TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

VIỆN TRÍ TUỆ NHÂN TẠO

Machine learning final project report

Project: Mushroom edible classification

Giảng viên hướng dẫn: TS Trần Quốc Long

Sinh viên thực hiện: Nguyễn Xuân Hiệp – 22022591

Ngô Huy Hoàn – 22022590

Đàm Văn Hiển – 22022664

Lớp môn học: INT3403 3

Hà Nội, ngày 26 tháng 5 năm 2024

LỜI CẢM ƠN

Thưa thầy Trần Quốc Long!

Nhóm chúng em muốn gửi lời cảm ơn thầy đã đem đến cho sinh viên lớp học phần INT3405 3 kiến thức tổng quan, thú vị, dễ hiểu về môn học, tạo điều kiện và hướng dẫn cụ thể cho chúng em về đề tài cuối kỳ. Với đề tài cuối kỳ lần này, do chưa có nhiều kinh nghiệm về làm báo cáo cũng như những hạn chế về mặt kiến thức, trong bài báo cáo chắc chắn không thể tránh khỏi những thiếu sót, nhầm lẫn . Chúng em mong nhận được sự đóng góp, phê bình từ thầy để sản phẩm của em được hoàn thiện hơn. Em rất mong tiếp tục được thầy chỉ dạy ở các môn học khác.

Một lần nữa, em xin chân thành cảm ơn thầy!

Hà Nội, Tháng 5 năm 2024

Đại diện nhóm

Hiển

Đàm Văn Hiển

Bảng phân chia công việc

Công việc	Ngô Huy Hoàn	Đàm Văn Hiển	Nguyễn Xuân Hiệp
Tìm hiểu dữ liệu, visualize data	X	X	X
Mô hình kNN	X		
Mô hình Decision Tree		X	
Mô hình Logistic regression			X
Mô hình SVM	X		
Random forest		X	
Boosting			X
Viết báo cáo	X	X	X

Link github: https://github.com/hiendamvan/mushroom_edible_classification

Mục lục

I. Giới thiệu đề tài

Nấm là một loại sinh vật quen thuộc trong thế giới tự nhiên. Nấm có giá trị lớn trong nhiều lĩnh vực, như y học, môi trường và công nghiệp. Nấm được sử dụng trong sản xuất thuốc, chất cần thiết trong vi sinh vật học và làm phân bón hữu cơ. Ngoài ra, một số loài nấm được ưa chuộng trong ẩm thực vì hương vị độc đáo và giá trị dinh dưỡng cao. Tuy nhiên, một số loài nấm cũng có thể gây hại cho cây trồng, động vật hoặc có độc cho con người. Vì vậy, việc phân biệt các loại nấm, cũng như phân tích xem nấm có ăn được hay không, là một đề tài quan trọng trong ngành sinh học.

Xuất phát từ vấn đề đó, nhóm chúng em quyết định chọn đề tài Mushroom edible classification (Phân biệt nấm ăn được hay không) cho project của môn học này. Bài toán học máy của chúng em là binary classification có input và output như sau:
Input: Các thông số của một cây nấm ()
Output: 1 (Nấm ăn được), 0 (Nấm không ăn được)

II. Data visualization & pre processing data

- Link dataset: https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset/data
- Dataset gồm có 54035 dòng và 9 cột:

  * Các cột không có đơn vị là dữ liệu categorical

Feature	Variable Type	Variable	Đơn vị
Đường kính nắp	int	cap-diameter	cm
Hình dạng nắp	int	cap-shape	
Sự gắn kế của lá với thân	int	gill-attachment	
Màu lá tia	int	gill-color	
Chiều cao thân	float	stem-height	cm
Chiều rộng thân	int	stem-width	mm
Màu thân	int	stem-color	
Mùa	float	season	
Có độc hay không	int	class	

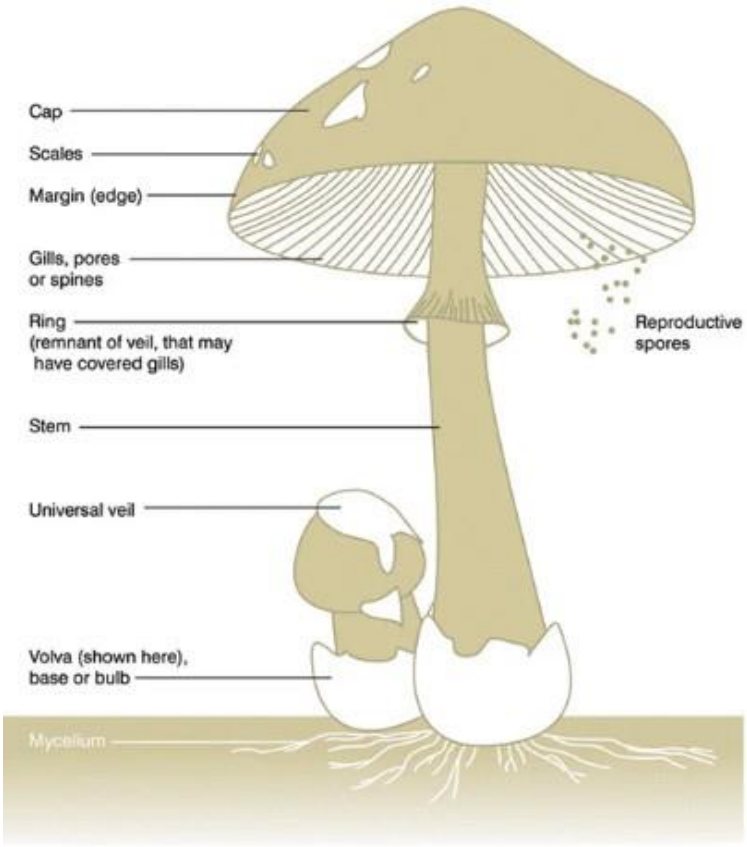
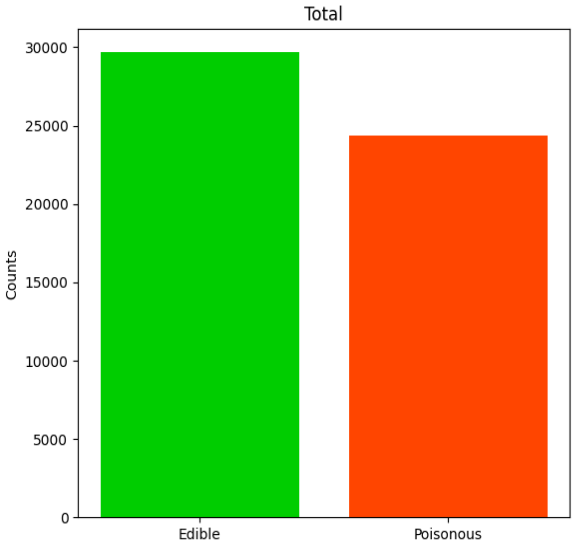


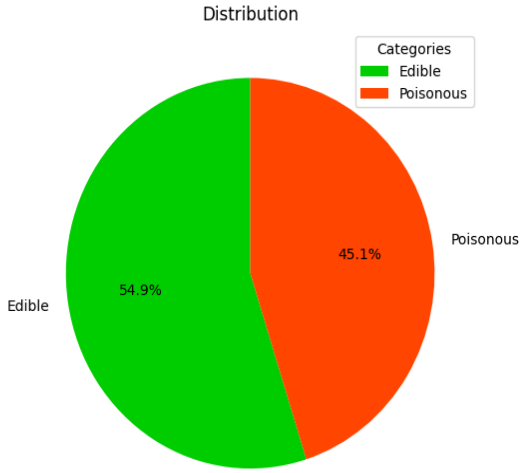
Image Credit: The Mushroom Diary

- Các thông số của dữ liệu:

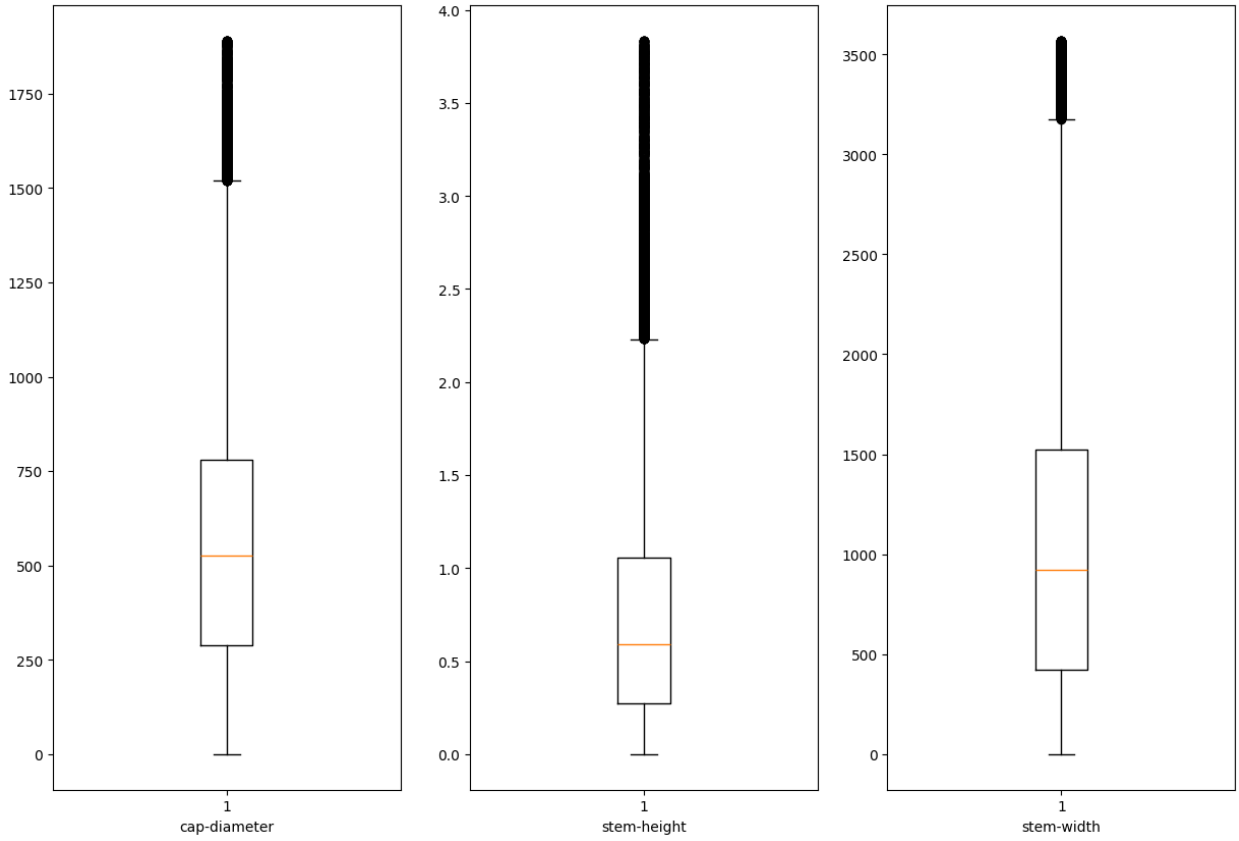
	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
count	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000
mean	567.257204	4.000315	2.142056	7.329509	0.759110	1051.081299	8.418062	0.952163	0.549181
std	359.883763	2.160505	2.228821	3.200266	0.650969	782.056076	3.262078	0.305594	0.497580
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	289.000000	2.000000	0.000000	5.000000	0.270997	421.000000	6.000000	0.888450	0.000000
50%	525.000000	5.000000	1.000000	8.000000	0.593295	923.000000	11.000000	0.943195	1.000000
75%	781.000000	6.000000	4.000000	10.000000	1.054858	1523.000000	11.000000	0.943195	1.000000
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273	1.000000

