations, and flips to the existing data.
- **Synthetic minority over-sampling technique (SMOTE)**
  - is a type of oversampling technique
  - involves creating synthetic examples of the minority class.
- **Ensemble techniques**
  - involve combining multiple models to improve performance.
  - bagging, boosting, and stacking.

Example


Input Image
Input Phase
GoogLeNet
ResNet-18
DenseNet-121
Base Learners using deep transfer learning
Pre_0 Pre_1 Pre_2
Rec_0 Rec_1 Rec_2
$F1_0$ $F1_1$ $F1_2$
AUC_0 AUC_1 AUC_2
Generate
$A^{(i)} = \{Pre_i, Rec_i, F1_i, AUC_i\}$
using four evaluation metrics
$w^{(i)} = \sum_{x \in A^{(i)}} \tanh(x)$
Compute Weights
$y = tanh(x)$
$ens_j = \frac{\sum_i w^{(i)} \times p_j^{(i)}}{\sum_i w^{(i)}}$
Compute ensembled probability score
$argmax(ens)$
Pneumonia
Normal
Final Prediction

Applications

- **Disease detection**

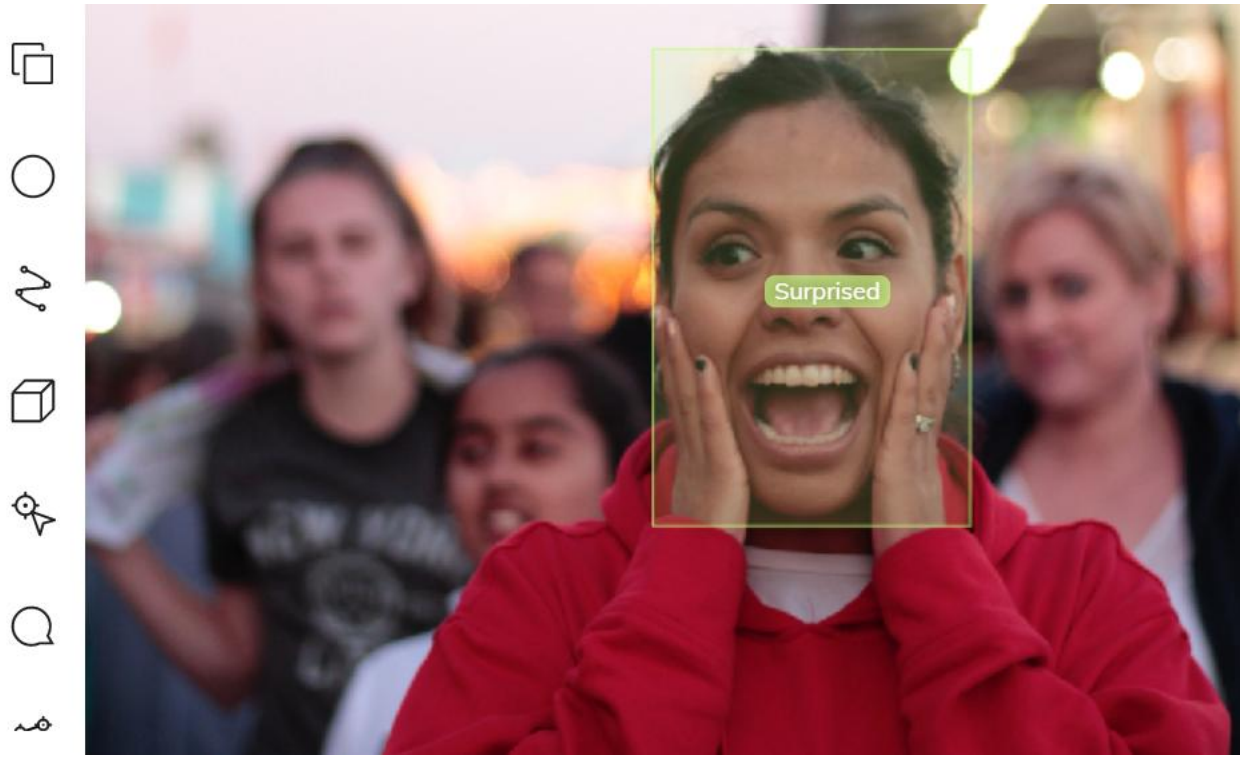


Applications

- **Remote Sensing**
- Monitoring of physical characteristics of a target area without coming in physical contact,

Applications

- **Speech emotion recognition**



Libraries for ensemble learning

- from sklearn.ensemble import BaggingClassifier
- from sklearn.tree import DecisionTreeClassifier
- bagging = BaggingClassifier(base_estimator=DecisionTreeClassifier(),n_estimators=10, max_samples=0.5, max_features=0.5)

- from sklearn.ensemble import BaggingRegressor
- bagging = BaggingRegressor(DecisionTreeRegressor())
- bagging.fit(X_train, y_train)
- model.score(X_test,y_test)

Libraries for ensemble learning

- from sklearn.ensemble import AdaBoostClassifier
- model = AdaBoostClassifier(n_estimators=100)
- model.fit(X_train, y_train)
- model.score(X_test,y_test)

- from sklearn.ensemble import GradientBoostingClassifier
- model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
- model.fit(X_train, y_train)
- model.score(X_test,y_test)