**Ensemble Learning (with Python)**

Ensemble learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

The underlying concept behind ensemble learning is to combine the outputs of diverse models to create a more precise prediction. By considering multiple perspectives and utilizing the strengths of different models, ensemble learning improves the overall performance of the learning system. This approach not only enhances accuracy but also provides resilience against uncertainties in the data. By effectively merging predictions from multiple models, ensemble learning has proven to be a powerful tool in various domains, offering more robust and reliable forecasts.

**Common Ensemble Techniques**

In this section, we will look at a few simple but powerful techniques, namely:

1. Voting
2. Bagging
3. Boosting

**Ensemble learners and base learners**

Ensemble learning uses multiple machine learning models to try to make better predictions on a dataset. An ensemble model works by training different models on a dataset and having each model make predictions individually. The predictions of these models are then combined in the ensemble model to make a final prediction.

Every model has its strengths and weaknesses. Ensemble models can be beneficial by combining individual models to help hide the weaknesses of an individual model.

In this tutorial, we will use a Voting Classifier in which the ensemble model makes the prediction by majority vote. For example, if we use three models and they predict [1, 0, 1] for the target variable, the final prediction that the ensemble model would make would be 1, since two out of the three models predicted 1.

We will use two different models to put into our Voting Classifier: Naïve Bayes, Logistic Regression. We will use the Scikit-learn library in Python to implement these methods and use the impact of mobile phone on students’ academic performance dataset in our example.

This dataset explores the relationship between students' health and use of mobile phone and their academic performance. It contains multiple rows of data, each representing a student, and multiple columns, including variables such as:

* ***Names: Student names***
* ***Age: Student age range:***

**(0:16-21 / 1:21-25 / 2:26-30 / 3:31-35).**

* ***Gender:***

**(Female:0 / Male:1)**

* ***Mobile phone: Do students own a mobile phone?***

**(No:0 / Yes:1)**

* ***Mobile Operating System: Type of mobile operating system used***

(**Android:0 / iOS:1)**

* ***Mobile phone use for education : Do students use their mobile phone for educational purposes?***

**(Sometime:2 / Frequently: 0 / Rarely:1)**

* ***Mobile phone activities : List of mobile phone activities for educational purposes***

**(All of these: 0 / Messaging:1 / Social media:2 / Web-browsing:3**)

* ***Helpful for studying : Do students find mobile phone use helpful for studying?***

(**Yes :1 / No:0**)

* ***Educational Apps : List of educational apps used.***

**(Educational videos:0 / Language:1 / Productivity tools:2 / Study planner:3)**

* ***Daily usages : Average daily time spent using mobile phone for educational purposes (in hours)***

(**<2:0 / 2-4:1 / 4-6:2 / >6:3)**

* ***Performance impact : How does mobile phone use impact academic performance?***

(**Strongly Agree:5 / Agree:4 / Neutral:3 / Disagree:2 / Strongly disagree:1**)

* ***Usage distraction: Does mobile phone use distract from studying?***

(**During Exams:1 / Not Distracting:2 / During Class Lectures:0 / While Studying:3**)

* ***Attention span : Has mobile phone use affected attention span?***

(**Yes:1 / No:0**)

* ***Useful features : What features of mobile phones are useful for learning?***

**(Internet Access:2 / Camera:1 / Calculator:0 / Notes Taking App:3 )**

* ***Health Risks : Are students aware of potential health risks associated with excessive mobile phone use?*** (**Yes:2 / No:0 / Only Partially:1**)
* ***Beneficial subject : Which subjects benefit most from mobile phone use?***

**(Coding:1 / Browsing Material:0 / Research:2)**

* ***Usage symptoms : Are students experiencing any physical or mental symptoms related to mobile phone*** use?