- Tỉ lệ giữa 2 lớp ăn được và không ăn được khá cần bằng => balanced dataset



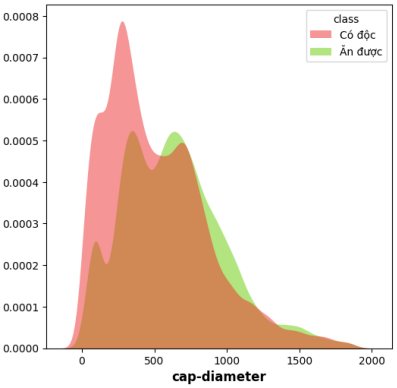


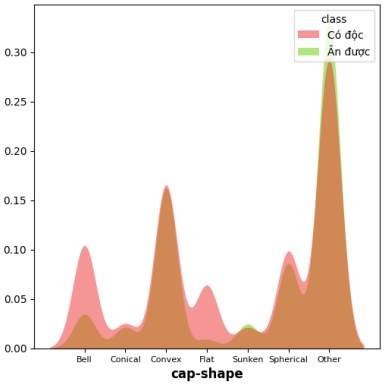
- Outlier dữ liệu:

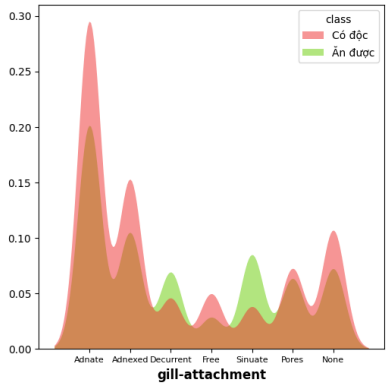


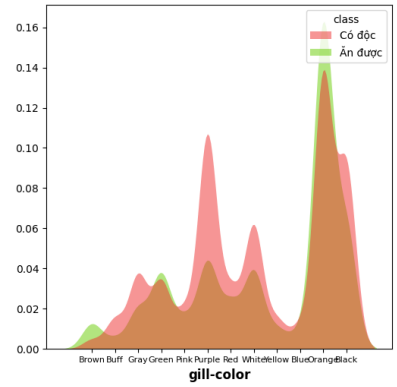
- Tương quan các trường dữ liệu phân tách bởi lớp ăn được hoặc không ăn được:

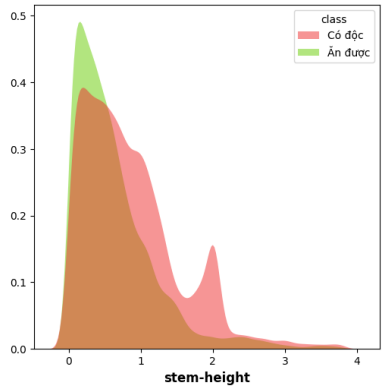
Tương quan
giữa nấm Ăn được và Có độc

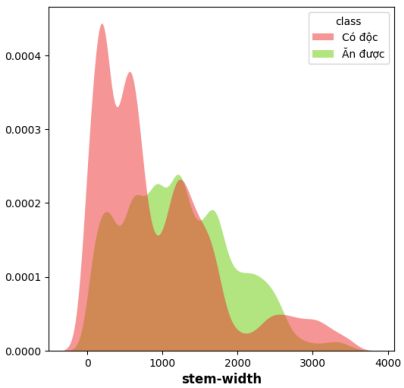


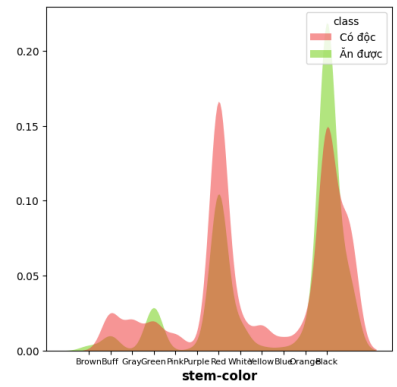


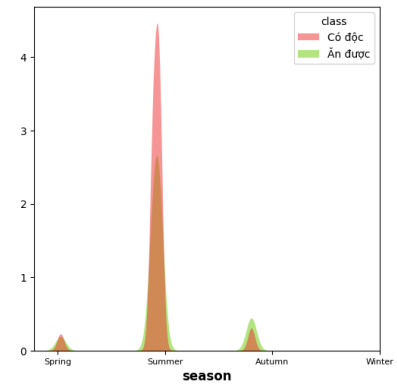




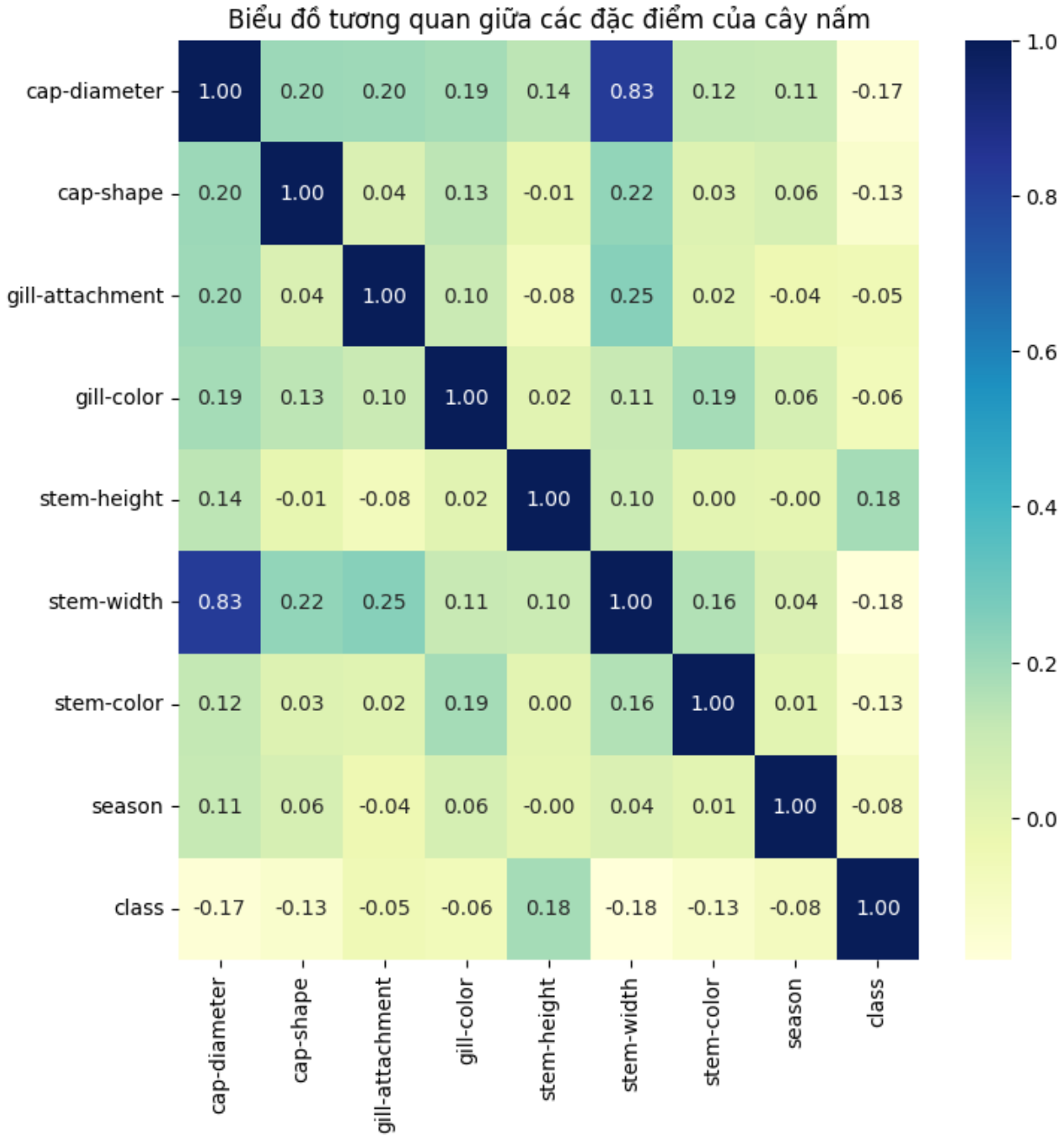








- Tương quan giữa các trường dữ liệu với nhau:



- Pre-process data: Tập dữ liệu chuyển về dạng dataframe, lọc X = features và y = output, sau đó chia thành 3 tập nhỏ: train, valid và test với tỉ lệ 60:20:20.

```
# 60 : 20 : 20
len(X_train), len(X_valid), len(X_test)
```

```
(32421, 10807, 10807)
```

III. Các mô hình sử dụng

Trong phần này, nhóm chúng em sẽ trình bày 6 mô hình chúng em đã thử nghiệm cho bài toán Mushroom edible classification. Với mỗi mô hình, chúng em sẽ trình bày các phần theo thứ tự: Baseline, tối ưu mô hình, diễn giải kết quả, ưu nhược điểm, và so sánh với mô hình trên Kaggle.

