ĐẠI HỌC QUỐC GIA TP. HCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

----------○★○----------

GIẢNG VIÊN HƯỚNG DẪN:

TS Nguyễn Vinh Tiệp

Đồ án môn học: Lập trình Python cho máy học – CS116.M12.KHCL

K-Nearest Neighbors

195220572 –Trần Hồ Thiên Phước

19522295 - Trương Thị Kim Thoa

19522309 -Nguyễn Việt Thư

Thành phố Hồ Chí Minh, Năm 2021

MỤC LỤC

DANH MỤC HÌNH ẢNH

Chương 1

GIỚI THIỆU CHUNG

1.1. Giới thiệu

- K-nearest neighbors (KNN) là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Thuật toán KNN giả định sự giống nhau giữa dữ liệu mới với các dữ liệu có sẵn và phân loại dữ liệu mới vào labels giống nhất với các dữ liệu có sẵn. Tính toán chỉ được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới.

Hình 1.1. Minh họa bài toán phân loại

- KNN có thể sử dụng cho cả bài toán Classification và Regression, nhưng thường được sử dụng nhất cho bài toán Classification. Khi training, thuật toán này không có giai đoạn huấn luyện chuyên biệt thay vào đó nó sử dụng tất cả dữ liệu để huấn luyện trong khi phân loại (đây cũng là lý do thuật toán này xếp vào loại lazy learning). KNN là thuật toán phi tham số có nghĩa là nó không đưa ra bất kỳ giả định nào về dữ liệu.

Chương 2

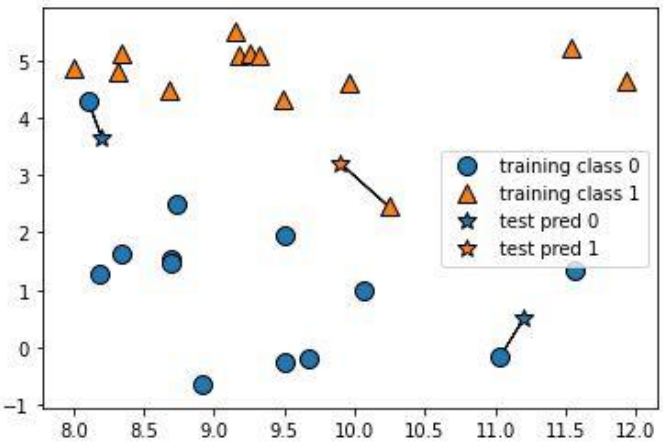
PHÂN TÍCH THUẬT TOÁN

2.1. Cách thức hoạt động

- Hoạt động của KNN có thể mô tả đơn giản trong các bước sau:
  - Bước 1: Chọn số lượng K cho mô hình.
  - Bước 2: Tính toán khoảng cách của K láng giềng gần nhất.
  - Bước 3: Trong số K lân cận này, đếm số điểm dữ liệu trong mỗi loại.
  - Bước 4: Gán labels các điểm dữ liệu mới theo labels của hàng xóm có số lượng lớn nhất.
- K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression.

2.1.1. K-neighbors Classification

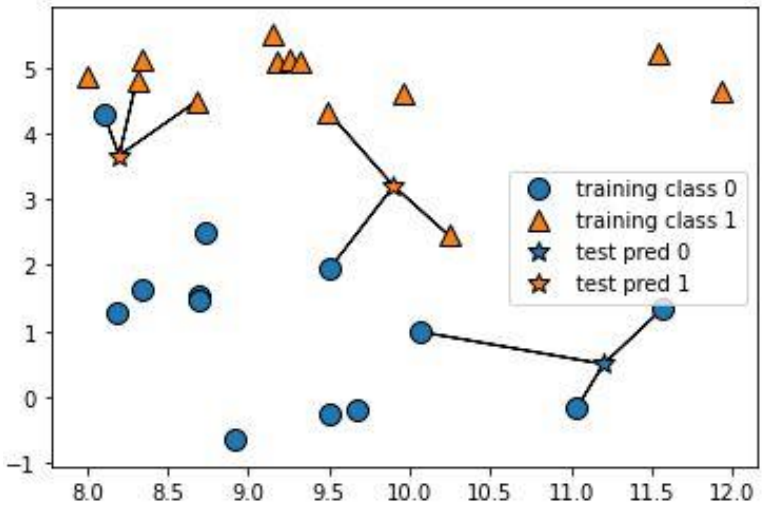
- Trong bài toán Classification, label của mỗi điểm dữ liệu trong tập test được suy ra từ K điểm dữ liệu gần nhất trong tập train.