**(Sleep disturbance:4 / headaches:2 / Anxiety or:1 Stress / All symptoms:0 / none:3)**

* ***Symptom frequency : How often are symptoms experienced?***

**(Sometimes:3 / Never:1 / Rarely:2 / Frequently:0)**

* ***Health precautions : Are students taking precautions to mitigate potential health risks?***

(T**aking Break during prolonged use: 2 / Using Blue light filter:3 / Limiting Screen Time:0 / None of Above:1**)

* ***Health rating : How would students rate their overall academic performance?***

**(Good: 1 / Fair: 0 / Poor: 2)**

***This dataset can be used to analyse the impact of health and mobile phone use on academic success, identify potential predictors of academic performance, and inform interventions to support students' overall well-being and academic achievement.***

Note: Ensemble models can also be used for regression problems, where the ensemble model will use either the mean output of the different models or weighted averages for its final prediction.

**Load your dataset**

We will use the Scikit-learn library in Python model the impact of mobile phone on students’ academic performance dataset in our examples.

#Load your dataset: we are using Impact of Mobile Phone on Students Performance dataset

import pandas as pd

#load the dataset

df = pd.read\_csv('/content/Impact\_of\_Mobile\_Phone\_on\_Students\_Performance.csv')

#take a look at the data

df.head()

A screenshot of a black and white screen

Description automatically generated

**Explore and preprocess your data**

1) You will notice that from the questionnaire you received that all the variables are of categorical nature, therefore we convert their type from numeric to object. Then, display their basic stats.

#Display the basic stats

df.astype('object').describe().transpose()

A screenshot of a computer

Description automatically generated

2) Define your X inputs and your target y variables. Observe that “Names” is an unnecessary variable as it neither affects the target nor any other input. While the variable “Mobile Phone” is useless since it has no change of values. Drop these variables from your data, then split the dataset into training and test subsets.

#Split the data into inputs and targets

X = df.drop(columns = ['Names', 'Mobile Phone', ’Academic performance’])

y = df[' Academic performance']

#Split data into train and test sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y)

**Building Base learners**

1)Let’s build three different base model, first multinomial naïve bayes. Test the NB model by performing predictions on the test data, then display the confusion matrix and the classification report.

# import the NB algorithm, build the NB model then test it by predicting on the test data

from sklearn.naive\_bayes import MultinomialNB

nb=MultinomialNB()

nb=nb.fit(X\_train,y\_train)

y\_pred\_nb=nb.predict(X\_test)

# Evaluate your NB model byt generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for NB")

print(classification\_report(y\_test,y\_pred\_nb))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for NB")

nb\_cm=confusion\_matrix(y\_test,y\_pred\_nb)

disp=ConfusionMatrixDisplay(confusion\_matrix=nb\_cm,display\_labels=nb.classes\_)

disp.plot()

A screenshot of a computer

Description automatically generated

2)Let’s build a logistic regression base learner. Test the LR model by performing predictions on the test data, then display the confusion matrix and the classification report.

# import the LR algorithm, build the LR model then test it by predicting on the test data

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression()

lr=lr.fit(X\_train,y\_train)

y\_pred\_lr=lr.predict(X\_test)

# Evaluate your LR model byt generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for LR")

print(classification\_report(y\_test,y\_pred\_lr))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for lr")

lr\_cm=confusion\_matrix(y\_test,y\_pred\_lr)

disp=ConfusionMatrixDisplay(confusion\_matrix=lr\_cm,display\_labels=lr.classes\_)

disp.plot()

A screenshot of a computer

Description automatically generated

3)Let’s build a decision tree base learner. Test the DT model by performing predictions on the test data, then display the confusion matrix and the classification report.

# import the DT algorithm, build the DT model then test it by predicting on the test data

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()

dt=dt.fit(X\_train,y\_train)

y\_pred\_dt=dt.predict(X\_test)

# Evaluate your DT model by generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for DT")

print(classification\_report(y\_test,y\_pred\_dt))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for DT")

dt\_cm=confusion\_matrix(y\_test,y\_pred\_dt)

disp=ConfusionMatrixDisplay(confusion\_matrix=dt\_cm,display\_labels=dt.classes\_)

disp.plot()

A screenshot of a computer

Description automatically generated

**Voting (Hard Voting)**

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.