1. Decision Tree

1.1. Giới thiệu

Decision tree là thuật toán học máy có giám sát sử dụng cấu trúc cây để đưa ra quyết định dựa trên đặc điểm của đầu vào. Mỗi nút trên cây đại diện cho một thuộc tính của dữ liệu, mỗi nhánh đại diện cho một giá trị của thuộc tính đó.

1.2. Baseline

- Với baseline của decision tree, ta chạy mô hình với các tham số mặc định:

```
# build the baseline model
# create decision tree model object
model = DecisionTreeClassifier()

# train model
model = model.fit(X_train, y_train)

# predict the response for X_test
y_pred = model.predict(X_valid)

# Evaluate model : accuracy
print("Baseline accuracy:", metrics.accuracy_score(y_valid, y_pred))
```

```
Baseline accuracy: 0.9758489867678357
```

- **Baseline accuracy: 97.58 %**
- Một số tham số mặc định của hàm Decision Tree:
    + criterion: gini (hàm tính lỗi khi split)
    + max_depth: None (sẽ chia cây đến khi pure hoặc số sample ở các lá nhỏ hơn min_samples_split)
    + min_samples_split: 1 (số lượng sample tối thiểu để chia 1 node)
    + min_samples_leaf: 1 (số lượng sample tối thiểu ở 1 node lá)

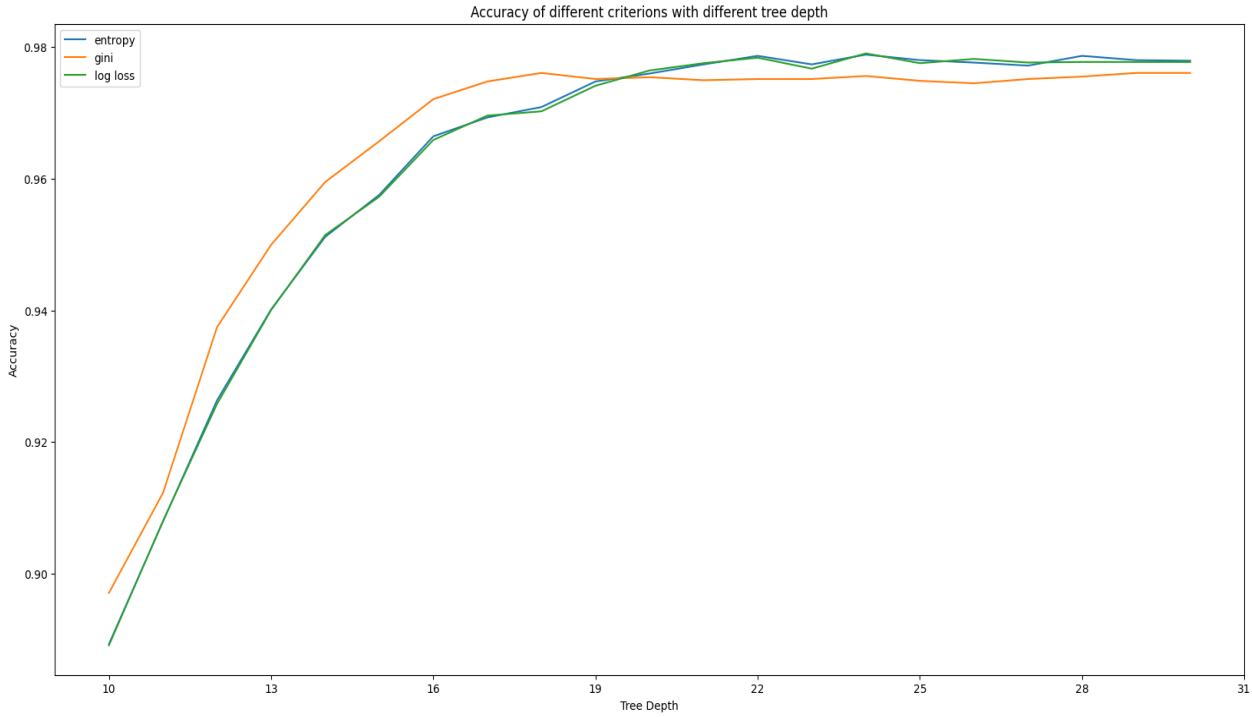
1.3. Optimize model

Với decision tree, để tối ưu hoá mô hình và giảm overfit, một số phương pháp có thể dùng như: early stopping (giảm max_depth), thay đổi split criterion (gini, logloss hoặc entropy), thay đổi stopping criterion (min_samples_split, min_samples_leaf)

- Tuning criterion và max_depth:

  Chúng em đã chạy mô hình với 3 hàm splitting criterion (log loss, entropy, gini) với các độ sâu khác nhau (từ 18 đến 30):

```
def decisionTree(crite):
    acc = []
    for i in range (10, 31):
        model = DecisionTreeClassifier(criterion= crite, max_depth= i)

        # train model
        model = model.fit(X_train, y_train)

        # predict the response for X_valid
        y_pred = model.predict(X_valid)

        # Evaluate model : accuracy
        accuracy = metrics.accuracy_score(y_valid, y_pred)
        acc.append(accuracy)
    return acc

entropy = decisionTree('entropy')
gini = decisionTree('gini')
logLoss = decisionTree('log_loss')
```



Từ đồ thị, ta thấy mô hình với criterion = log loss, max_depth = 24 đạt độ chính xác cao nhất trên tập validation : **97.90 %.** Hai hàm entropy và log loss đạt độ chính xác cao hơn khi độ cao của cây >= 20 so với hàm định gini.

- Tuning hyperparameters by GridSearch :

  Chạy mô hình với từng bộ tham số trong ma trận:

```
# optimal hyperparameters using GridSearchCV
parameter = {
    'criterion' :['entropy','log_loss'],
    'max_depth':[21, 22, 23, 24, 25],
    'min_samples_split' : [1, 2, 3],
    'min_samples_leaf': [1, 2, 3],
}

model = DecisionTreeClassifier()
cv = GridSearchCV(model, param_grid=parameter, cv=5)
cv.fit(X_train, y_train)
✓ 1m 17.3s
```

Ta thu được độ chính xác cao nhất là **97.76 %** với bộ tham số sau:

```
cv.best_params_
✓ 0.0s
{'criterion': 'entropy',
 'max_depth': 23,
 'min_samples_leaf': 1,
 'min_samples_split': 2}
```

- Feature selections: Feature selection là kĩ thuật chỉ lựa chọn 1 phần trong các cột của dữ liệu để training:
    - Filter method: sử dụng variance threshold = 0.1, chỉ lựa chọn các cột có importances > threshold : Độ chính xác đạt được là **95.11 %** trên tập validation.

    - EFS (Exhaustive feature selection) : duyệt toàn bộ tập con của các features. Độ chính xác lớn nhất đạt được là **97 %**, khi ta sử dụng toàn bộ 8 features

```
# using Exhaustive Features Selection

# Import ExhaustiveFeatureSelector from Mlxtend
import mlxtend
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
model = DecisionTreeClassifier(criterion='log_loss', max_depth=24)

efs = EFS(model,
          min_features = 1,
          max_features = 8,
          scoring = 'accuracy',
          print_progress = True,
          cv = 5)
efs.fit(X_train, y_train)

print('Best Accuracy Score: %.2f' % efs.best_score_)
print('Best subset (indices): ', efs.best_idx_)
print('Best subset (corresponding names): ', efs.best_feature_names_)
```

```
Features: 255/255
Best Accuracy Score: 0.97
Best subset (indices):  (0, 1, 2, 3, 4, 5, 6, 7)
Best subset (corresponding names):  ('cap-diameter', 'cap-shape', 'gill-attachment', 'gill-color', 'stem-height', 'stem-width'
```

1.4. Diễn giải kết quả

Mô hình/ Phương pháp tối ưu	Bộ siêu tham số	Accuracy on validation data
Baseline	Default	97.58%
Tuning criterion & max_depth	criterion = log loss max_depth = 24	97.90%
Tuning by Gridsearch	criterion = entropy max_depth = 23 min_samples_leaf = 1 min_samples_split = 2	97.76%
Feature selection using filter method	threshold = 0.1	95.11 %
Feature selection using EFS		97%

Như vậy, với decision tree, độ chính xác cao nhất đạt được trên tập validation là 97.90. Áp dụng mô hình với bộ tham số đạt kết quả cao nhất trên tập valid vào tập test, ta được độ chính xác **98.08%.**

```
model = DecisionTreeClassifier(criterion='log_loss', max_depth=24)

# train model
model = model.fit(X_train, y_train)

# predict the response for X_test
y_pred = model.predict(X_test)

# Evaluate model : accuracy
print("Final accuracy for decision tree method:", metrics.accuracy_score(y_test, y_pred))
```

```
Final accuracy for decision tree method: 0.9808457481262145
```

1.5. Ưu nhược điểm mô hình

- Ưu điểm:
  - Mô hình dễ hiểu, dễ giải thích, cần ít dữ liệu để huấn luyện
  - Có thể xử lý tốt dữ liệu dạng số và dữ liệu hạng mục
  - Xây dựng nhanh, chi phí thấp
- Nhược điểm:

- Không đảm bảo xây dựng được cây tối ưu
- Cây quyết định phụ thuộc rất lớn vào dữ liệu

2. Support vector machine (SVM)

2.1. Giới thiệu

SVM là một thuật toán học máy có giám sát, dựa trên khái niệm siêu phẳng phân lớp, được tạo ra để phân biệt giữa các trường hợp positive và các trường hợp khác, cho phép tối đa hóa. Nó vẽ mỗi điểm dữ liệu trên một không gian n-chiều, trong đó n là số lượng đặc trưng.