Hình 2.1 Dự đoán Classification trên fore dataset mglearn với K=1

(Ở đây ta thêm 3 điểm data mới (biểu diễn bằng hình ngôi sao). Ở mỗi điểm test, tìm điểm gần nó nhất trong tập train, label của điểm test chính là label của điểm train gần nó nhất (biểu diễn bằng màu))

- Khi K>1, chúng ta sử dụng *voting* giữa các điểm gần nhất để quyết định labels cho tập test. Điều này có nghĩa là ở mỗi điểm test, ta đếm có bao nhiêu điểm trong tập train gần nó nhất thuộc labels nào, labels có số lượng lớn nhất thì labels đó sẽ chính là labels cho điểm test đó.

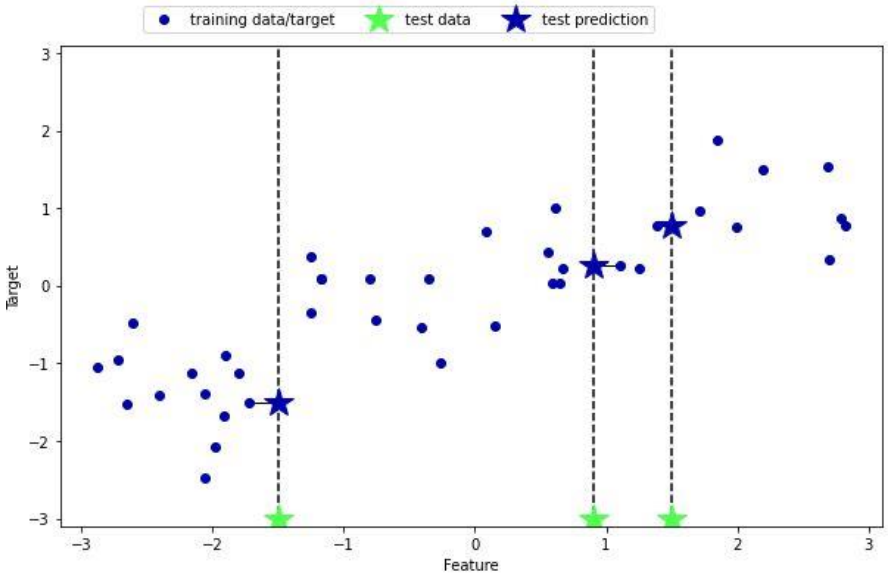


Hình 2.2. Dự đoán classification trên fore dataset mglearn với K=3

(Ở đây ta có thể thấy với các điểm data mới như vậy nhưng dự đoán label ở điểm test ở vị trí trái trên đã thay đổi so với dự đoán lúc K=1)

2.1.2. K-neighbors Regression

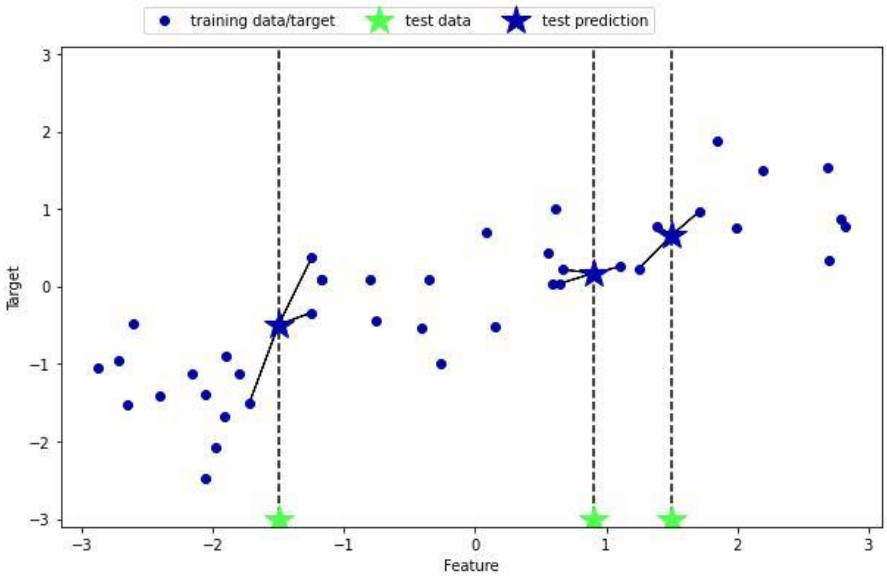
- Trong bài toán Regression, đầu ra của một điểm dữ liệu test sẽ bằng chính đầu ra của điểm dữ liệu gần nhất khi K = 1.