For example, when you asked 5 of your colleagues to rate your movie (out of 5); we’ll assume three of them rated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken as 4. **You can consider this as taking the mode of all the predictions.**

1) let’s combine the weakest two learners from the three base learners previously built, NB and LR.

#initiate a new ensemble model

from sklearn.ensemble import VotingClassifier

#create a dictionary of our base learner models

base\_learners=[('NB', nb), ('LR', lr)]

#create our voting classifier, inputting our models

ensemble\_learner = VotingClassifier(base\_learners, voting='hard')

#fit model to training data

ensemble\_learner = ensemble\_learner.fit(X\_train, y\_train)

y\_pred\_ensembler = ensemble\_learner.predict(X\_test)

# Evaluate your ensemble model by generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for Ensmebler")

print(classification\_report(y\_test,y\_pred\_ensembler))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for ensember learner")

ensemble\_learner\_cm=confusion\_matrix(y\_test,y\_pred\_ensembler)

disp=ConfusionMatrixDisplay(confusion\_matrix=ensemble\_learner\_cm,display\_labels = ensemble\_learner.classes\_)

disp.plot()

A screenshot of a computer screen

Description automatically generated

A colorful squares with numbers and labels

Description automatically generated

**Voting (Soft Voting)**

In soft voting, the base classifiers output probabilities or numerical scores. For instance, in our classification, logistic regression output can be interpreted as the probability of the object belonging to class 1.

**A soft-voting ensemble calculates the average score (or probability)** and compares it to a threshold value. Now let’s combine our base learners in a soft voting classifier:

#initiate a new ensemble model

from sklearn.ensemble import VotingClassifier

#create a dictionary of our base learner models

base\_learners=[('NB', nb), ('LR', lr)]

#create our voting classifier, inputting our models

ensemble\_learner = VotingClassifier(base\_learners, voting='soft')

#fit model to training data

ensemble\_learner = ensemble\_learner.fit(X\_train, y\_train)

y\_pred\_ensembler = ensemble\_learner.predict(X\_test)

# Evaluate your ensemble model by generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for Ensmebler")

print(classification\_report(y\_test,y\_pred\_ensembler))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for ensember learner")

ensemble\_learner\_cm=confusion\_matrix(y\_test,y\_pred\_ensembler)

disp=ConfusionMatrixDisplay(confusion\_matrix=ensemble\_learner\_cm,display\_labels = ensemble\_learner.classes\_)

disp.plot()

A screenshot of a graph

Description automatically generated

A chart of different colored squares

Description automatically generated

**Hard-voting ensembles output the mode** of the base classifiers’ predictions, whereas **soft-voting ensembles average predicted probabilities** (or scores).

**Bootstrap Aggregation (BAGGING)**

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here’s a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.

Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

A diagram of a data flow

Description automatically generated

Bagging algorithms is one of the most commonly used techniques in machine learning. In this section, we will look at them in detail.

**Bagging algorithms:**

***Random forest:*** Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Looking at it step-by-step, this is what a random forest model does:

* Random subsets are created from the original dataset (bootstrapping).
* At each node in the decision tree, only a random set of features are considered to decide the best split.
* A decision tree model is fitted on each of the subsets.
* The final prediction is calculated by averaging the predictions from all decision trees.