2.2. Baseline

Baseline SVM : đầu vào dữ liệu đã được chuẩn hóa theo phương pháp Z-core, kernel = 'linear'.

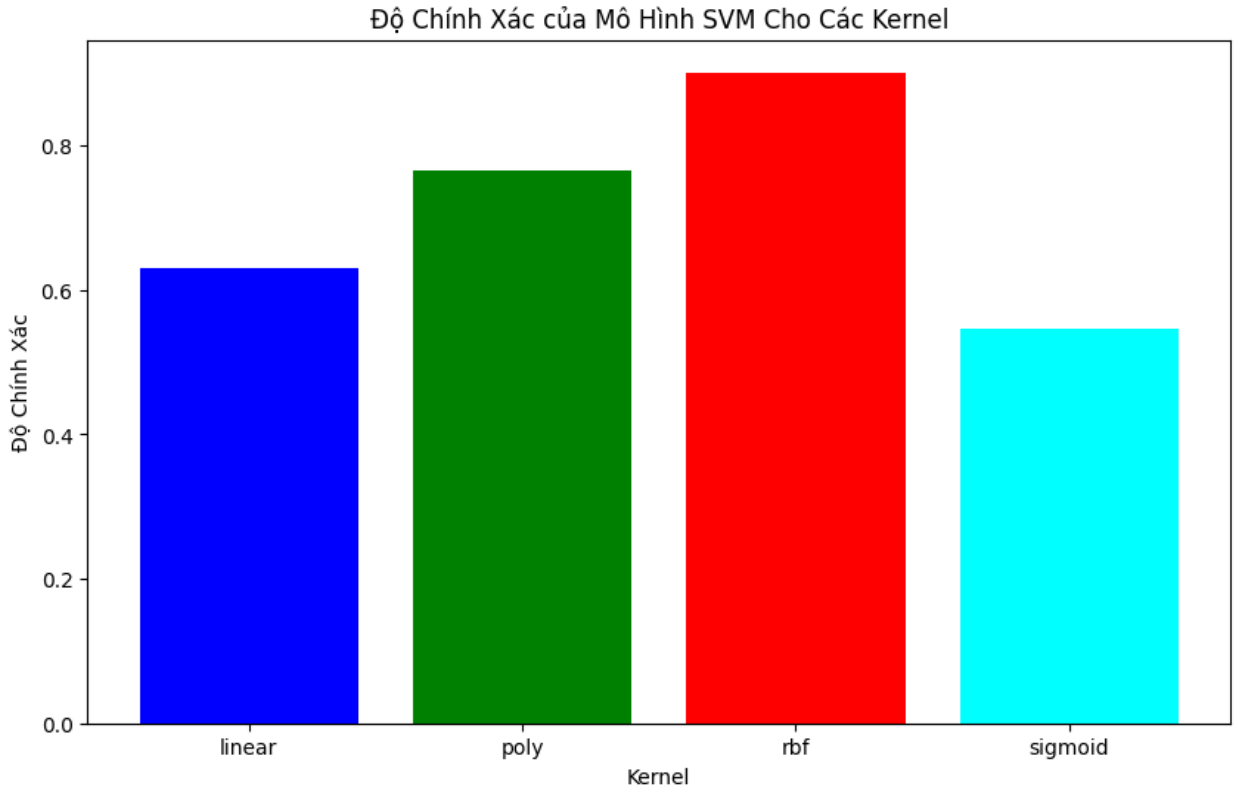
```
# Xây dựng mô hình.
svm_model = SVC(kernel='linear') # Sử dụng kernel tuyến tính tính tích vô hướng.
svm_model.fit(X_train_scaled, y_train)
# Dự đoán với tập dữ liệu kiểm tra.
y_pred_svm = svm_model.predict(X_valid_scaled)
# Đánh giá mô hình.
print("Baseline accuracy:",accuracy_score(y_valid, y_pred_svm))
print(classification_report(y_valid, y_pred_svm))
```

```
C:\Users\nhuyh\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0
  y = column_or_1d(y, warn=True)
Baseline accuracy: 0.6304158502068364
```

Độ chính xác ban đầu tương đối thấp là 0.63.

2.3. Optimize model (trình bày các phương pháp optimize)

Với mô hình SVM, để tối ưu và giảm thiểu overfit một số phương pháp được sử dụng : Chuẩn hóa dữ liệu, thay đổi cách tính tích vô hướng, thay đổi C, thay đổi gamma.



Dựa theo biểu đồ trên là kết quả của 4 cách tính tích vô hướng. kernel = ‘rbf’ thu được độ chính xác cao nhất là 0.90.

Sử dụng phương pháp GridSeachCV để tìm ra bộ tham số tốt nhất.

```
from sklearn.model_selection import GridSearchCV

# xác định phạm vi tham số
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# phù hợp với mô hình để tìm kiếm
grid.fit(X_train_scaled, y_train)
```

Bộ tham số tốt nhất thu được :

```
{'C': 10, 'gamma': 1, 'kernel': 'rbf'}
SVC(C=10, gamma=1)
```

```python
    # Dự đoán trên tập valid
    svm_model_2 = SVC(kernel='rbf',C= 10, gamma=1)
    svm_model_2.fit(X_train_scaled, y_train)
    y_pred_2 = svm_model_2.predict(X_valid_scaled)
    # Đánh giá mô hình.
    print("Accuracy:",accuracy_score(y_valid, y_pred_2))
    print(classification_report(y_valid, y_pred_2))
```

```
C:\Users\nhuyh\AppData\Local\Packages\PythonSoftwareFoundation.Pytho
  y = column_or_1d(y, warn=True)
Accuracy: 0.988569562377531
```

Kết quả dự đoán trên tập valid với siêu tham số tốt nhất là 0.988.

```python
    svm_model_2 = SVC(kernel='rbf',C= 10, gamma=1)
    svm_model_2.fit(X_train_scaled, y_train)
    y_pred_2 = svm_model_2.predict(X_test_scaled)
    # Đánh giá mô hình.
    print("Accuracy:",accuracy_score(y_test, y_pred_2))
    print(classification_report(y_test, y_pred_2))
```

```
C:\Users\nhuyh\AppData\Local\Packages\PythonSoftwareFoundation.Pytho
  y = column_or_1d(y, warn=True)
Accuracy: 0.9916111522329139
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3656
           1       0.99      0.99      0.99      4450

    accuracy                           0.99      8106
   macro avg       0.99      0.99      0.99      8106
weighted avg       0.99      0.99      0.99      8106
```

Kết quả dự đoán trên tập test là 0.991

2.4. Diễn giải kết quả

Phương pháp tối ưu	Bộ tham số	Accuracy on validation data
Baseline	kernel = ‘linear’	0.63
Tính tích vô hướng	kernel = ‘linear’	0.63
	kernel = ‘poly’	0.76
	kernel =’rbf’	0.90
	kernel = ‘sigmoid’	0.54
GridSeachCV	**kernel = ‘rbf’, C = 10, gamma = 1**	**0.988**

Kết quả cuối cùng trên tập test với bộ tham số tốt nhất :

```
Accuracy: 0.9916111522329139
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3656
           1       0.99      0.99      0.99      4450

    accuracy                           0.99      8106
   macro avg       0.99      0.99      0.99      8106
weighted avg       0.99      0.99      0.99      8106
```

2.5. Ưu nhược điểm mô hình

- Ưu điểm
  - Hiệu suất cao trong không gian lớn : SVM hoạt động tốt trên dữ liệu có số chiều lớn và vẫn có hiệu quả ngay cả khi số chiều lớn hơn số mẫu dữ liệu.
  - Ổn định
  - Kỹ thuật Kernel : SVM hỗ trợ các kỹ thuật kernel, giúp chuyển đổi dữ liệu phi tuyến tính thành không gian đặc trưng cao hơn, làm cho nó có thể phân tách các lớp dữ liệu mà không cần phải thực hiện trực tiếp trên dữ liệu gốc.

        - Tránh overfiting : SVM tìm kiếm siêu phẳng tốt nhất để phân tách dữ liệu, nó có xu hướng tránh overfitting
    - Nhược điểm
        - Thời gian huấn luyện dài : SVM mất nhiều thời gian với dữ liệu lớn
        - Tiêu tốn nhiều tài nguyên : Do thuật toán phụ thuộc vào 1 số vector hỗ trợ nên nó tốn nhiều bộ nhớ
        - Khó điều chỉnh tham số : Việc chọn kernel phù hợp hay điều chỉnh siêu tham số phức tạp và mất nhiều thời gian
        - Không hoạt động tốt với dữ liệu nhiều lớp

3. K-Nearest Neighbors (KNN)

3.1. Giới thiệu

Mô hình KNN (K-Nearest Neighbors) là một thuật toán học máy sử dụng trong các bài toán phân loại và hồi quy. Ý tưởng cơ bản của KNN là tìm ra K điểm dữ liệu gần nhất trong tập huấn luyện để dự đoán nhãn hoặc giá trị của một điểm dữ liệu mới dựa trên đa số nhãn của các điểm gần nhất đó.

3.2. Baseline

Baseline của KNN chạy với các tham số mặc định :

```
# Model KNN
knn_base = KNeighborsClassifier()
# Huấn luyện model
knn_base = knn_base.fit(X_train, y_train)
# Dự đoán
y_pred_knn_base = knn_base.predict(X_valid)
# Đánh giá model
print("Baseline accuracy:",accuracy_score(y_valid, y_pred_knn_base))
print(classification_report(y_valid, y_pred_knn_base))
```

```
✓ 0.3s
C:\Users\nhuyh\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\Loca
  return self._fit(X, y)
Baseline accuracy: 0.7111909427389506
```

- **Baseline accuracy : 71.11%**
- Một số tham số mặc định của KNN :
    - n_neighbors=5: Số lượng láng giềng gần nhất được sử dụng để phân loại.
    - `weights='uniform'`: Tất cả các láng giềng đều có trọng số bằng nhau trong quá trình phân loại.