Hình 2.3. Dự đoán regression trên wave dataset mglearn với K=1

(Thêm 3 điểm data mới minh họa là các ngôi sao màu xanh trên trục x, giá trị dự đoán là giá trị mục tiêu gần nhất (biểu diễn bằng ngôi sao màu xanh))

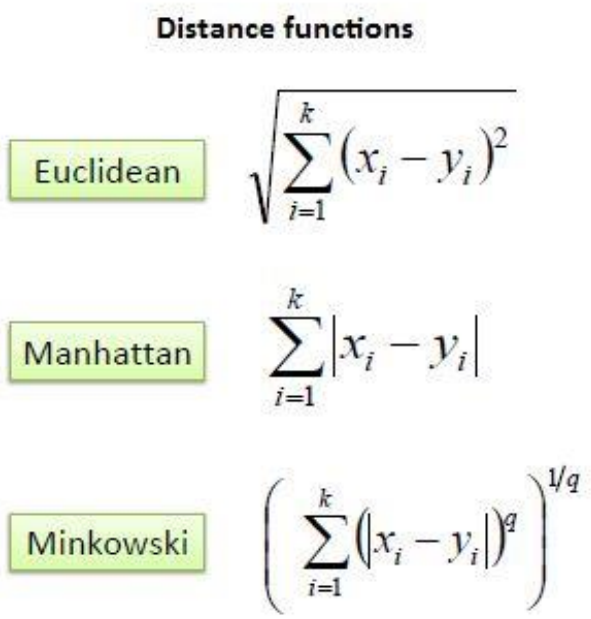
- Ta có thể sử dụng K>1 trong trường hợp này giá trị dự đoán có thể là trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.



Hình 2.4. Dự đoán regression trên wave dataset mglearn với K=3

2.2. Cơ sở toán học

- Trong không gian một chiều, khoảng cách giữa hai điểm là trị tuyệt đối giữa hiệu giá trị của hai điểm đó. Trong không gian nhiều chiều, khoảng cách giữa hai điểm có thể được định nghĩa bằng nhiều hàm số khác nhau.
- Việc tìm khoảng cách giữa 2 điểm trong KNN cũng có nhiều công thức có thể sử dụng, tùy trường hợp mà chúng ta lựa chọn cho phù hợp. Đây là 3 cách cơ bản để tính khoảng cách 2 điểm dữ liệu x, y có k thuộc tính.



Hình 2.5. Các công thức tính khoảng cách thông dụng

2.3. Phân tích các tham số của mô hình trong sklearn

2.3.1. n_neighbors(int), default = 5

- Số neighbors sử dụng cho truy vấn, tìm kiếm. Việc lựa chọn số neighbors là vô cùng quan trọng. Trong trường hợp số phân lớp labels là chẵng ta nên chọn K lẻ và ngược lại.
- K quá nhỏ sẽ chịu ảnh hưởng rất lớn nếu dữ liệu nhiễu.
- K quá lớn dẫn đến việc phân lớp bị sai.

2.3.2. weights{‘uniform’,’distance’} or callable, default = ‘uniform’

- Weight được dùng trong việc phân chia tầm quan trọng của các điểm neighbors. Nhìn chung KNN tuân theo quy tắc major voting. Nghĩa là chọn lớp số đông theo bầu cử. Bao gồm:
    - ‘uniform’: trọng lượng đồng đều. Tất cả các điểm neighbors được đánh giá ngang nhau về tầm quan trọng khi voting.