*Note: The decision trees in random forest can be built on a subset of data and features. Particularly, the sklearn model of random forest uses all features for decision tree and a subset of features are randomly selected for splitting at each node.*

***Random Forest has several hyper parameters:***

* n\_estimators:
  + It defines the number of decision trees to be created in a random forest.
  + Generally, a higher number makes the predictions stronger and more stable, but a very large number can result in higher training time.
* criterion:
  + It defines the function that is to be used for splitting.
  + The function measures the quality of a split for each feature and chooses the best split.
* max\_features :
  + It defines the maximum number of features allowed for the split in each decision tree.
  + Increasing max features usually improve performance but a very high number can decrease the diversity of each tree.
* max\_depth:
  + Random forest has multiple decision trees. This parameter defines the maximum depth of the trees.
* min\_samples\_split:
  + Used to define the minimum number of samples required in a leaf node before a split is attempted.
  + If the number of samples is less than the required number, the node is not split.
* min\_samples\_leaf:
  + This defines the minimum number of samples required to be at a leaf node.
  + Smaller leaf size makes the model more prone to capturing noise in train data.
* max\_leaf\_nodes:
  + This parameter specifies the maximum number of leaf nodes for each tree.
  + The tree stops splitting when the number of leaf nodes becomes equal to the max leaf node.
* n\_jobs:
  + This indicates the number of jobs to run in parallel.
  + Set value to -1 if you want it to run on all cores in the system.
* random\_state:
  + This parameter is used to define the random selection.
  + It is used for comparison between various models.

To sum up, Random Forest **randomly**selects data points and features and builds **multiple trees (Forest). Let’s build a random forest:**

#create a new random forest classifier and grid search cv

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

#create a dictionary of all values we want to test for n\_estimators

params\_rf = {'n\_estimators': [50, 100, 200]}

#use gridsearch to test all values for n\_estimators

rf\_gs = GridSearchCV(rf, params\_rf, cv=10)

#fit model to training data

rf\_gs.fit(X\_train, y\_train)

#save best model

rf\_best = rf\_gs.best\_estimator\_

#check best n\_estimators value

print(rf\_gs.best\_params\_)

#make prediction on the test data

y\_pred\_rf = rf\_best.predict(X\_test)

# Evaluate your rf model by generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for RF")

print(classification\_report(y\_test,y\_pred\_rf))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for RF")

rf\_cm=confusion\_matrix(y\_test,y\_pred\_rf)

disp=ConfusionMatrixDisplay(confusion\_matrix=rf\_cm,display\_labels = rf\_best.classes\_)

disp.plot()

A screenshot of a computer

Description automatically generated

Observe that the performance of this Random Forest is just better than the performance of a single fully grown decision tree.

**Boosting**

If a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.

Boosting is a sequential process where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let’s understand the way boosting works in the below steps.

1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole dataset.
5. Errors are calculated using the actual values and predicted values.
6. The observations which are incorrectly predicted, are given higher weights.
7. Another model is created and predictions are made on the dataset.  
   (This model tries to correct the errors from the previous model)
8. Similarly, multiple models are created, each correcting the errors of the previous model.
9. The final model (strong learner) is the weighted mean of all the models (weak learners).

Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

***AdaBoost:***Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted, and the subsequent model works to predict these values correctly.

***AdaBoost Hyperparameters***

* estimators:
  + It helps to specify the type of base estimator, that is, the machine learning algorithm to be used as base learner. If None, then the base estimator is **DecisionTreeClassifier** initialized with max\_depth=1.
* n\_estimators:
  + It defines the number of base estimators.
  + The default value is 10, but you should keep a higher value to get better performance.
* max\_depth:
  + Defines the maximum depth of the individual estimator.
  + Tune this parameter for best performance.
* random\_state :
  + An integer value to specify the random data split.
  + A definite value of random\_state will always produce same results if given with same parameters and training data.