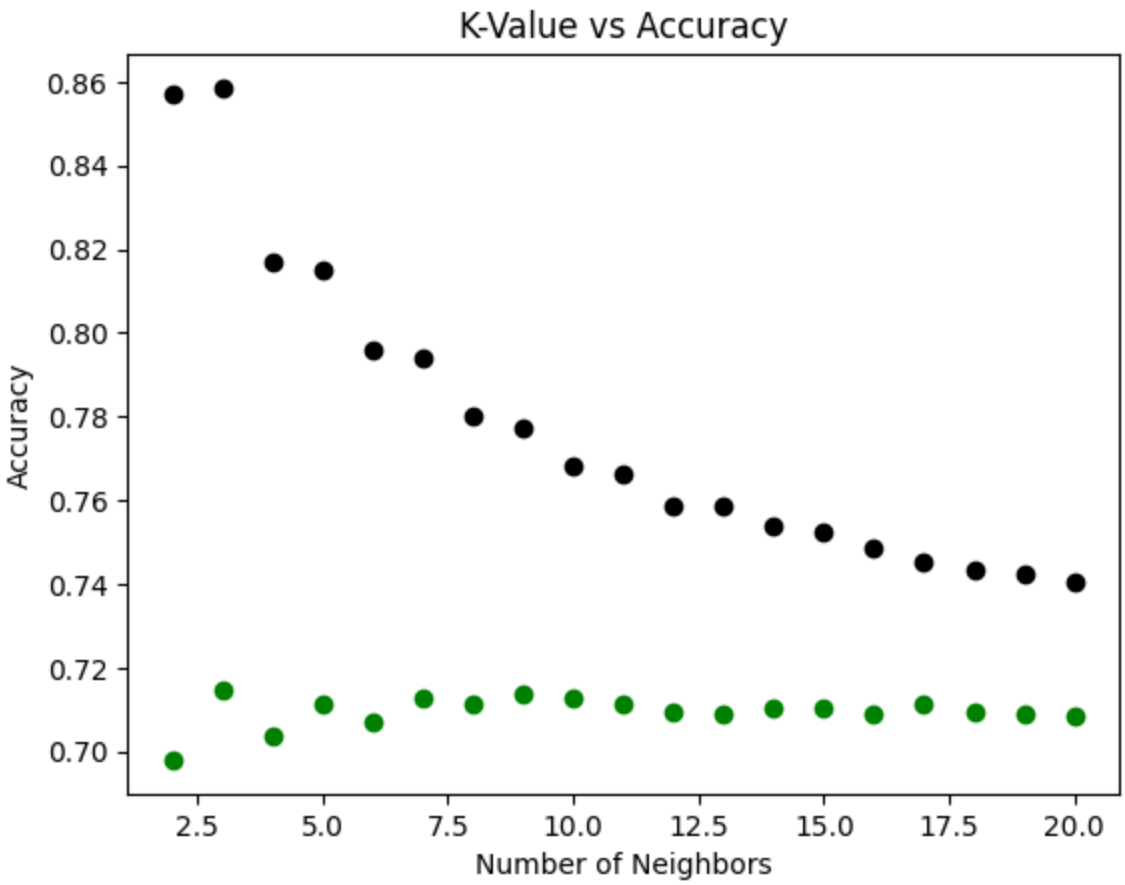
- `leaf_size=30`: Kích thước của các lá trong cây tìm kiếm láng giềng gần nhất (chỉ áp dụng khi `algorithm` là 'ball_tree' hoặc 'kd_tree').
- `p=2`: Tham số quyền lực cho khoảng cách Minkowski.
- `metric='minkowski'`: Tính khoảng cách được sử dụng cho cây tìm kiếm.

3.3. Optimize model (trình bày các phương pháp optimize)

Với mô hình KNN, để tối ưu và giảm thiểu overfit một số phương pháp được sử dụng : thay đổi k, chuẩn hóa dữ liệu (Z-core), thay đổi hàm tính khoảng cách.

- Thay đổi k :
  - Huấn luyện mô hình với k=2 đến 20

```
# Train model k = 2 đến 20
K = []
training = []
test = []
scores = {}
for k in range(2, 21):
    knn_1 = KNeighborsClassifier(n_neighbors = k)
    knn_1.fit(X_train, y_train)

    training_score = knn_1.score(X_train, y_train)
    test_score = knn_1.score(X_valid, y_valid)
    K.append(k)

    training.append(training_score)
    test.append(test_score)
    scores[k] = [training_score, test_score]
```



- Chấm đen là acc trên tập huấn luyện
- Chấm xanh lá là acc trên tập val

- Từ đồ thị, thấy k = 3 mô hình hoạt động tốt nhất với độ chính xác trên tập val là 0.714.

- Chuẩn hóa dữ liệu bằng Z-core

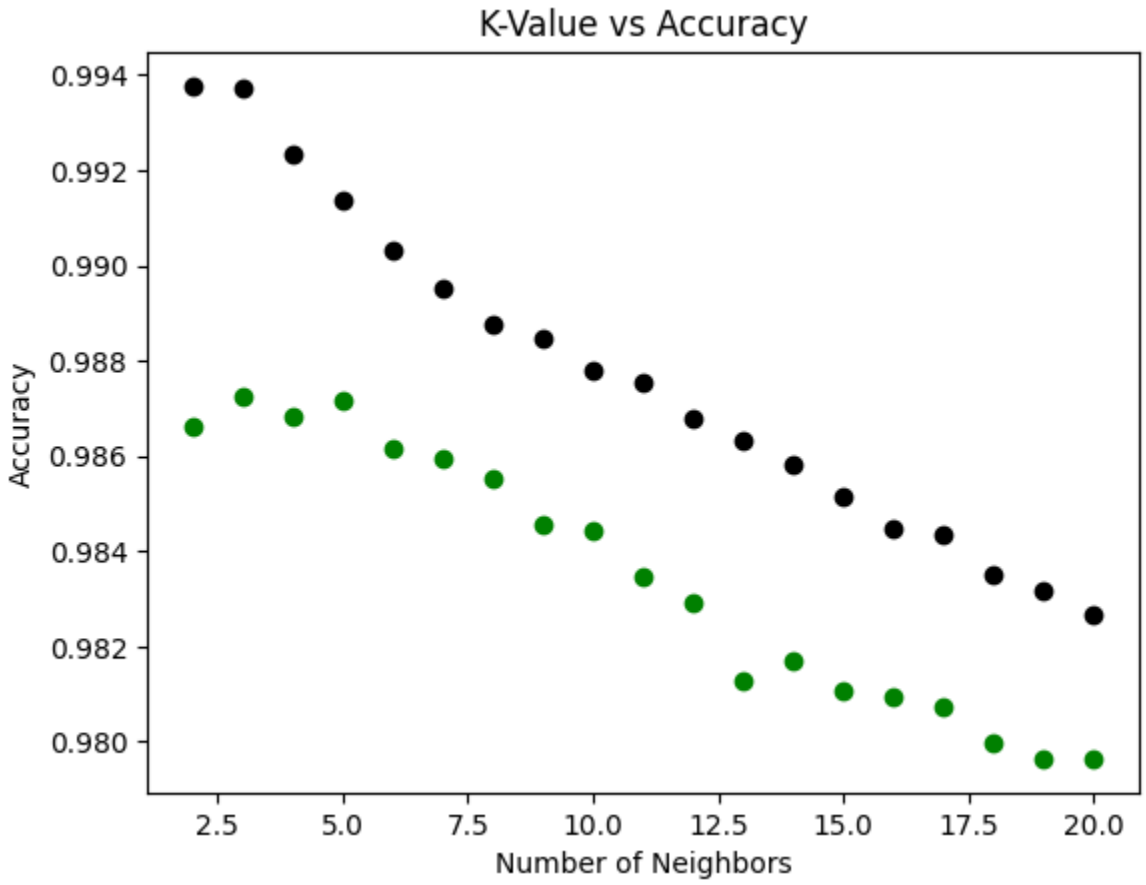
  Z = (X - μ) / σ

  Trong đó:

  - Z : Z-score
  - X : Giá trị cụ thể cần chuẩn hóa
  - μ : Giá trị trung bình của tập dữ liệu
  - σ : Độ lệch chuẩn của tập dữ liệu

  - Sử dụng thư viện StandardScaler của python

```
scaler = StandardScaler()
# Fit dữ liệu và biến đổi nó
X_standardized_train = scaler.fit_transform(X_train)
X_standardized_valid = scaler.transform(X_valid)
X_standardized_test = scaler.transform(X_test)
```



- Từ đồ thì cho thấy kết quả cải thiện tốt, độ chính xác cao nhất với k = 3 với độ chính xác 0.984.