- ‘distance’: trọng lượng của điểm bằng nghịch đảo khoảng cách. Trong trường hợp này, những điểm nighbors gần hơn của một điểm truy vấn sẽ có ảnh hưởng lớn hơn những điểm neighbors ở xa hơn. Đánh giá công bằng hơn cho bài toán.

- [callable]: một hàm do người dùng định nghĩa chấp nhận một mảng khoảng cách và trả về một mảng có cùng hình dạng chứa các trọng số.

2.3.3. algorithm{‘auto’, ’ball_tree’, ’kd_tree’, ‘brute’}, default=’auto’

- Thuật toán được sử dụng để tính các hàng xóm gần nhất:
  - ‘ball_tree’.
  - ‘kd_tree’.
  - ‘brute’ : vét cạn tính khoảng cách giữa các cặp điểm.
  - ‘auto’ : sẽ cố gắng quyết định thuật toán thích hợp nhất dựa trên các giá trị được truyền cho phương thức phù hợp.

2.3.4. leaf_size(int), default = 30

- Kích thước lá được dùng cho BallTree hoặc KdTree. Điều này có thể ảnh hưởng đến tốc độ xây dựng và truy vấn, cũng như bộ nhớ cần thiết để lưu trữ cây. Giá trị tối ưu phụ thuộc vào bản chất của vấn đề.

2.3.5. p(int), default = 2

- Tham số công suất cho chỉ số Minkowski (norm). Khi p=1 (norm 1) điều này tương đương với việc sử dụng manhattan_distance và Euclidean_distance cho p=2 (norm 2). Đối với p tùy ý minkowski_distance được sử dụng.

2.3.6. metric(str) or callable, default = ‘minkowski’

- Ma trận khoảng cách sẽ sử dụng cho mô hình. Số liệu mặc định là minkowskin và p=2 tương đương với số liệu Euclidean tiêu chuẩn. Nếu chỉ số được ‘precomputed’, X được giả định là ma trận khoảng cách và phải là hình vuông trong khi điều chỉnh. X có thể là một đồ thị thưa thớt trong trường hợp đó, chỉ các phần tử “khác không” mới có thể được coi là neighbor.

2.3.7. metric_paramsdict, default=None

- Bổ sung keyword cho metric ở trên.

2.3.8. n_jobs(int), default=None

- Số lượng công việc song song để chạy tìm kiếm neighbors. Không có nghĩa là 1 trừ khi trong ngữ cách joblib.paralled_backend. -1 có nghĩa là sử dụng tất cả các bộ xử lý. Nó không hề ảnh hưởng đến phương pháp phù hợp.

2.4. Cách điều chỉnh tham số cho mô hình

- Các mô hình học máy khác nhau có các siêu tham số khác nhau, việc lựa chọn điều chỉnh như thế nào có thể tùy thuộc vào bài toán và người làm. Để điều chỉnh tham số phù hợp thông thường cách đơn giản là cho chạy thử mô hình trên các tham số khác nhau và chọn ra tham số cho kết quả tốt nhất.
- Nhưng nhìn chung, đối với KNN có 2 siêu tham số rất quan trong đó là n_neighbors (số lượng neighbors K) và phương thức tính khoảng cách metric.
- Để điều chỉnh tham số ta có thể làm bằng nhiều phương pháp khác nhau như Bo, Gridsearch….