#initiate AdaBoost Classifier

from sklearn.ensemble import AdaBoostClassifier

#specify the type of weak learner you want to boost, this is called base model

base\_model = DecisionTreeClassifier(max\_depth=1)

AdaBoost = AdaBoostClassifier(estimator=base\_model)

#create a dictionary of all values we want to test for n\_estimators

params\_AdaBoost = {'n\_estimators': [50, 100, 200, 500, 1000]}

#use gridsearch to test all values for n\_estimators

AdaBoost\_gs = GridSearchCV(AdaBoost, params\_AdaBoost, cv=10)

#fit model to training data

AdaBoost\_gs.fit(X\_train, y\_train)

#save best model

AdaBoost\_best = AdaBoost\_gs.best\_estimator\_

#check best n\_estimators value

print(AdaBoost\_gs.best\_params\_)

#make prediction on the test data

y\_pred\_AdaBoost = AdaBoost\_best.predict(X\_test)

# Evaluate your AdaBoost model by generating the classification report and the confusion matrix

from sklearn.metrics import classification\_report

print("Classification report for AdaBoost")

print(classification\_report(y\_test,y\_pred\_AdaBoost))

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

print("confusion\_matrix for Adaboost")

AdaBoost\_cm=confusion\_matrix(y\_test,y\_pred\_AdaBoost)

disp = ConfusionMatrixDisplay(confusion\_matrix=AdaBoost\_cm,display\_labels = AdaBoost\_best.classes\_)

disp.plot()

A screenshot of a computer

Description automatically generated

**Predict your potential academic performance**

1) You can use your best machine learning classifier to predict your potential academic performance, fill-in the survey form at hand, **DO NOT WRITE YOUR NAME** on the form.

2) Modify the following code block to produce predictions based on your entered data.

# Create a new DataFrame from scratch to predict Academic performance, we'll call this `df2`:

data = []

data.append( {"Age":value,

"Gender":value,

"Mobile Operating System":value,

"Mobile phone use for education":value,

"Mobile phone activities":value,

"Helpful for studying":value,

"Educational Apps":value,

"Daily usages":value,

"Performance impact":value,

"Usage distraction":value,

"Attention span":value,

"Useful features":value,

"Health Risks":value,

"Beneficial subject":value,

"Usage symptoms":value,

"Symptom frequency":value,

"Health precautions":value} )

df2 = pd.DataFrame(data)

# Add a new column to `df2` with the predicted prices:

df2["Predicted academic performance"] = model.predict(df2)

df2.head()

**Machine Learning ethical concerns**

Ethical issues with machine learning classification arise from the potential for bias, lack of transparency, and accountability, particularly when algorithms are used in high-stakes decision-making. Here's a breakdown of key ethical concerns and let’s relate these as much as possible to your previous prediction task:

* **Bias and Fairness:**
  + Machine learning models are trained on data, and if that data reflects existing societal biases, the models can perpetuate and even amplify those biases, leading to unfair or discriminatory outcomes.
  + For example, a loan application system trained on biased data might unfairly deny loans to certain demographic groups.
  + Mitigation: Careful data collection and preprocessing, bias detection and mitigation techniques, and ongoing monitoring are crucial.
* **Transparency and Explainability:**
  + "Black box" algorithms, where the decision-making process is opaque, can raise ethical concerns, especially in areas like criminal justice or healthcare.
  + Mitigation: Developing explainable AI (XAI) techniques that can provide insights into how models make decisions is important.
* **Accountability:**
  + When algorithms make decisions with real-world consequences, it's essential to establish who is accountable for those decisions and how to address errors or harms.
  + Mitigation: Clear lines of responsibility and mechanisms for redress are needed.
* **Privacy:**
  + Machine learning models often rely on large datasets, which can include sensitive personal information.
  + Mitigation: Data anonymization, differential privacy techniques, and robust data governance frameworks are necessary.
* **Economic Impact:**
  + The deployment of AI and ML can have significant economic consequences, potentially leading to job displacement or exacerbating inequality.
  + Mitigation: Addressing the economic impact of AI through retraining programs, social safety nets, and other policies is crucial.
* **Other Ethical Considerations:**
  + Data quality and quantity: Ensuring that the data used to train ML models is accurate, representative, and sufficient is essential.
  + Human oversight: Maintaining human oversight and control over AI systems, especially in critical applications, is important.