- Thay đổi cách tính hàm tính khoảng cách
  - Hàm manhattan

```
knn_4 = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
knn_4 = knn_4.fit(X_standardized_train, y_train)
y_pred_knn_4 = knn_4.predict(X_standardized_valid)
print("Accuracy:",accuracy_score(y_valid, y_pred_knn_4))
print(classification_report(y_valid, y_pred_knn_4))
```

✓ 0.6s

```
Accuracy: 0.9851948617461355
```

  - Hàm euclidean

```
knn_5 = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_5 = knn_5.fit(X_standardized_train, y_train)
y_pred_knn_5 = knn_5.predict(X_standardized_valid)
print("Accuracy:",accuracy_score(y_valid, y_pred_knn_5))
print(classification_report(y_valid, y_pred_knn_5))
```

✓ 0.6s

```
Accuracy: 0.9845416938819943
```

- Hàm minkowski

```
knn_6 = KNeighborsClassifier(n_neighbors=3, metric='minkowski')
knn_6 = knn_6.fit(X_standardized_train, y_train)
y_pred_knn_6 = knn_6.predict(X_standardized_valid)
print("Accuracy:",accuracy_score(y_valid, y_pred_knn_6))
print(classification_report(y_valid, y_pred_knn_6))
```

✓ 0.6s

```
Accuracy: 0.9845416938819943
```

Vậy, với bộ tham số n_neighbors = 3, metric =’manhattan’ mô hình KNN chạy tốt nhất với độ chính xác là 0.985 trên tập val

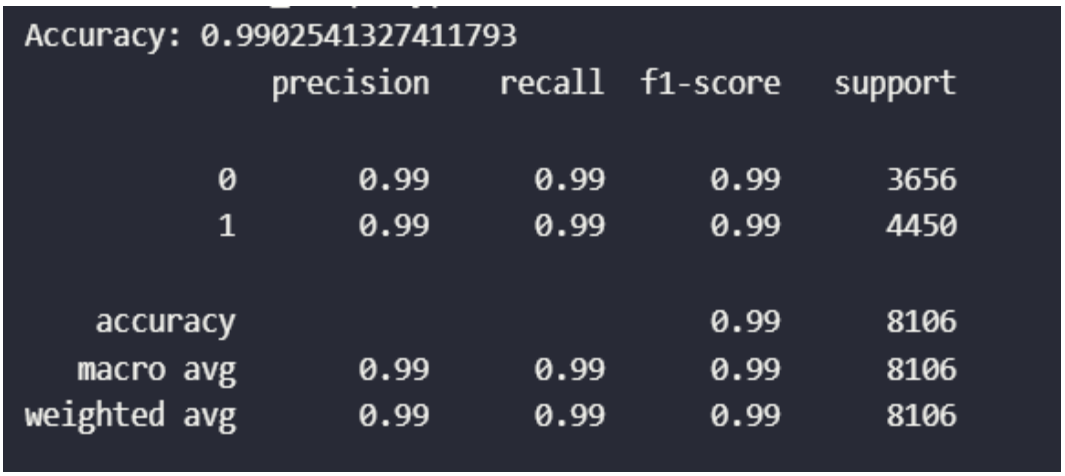
```
Accuracy: 0.9902541327411793
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3656
           1       0.99      0.99      0.99      4450

    accuracy                           0.99      8106
   macro avg       0.99      0.99      0.99      8106
weighted avg       0.99      0.99      0.99      8106
```

Kết quả trên tập test là 0.99

3.4. Diễn giải kết quả

Phương pháp tối ưu	Bộ tham số	Accuracy on validation data
Baseline	Mặc định	0.711
Thay đổi k	k = 3	0.714
Chuẩn hóa tham số	k = 3	0.984
Hàm manhattan	**k = 3, metric= 'manhattan'**	**0.985**
Hàm eculidean	k = 3, metric ='eculidean'	0.984
Hàm minkowski	k = 3, metric ='minkowski'	0.984

Như vậy, mô hình KNN đạt được độ chính xác cao nhất trên tập val là 0.984



Accuracy: 0.9902541327411793

	precision	recall	f1-score	support
0	0.99	0.99	0.99	3656
1	0.99	0.99	0.99	4450
accuracy			0.99	8106
macro avg	0.99	0.99	0.99	8106
weighted avg	0.99	0.99	0.99	8106

Kết quả cuối cùng trên tập test với bộ tham số tốt nhất đạt độ chính xác là 0.99

3.5. Ưu nhược điểm mô hình

- Ưu điểm
  - Mô hình đơn giản, dễ tiếp cận
  - Xử lí tốt với dữ liệu số
  - Độ phức tạp tính toán nhỏ
- Nhược điểm
  - Yêu cầu thời gian tính toán khi phải tính khoảng cách với tất cả điểm trong tập dữ liệu
  - Cần chuyển đổi dữ liệu thành các yếu tố định tính

4. Random forest

4.1. Giới thiệu

Random forest là một thuật toán học máy kết hợp nhiều mô hình 'yếu' để tạo ra một mô hình tổng hợp mạnh mẽ hơn. Mỗi cây trong random forest được huấn luyện trên 1 tập con ngẫu nhiên của dữ liệu, được chọn lại với phép bootstrap, từ đó giảm overfitting bằng cách giới hạn sự tương quan giữa các cây.

4.2. Baseline

- Với baseline của Random forest , ta chạy mô hình với các tham số mặc định:

```
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(x_train, y_train)

# performing predictions on the test dataset
y_pred = clf.predict(x_valid)

# using metrics module for accuracy calculation
print("BASELINE ACCURACY :", metrics.accuracy_score(y_valid, y_pred))
```

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base
  return fit_method(estimator, *args, **kwargs)
BASELINE ACCURACY : 0.988988618488017
```

- **Baseline accuracy: 98.90 %**
- Một số tham số mặc định của model RandomForest:
  - + n_estimators (số lượng cây trong forest): 100
  - + max_depth (độ sâu tối đa của mỗi cây trong forest): None
  - + max_features (số lượng feature khi tìm kiếm best split): sqrt
  - + criterion (splitting criterion mỗi lần chia 1 cây) : gini

4.3. Optimize model

- Tuning hyperparameters by GridSearch:
  Quá trình tuning sử dụng bộ tham số dưới đây:

```
# 1. Use grid search to find the best parameters

param_grid = {
    'n_estimators': [40, 60, 80, 100],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [15, 18, 21],
    'criterion': ['gini', 'entropy', 'log_loss']
}
```

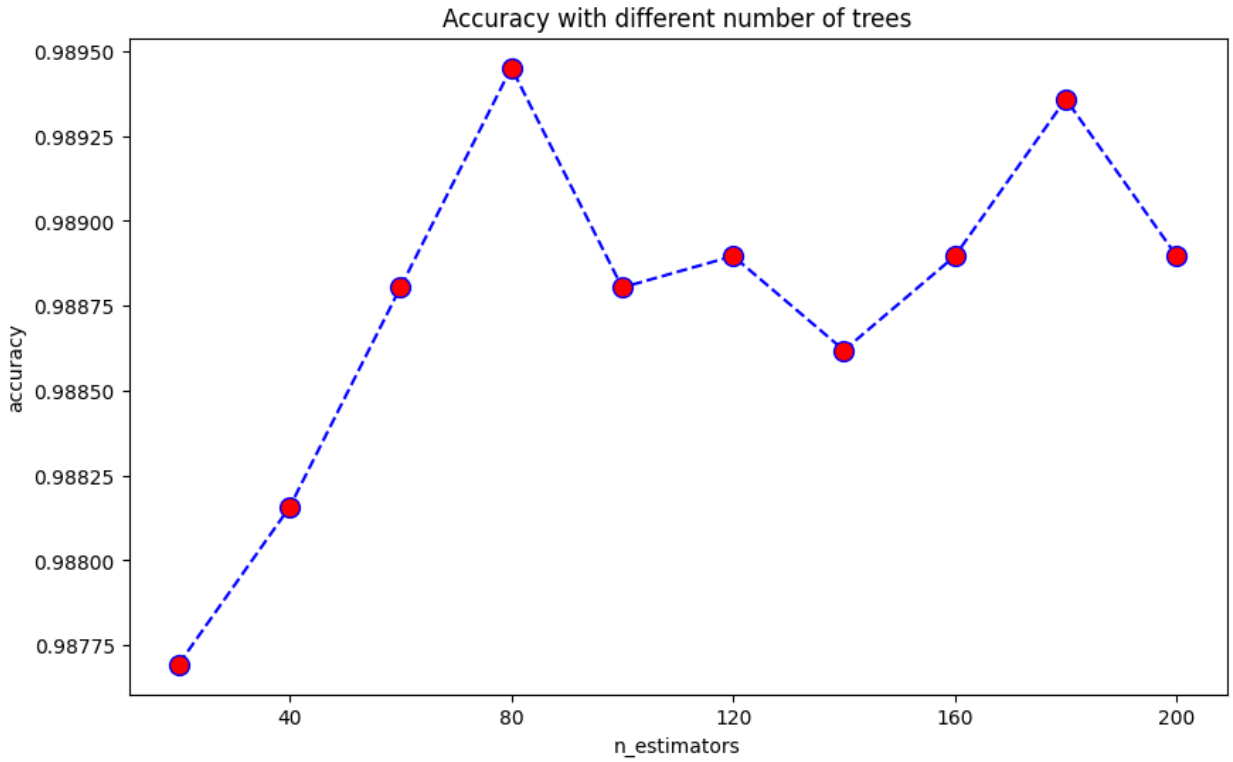
Bộ tham số cho độ chính xác cao nhất **98.89%** trên tập validation là:

```
RandomForestClassifier(criterion='entropy', max_depth=21, max_features='log2',
                       n_estimators=80)
```

- Tuning number of tree (n_estimators):

```
# choose number of trees (n_estimators)
acc = []
for i in range(1, 11):
    model = RandomForestClassifier(n_estimators= 20 * i) # 20 40 60 ... 200
    model.fit(x_train, y_train)
    y_pred = model.predict(x_valid)
    acc.append(metrics.accuracy_score(y_valid, y_pred))
```

Ta có biểu đồ sau:



Từ biểu đồ, ta thấy với n_estimators = 80, độ chính xác trên tập validation đạt giá trị lớn nhất là **98.95%**.

- Error Analysis: Chúng em đã tìm hiểu các dữ liệu dự đoán sai trên tập validation, cụ thể với random forest, có 121 trường hợp dự đoán sai trên tập validation, dưới đây là thông tin về các dữ liệu miss classified:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class	pred
count	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000
mean	648.726496	4.709402	2.153846	9.017094	0.745603	1319.794872	10.034188	0.920267	0.487179	0.512821
std	245.519773	1.884876	2.680116	2.445908	0.599516	513.226954	2.456278	0.119717	0.501985	0.501985
min	136.000000	0.000000	0.000000	1.000000	0.007532	267.000000	1.000000	0.027372	0.000000	0.000000
25%	520.000000	2.000000	0.000000	10.000000	0.333809	1051.000000	11.000000	0.888450	0.000000	0.000000
50%	686.000000	6.000000	1.000000	10.000000	0.609211	1412.000000	11.000000	0.943195	0.000000	1.000000
75%	776.000000	6.000000	6.000000	10.000000	0.887740	1669.000000	11.000000	0.943195	1.000000	1.000000
max	1568.000000	6.000000	6.000000	11.000000	2.259640	2497.000000	11.000000	1.804273	1.000000	1.000000

Tuy nhiên, sau khi vẽ các phân bố từng feature của dữ liệu bị miss và so sánh với phân bố dữ liệu ban đầu, chúng em chưa thấy có nhiều sự khác biệt, chứng tỏ dữ liệu đã được chuẩn hoá tương đối phù hợp với mô hình, kể các trên các outlier, mô hình vẫn dự đoán đúng với tỉ lệ ngang với các phần dữ liệu khác.

4.4. Diễn giải kết quả

Model/ Phương pháp	Bộ tham số	Accuracy on validation data
Baseline	default	98.90%
Tuning by GridSearch	n_estimators = 80 max_depth = 21 criterion = ‘entropy’ max_features = ‘log2’	98.89%
Tuning number of tree	n_estimators = 80	98.95%

Áp dụng bộ mô hình với bộ tham số n_estimators = 80, các siêu tham số khác giữ mặc định vào tập test, ta thu được độ chính xác cuối cùng là **99.16%**