Chương 3

ĐÁNH GIÁ MỞ RỘNG

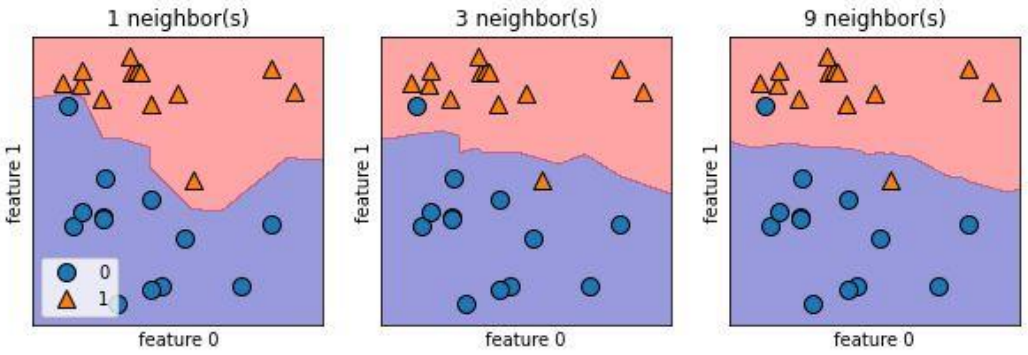
3.1. Phân tích mở rộng

3.1.1. KneighborsClassification

- Ví dụ visualize ranh giới quyết định cho forge dataset trong mglearn cho 1, 3 , 9 neighbors (K).
- Code

```
X,y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1,3, figsize=(10,3))

for n_neighbors, ax in zip([1,3,9], axes):
      clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit
(X,y)
      mglearn.plots.plot_2d_separator(clf,X, fill=True, eps =
 0.5,ax=ax,alpha=.4)
      mglearn.discrete_scatter(X[:,0],X[:,1],y,ax=ax)
      ax.set_title("{} neighbor(s)".format(n_neighbors))
      ax.set_xlabel("feature 0")
      ax.set_ylabel("feature 1")
      axes[0].legend(loc=3)
```



Hình 3.1 Ranh giới quyết định tạo bởi Knn model cho K khác nhau

- Như ta có thể thấy với mô hình đầu tiên sử dụng K=1 dẫn đến decision boundary (ranh giới quyết định) theo sát với tập dữ liệu huấn luyện. Khi K càng

lớn thì ranh giới càng mượt hơn (smooth). Một ranh giới smoother tương ứng với một mô hình đơn giản hơn. Nói cách khác, K càng nhỏ tương ứng với mô hình càng phức tạp, K càng lớn thì mô hình càng đơn giản. Nếu xem xét trong trường hợp cực đoan nhất là K bằng số lượng điểm có trong train dataset thì mỗi điển test sẽ chính xác giống neighbors (điểm training) và tất cả dự đoán sẽ giống nhau: lớp nhiều nhất trong tập huấn luyện.

- Để chứng minh nhận định trên và tính tổng quát, ta sẽ xem xét trên bộ dữ liệu thực tế Breast Cancer. Chúng ta chia bộ dữ liệu thành tập train và test và bắt đầu đánh giá với số lượng neighbors khác nhau.
- Code