```
model = RandomForestClassifier(n_estimators= 80)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print("Final accuracy: ", metrics.accuracy_score(y_test, y_pred))
```

✓ 8.1s

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python311\Lib\site-packages\s
  return fit_method(estimator, *args, **kwargs)
Final accuracy:  0.9915795317849542
```

4.5. Ưu nhược điểm mô hình

- Ưu điểm:
  - Giảm overfitting so với decision tree nhờ việc kết hợp nhiều cây và sử dụng phương pháp bagging
  - Độ chính xác cao
  - Chống nhiễu tốt hơn decision tree
  - Xử lý tốt dữ liệu số (liên tục và rời rạc) và dữ liệu hạng mục
- Nhược điểm:
  - Yêu cầu tài nguyên tính toán lớn, thời gian huấn luyện lâu hơn so decision tree
  - Việc tối ưu hoá cũng gặp khó khăn hơn thời gian huấn luyện lâu hơn

5. Logistic Regression

5.1. Giới thiệu

Logistic Regression là một thuật toán học máy được sử dụng cho các bài toán phân loại, đặc biệt là phân loại nhị phân (binary classification).

5.2. Baseline

Baseline của Logistic Regression chạy mới tham số mặc định:

```
# Baseline
LR = LogisticRegression()
LR.fit(X_train, y_train)
print(f'Baseline accuracy: {LR.score(X_val, y_val)}')
```

✓ 0.1s

```
Baseline accuracy: 0.635729742726617
```

Một số tham số mặc định:

+ C=1: Tham số này là hệ số nghịch đảo của sức mạnh điều chuẩn.
+ max_iter=100: Số lần lặp tối đa cho các thuật toán tối ưu hóa.
+ penalty=;2: chuẩn được sử dụng.
+ solver=lbfgs: Thuật toán được sử dụng cho tối ưu hóa. Các lựa chọn khác bao gồm 'newton-cg', 'liblinear', 'sag', và 'saga'.
+ class_weight=’None’ được sử dụng trong mô hình Logistic Regression để tự động cân bằng trọng số của các lớp dựa trên tỷ lệ của chúng trong dữ liệu huấn luyện.

5.3. Optimize model

Với mô hình Logistic Regression, tối ưu hóa bằng cách sử dụng regularization và cross validation.

Tìm kiếm các tham số tối ưu bằng GridSearchCV:

```
# Dùng Regularization và cross validation
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga', 'sag'],
    'max_iter': [200, 500, 1000]
}
LR = LogisticRegression(class_weight='balanced')
LR_regu = GridSearchCV(LR, param_grid=param_grid, scoring='accuracy', cv=5)
LR_regu.fit(X_train, y_train)
LR_regu.score(X_val, y_val)
```

Tìm được bộ tham số tốt nhất là:

+ C: 1
+ max_iter: 200
+ penalty: l1
+ solver: liblinear
+ class_weight=’balance’

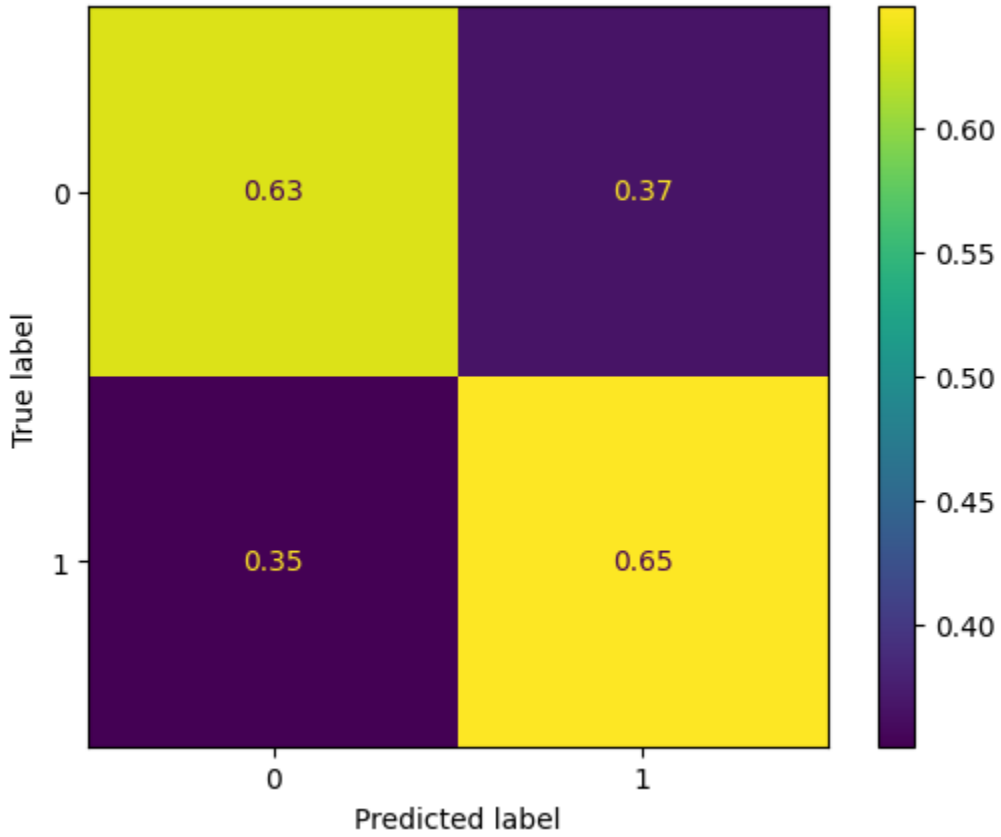
Cải thiện mô hình với accuracy lên đến **64,6%**

5.4. Diễn giải kết quả

Phương pháp tối ưu	Bộ tham số	Độ chính xác trên tập val
Baseline	Mặc định	63,57%
Tìm kiếm tham số tối ưu	C = 1 max_iter = 200 penalty = l1 solver = liblinear class_weight = balance	64,6%

Các chỉ số đánh giá dự đoán trên tập test của mô hình Logistic Regression như sau:

	precision	recall	f1-score	support
0	0.611485	0.633070	0.622090	2153.000000
1	0.669870	0.649251	0.659399	2469.000000
accuracy	0.641714	0.641714	0.641714	0.641714
macro avg	0.640678	0.641160	0.640745	4622.000000
weighted avg	0.642674	0.641714	0.642020	4622.000000



+ Độ chính xác tổng thể của mô hình dự đoán là chính xác là khoảng 64.2%.
+ Precision cho lớp 0 là khoảng 61.1%, cho lớp 1 là khoảng 67% dự đoán đúng trên tổng số mẫu thực tế đúng.
+ Recall cho lớp 0 là khoảng 63.3%, cho lớp 1 là 64,9% dự đoán đúng trên tổng số dự đoán.
+ F1-score cho lớp 0 là khoảng 62.2%, đây là một phép đo tổng thể về hiệu suất của mô hình.

+ Precision, recall và F1-score cho lớp 1 có các giá trị tương tự, không chênh lệch quá lớn.

Có khoảng 2153 mẫu thuộc lớp 0 và 2469 mẫu thuộc lớp 1 trong tập dữ liệu.

5.5. Ưu nhược điểm mô hình

Ưu điểm:

+ Logistic Regression thường làm việc hiệu quả trên các bộ dữ liệu lớn.
+ Đơn giản và dễ triển khai. Điều này làm cho nó trở thành một lựa chọn phổ biến cho các bài toán phân loại cơ bản.

Nhược điểm:

+ Nó không thể mô hình hóa các mối quan hệ phức tạp hoặc phi tuyến tính giữa các biến.
+ Khi tỷ lệ giữa các lớp trong dữ liệu mất cân bằng, mô hình Logistic Regression có thể không hiệu quả và dễ bị lệch về lớp thiểu số.
+ Trong trường hợp dữ liệu có quá nhiều biến hoặc có độ phức tạp cao, mô hình Logistic Regression có thể dễ bị overfitting.

6. Boosting

6.1. Giới thiệu

Boosting là một kỹ thuật trong học máy, được sử dụng để cải thiện hiệu suất dự đoán bằng cách kết hợp nhiều mô hình yếu thành một mô hình mạnh. Các mô hình yếu thường là những mô hình đơn giản, chẳng hạn như cây quyết định, và bản chất của boosting là huấn luyện các mô hình này theo cách mà mỗi mô hình kế tiếp tập trung vào các mẫu mà mô hình trước đó xử lý kém.

6.2. Baseline

```
DT = DecisionTreeClassifier(max_depth=3)
DT_Adaboost = AdaBoostClassifier(DT, random_state=0)
DT_Adaboost.fit(X_train, y_train).score(X_val, y_val)
```

Baseline accuracy: 0,9598

Một số tham số mặc định:

+ n_estimators=50: Số lượng mô hình cơ sở được tạo ra trong quá trình tăng cường.
+ learning_rate=1: Tỷ lệ học của mỗi mô hình cơ sở.
+ random_state=0: Để đảm bảo tính tái lập của kết quả khi chạy mô hình nhiều lần.

6.3. Optimize model

Tìm kiếm tham số tối ưu với GridSearchCV với Adaboost cho mô hình cây quyết định:

```
# AdaBoost với Decision Tree
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1]
}

DT = DecisionTreeClassifier(max_depth=3)
DT_AdaBoost_GS = GridSearchCV(AdaBoostClassifier(DT, random_state=0), param_grid=param_grid)
DT_AdaBoost_GS.fit(X_train, y_train)
```

Accuracy: 0,9836

Với bộ tham số là tối ưu nhất là:

+ learning_rate = 1
+ n_estimators = 200

Gradient Boosting sử dụng gradient descent để tối ưu hóa hàm mục tiêu. Mỗi mô hình mới được thêm vào để giảm thiểu lỗi của mô hình hiện tại bằng cách đi theo hướng gradient của hàm lỗi.

```
# Gradient Boosting với mặc định sử dụng cây quyết định
GB = GradientBoostingClassifier(random_state=0)
GB_grid_search = GridSearchCV(GB, param_grid=GB_param_grid, n_jobs=-1)
GB_grid_search.fit(X_train, y_train)
print(f'Accuracy: {GB_grid_search.score(X_val, y_val)}')
```

✓ 2m 14.8s

```
Accuracy: 0.9884587641259919
```

Accuracy: 0.9884

Với các tham số tối ưu nhất là :

+ learning_rate = 0,1
+ n_estimators = 200
+ max_depth = 7 (độ sâu của cây quyết định)
+ n_jobs=-1: Sử dụng tất cả các lõi CPU sẵn có để thực hiện việc tìm kiếm siêu tham số song song

Sử dụng XGBoost. XGBoost, viết tắt của "Extreme Gradient Boosting," là một thư viện phần mềm mã nguồn mở được sử dụng rộng rãi cho các bài toán học máy, đặc biệt là các bài toán dự đoán và phân loại. XGBoost được phát triển để cung cấp một giải pháp hiệu quả và mạnh mẽ cho việc xây dựng mô hình cây quyết định tăng cường.

```
# XGBoost
import xgboost as xgb

# Sử dụng GPU để huấn luyện mô hình
xgb_model = xgb.XGBClassifier(use_label_encoder=False, tree_method='gpu_hist', eval_metric='logloss', random_state=0)
xgb_grid_search = GridSearchCV(xgb_model, param_grid=GB_param_grid, n_jobs=-1)
xgb_grid_search.fit(
    X_train, y_train,
    early_stopping_rounds=10, # Nếu không cải thiện sau 10 vòng lặp thì dừng
    eval_set=[(X_val, y_val)], # Đánh giá hiệu suất
    verbose=True # Hiển thị quá trình
)
print(f'Accuracy: {xgb_grid_search.score(X_val, y_val)}')
```

Accuracy: 0.9874

Với các tham số tối ưu nhất là:

+ learning_rate = 0,1
+ n_estimators = 200
+ max_depth = 7 (độ sâu của cây quyết định)
+ use_label_encoder=False: Tham số này được sử dụng để vô hiệu hóa việc sử dụng LabelEncoder nội bộ của XGBoost cho các nhãn mục tiêu. Tránh gây ra lỗi khi môi hình này làm việc với dữ liệu phân loại.
+ tree_method='gpu_hist': Sử dụng GPU để tăng tốc độ huấn luyện, phù hợp cho các hệ thống có GPU hỗ trợ.
+ eval_metric='logloss': Để tối ưu hóa mô hình dựa trên hàm mất mát logarit, phù hợp cho các bài toán phân loại nhị phân.
+ early_stopping_rounds=10: Nếu không cải thiện sau 10 vòng lặp thì dừng.

Độ chính xác của XGBoost trên tập test là:

	precision	recall	f1-score	support
0	0.987494	0.990246	0.988868	2153.000000
1	0.991474	0.989064	0.990268	2469.000000
accuracy	0.989615	0.989615	0.989615	0.989615
macro avg	0.989484	0.989655	0.989568	4622.000000
weighted avg	0.989620	0.989615	0.989616	4622.000000

6.4. Diễn giải kết quả

Phương pháp tối ưu	Bộ tham số	Độ chính xác trên tập val
Baseline Adaboost với mô hình là Decision Tree	learning_rate=1 n_estimators=50 max_depth = 3 (Mặc định)	95,98%
Adaboost với mô hình là Decision Tree với bộ tham số tối ưu	learning_rate=1 n_estimators=200 max_depth = 3	98,36%
Gradient Boosting	learning_rate=0,1 n_estimators=200 max_depth = 7 n_jobs=-1	98,84%
XGBoost	learning_rate=0,1 n_estimators=200 max_depth = 7 use_label_encoder=False tree_method='gpu_hist'	98,74%

	eval_metric='logloss' early_stopping_rounds=10	

6.5. Ưu nhược điểm mô hình

	Ưu điểm	Nhược điểm
Adaboost	Đơn giản, dễn triển khai, hiệu quả cho dữ liệu nhiều nhiễu Không yêu cầu điều chỉnh tham số phức tạp	Hiệu suất thấp trên các tập dữ liệu lớn Dễ bị overfitting nếu số lượng cây quá lớn
Gradient Boosting	Hiệu suất rất tốt trên nhiều loại dữ liệu khác nhau. Có thể sử dụng nhiều loại hàm mất mát khác nhau. Có các tham số như số lượng cây, độ sâu cây, và tốc độ học giúp kiểm soát overfitting.	Thường yêu cầu thời gian huấn luyện lớn. Việc tối ưu hóa các siêu tham số có thể phức tạp và tốn thời gian.
XGBoost	Hiệu suất cao. Tối ưu hóa tốc độ và bộ nhớ. Nhiều tính năng nâng cao. Có thể được triển khai trên nhiều máy để xử lý các tập dữ liệu rất lớn.	Cần điều chỉnh nhiều siêu tham số. Khi tập dữ liệu lớn, yêu cầu bộ nhớ và thời gian tính toán. Phức tạp và khó hiểu hơn đối với những người mới bắt đầu.

IV. Kết quả

1.So sánh các mô hình sử dụng

	Recall	Precision	F1 score	Accuracy
Decision Tree	0.98	0.98	0.98	0.9808
Random forest	0.99	0.99	0.99	0.9916
K-Nearest Neighbors	0.99	0.99	0.99	0.9902
Support vector machine	0.99	0.99	0.99	0.9916
Logistic Regression	0.642	0.641	0.642	0.6417
Boosting	0.989	0.989	0.989	0.989

Bảng hiệu suất các mô hình

Random forest,KNN là 2 mô hình học máy hiệu quả nhất trong dự đoán nấm ăn được và nấm độc. Điều này chứng tỏ học máy có thể đóng vai trò quan trọng trong việc phân loại nấm và cải thiện chất lượng an toàn thực phẩm.

2.So sánh với mô hình trên Kaggle

Các project trên Kaggle

Binary Classification using SVM with 87% accuracy (kaggle.com)

Mushroom Classification | %93 Accuracy (kaggle.com)

99% acc Mushroom Data: Different algorithms (kaggle.com)

mushroom_classification_rf (kaggle.com)

Bảng tổng hợp các mô hình trên Kaggle

	Accuracy Kaggle	Accuracy project
SVM	86.4892%	99.16%
K-Nearest Neighbor	92.791%	99.02%

Random Forest	94%	99.16%
Decision Tree	97.8532%	98.08%
Naive Bayes	65.485%	none
Logistic Regression	63.967%	64.17%

Dựa vào bảng biểu trên, chúng ta có thể thấy rằng các mô hình học máy mà nhóm em đã sử dụng mang đến độ chính xác cao hơn hẳn so với các mô hình tương tự trên Kaggle. Điều này cho thấy rằng, những nỗ lực tối ưu và cải thiện từ chúng em đã mang lại hiệu quả như mong đợi.

Một kết quả đáng khích lệ là mô hình SVM, dẫu ở trên Kaggle chỉ đạt độ chính xác là 86.4892%, song mô hình SVM của chúng em đã được cải thiện đáng kể dựa trên cùng một bộ dữ liệu, đạt độ chính xác tới 99.16%.

Tương tự, mô hình K-Nearest Neighbor (KNN) của chúng em cũng đã thực sự hiệu quả hơn khi đạt độ chính xác là 99.02%, trong khi mô hình tương tự trên Kaggle chỉ đạt 92.791%.

Những mô hình học máy còn lại của chúng em cũng đều đạt được sự cải thiện trong độ chính xác, cao hơn từ 1% đến 5% so với các mô hình từ Kaggle.

Nhìn chung, những kết quả này lại khẳng định rằng, việc áp dụng các kỹ thuật và chiến lược tối ưu hóa cho các mô hình học máy đã mang lại những kết quả tích cực, cải thiện đáng kể độ chính xác trong việc phân loại và dự đoán dữ liệu.

V. Kết luận

Trong báo cáo này, chúng em đã thảo luận về ứng dụng của 6 mô hình học máy - K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest, Support Vector Machine (SVM), Logistic Regression, và Boosting - trong việc phân loại nấm ăn được và không ăn được.

Các mô hình học máy này đã đem lại dự đoán chính xác và hữu ích, hỗ trợ trong việc phân biệt các loại nấm và giúp những người tiêu dùng, người trồng nấm, và các nhà nghiên cứu trong lĩnh vực này. Tuy nhiên, việc áp dụng chúng yêu cầu sự cân nhắc và chuẩn bị cẩn thận, đặc biệt là trong việc thu thập và xử lý dữ liệu.

Dù còn một số thách thức, nhưng tiềm năng của trí tuệ nhân tạo và học máy trong việc phân loại nấm và cải thiện chất lượng và an toàn thực phẩm rất đáng kể và đáng để nghiên cứu tiếp.

Nhìn chung, kết quả của dự án nhấn mạnh sức mạnh của việc ứng dụng học máy trong việc giải quyết các vấn đề phân loại phức tạp và đóng góp vào sự phát triển bền vững và an toàn của cộng đồng.

Trong quá trình làm project, dù đã cố gắng nhưng chúng em không thể tránh khỏi những thiếu sót, những phần chưa hợp lý, chúng em rất mong nhận được sự đóng góp và nhận xét của thầy để bài làm của chúng em được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!