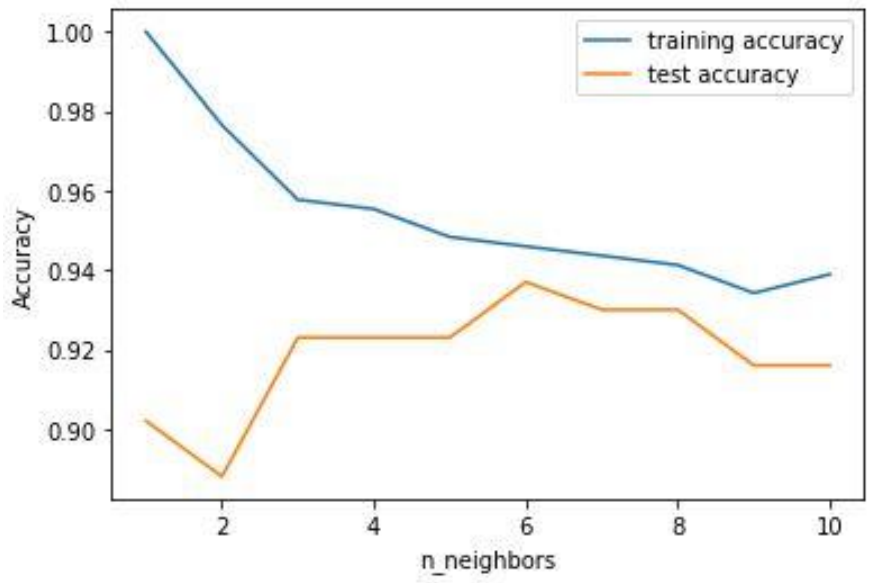
```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify = cancer.target, random_
state=66)
training_accuracy = []
test_accuracy = []

#try n_neighbors from 1 to 10
neighbors_setting = range(1,11)

for n_neighbors in neighbors_setting:
  #build model
  clf = KNeighborsClassifier(n_neighbors=n_neighbors)
  clf.fit(X_train,y_train)
  #record training set accuracy
  training_accuracy.append(clf.score(X_train, y_train))
  #record generalization accuracy
  test_accuracy.append(clf.score(X_test,y_test))

plt.plot(neighbors_setting, training_accuracy, label="training
accuracy")
plt.plot(neighbors_setting, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```



Hình 3.2. So sánh accuracy của tập train và test trên các n_neighbors bằng KNN classfication

(Đồ thị minh họa accuracy tập train và tập test trên trục y và n_neighbors trên trục x)

- Trong thế giới thực đồ thị hiếm khi mượt (smooth), chúng ta vẫn có thể thấy một số đặc điểm bị overfitting hoặc underfitting vì khi K càng nhỏ thì mô hình càng phức tạp. Ta có thể thấy với K=1, dự đoán trên tập train rất tốt, nhưng khi số lượng neighbors càng lớn mô hình càng đơn giản dẫn đến accuracy của tập train có xu hướng giảm. Mặc khác, dự đoán trên tập test cho accuracy với K=1 thấp hơn khi so với số lượng neighbors càng lớn, điều này đồng nghĩa với khi K càng thấp mô hình càng phức tạp.

3.1.2. KneighborsRegression

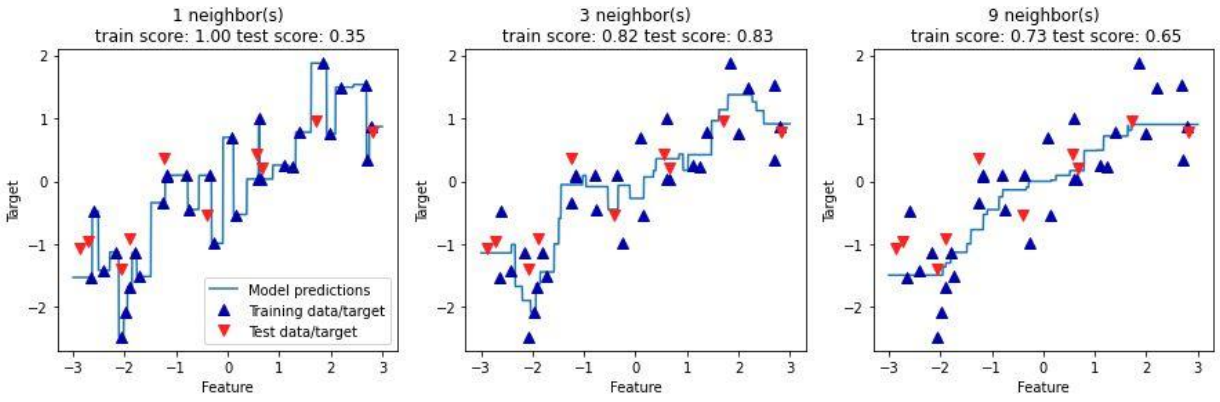
- Ví dụ visualize đồ thị điểm cho wage dataset trong mglearn cho 1, 3 , 9 neighbors (K).
- Code

```
X,y = mglearn.datasets.make_wave(n_samples=40)
X_train,X_test,y_train, y_test = train_test_split(X,y,random_stat
e=0)

fig, axes = plt.subplots(1,3,figsize=(15,4))
```

```
# create 1000 data points, evenly spaced between -3 and 3
line = np.linspace(-3,3,1000).reshape(-1,1)
for n_neighbors, ax in zip([1,3,9], axes):
  #make predictions using 1,3 or 9 neighbors
  reg = KNeighborsRegressor(n_neighbors=n_neighbors)
  reg.fit(X_train,y_train)
  ax.plot(line,reg.predict(line))
  ax.plot(X_train,y_train,'^', c=mglearn.cm2(0), markersize=8)
  ax.plot(X_test,y_test,'v', c=mglearn.cm2(1), markersize=8)

  ax.set_title("{} neighbor(s)\n train score: {:.2f} test score:
{:.2f}".format(
      n_neighbors, reg.score(X_train, y_train), reg.score(X_test,
y_test)))
  ax.set_xlabel("Feature")
  ax.set_ylabel("Target")
axes[0].legend(["Model predictions", "Training data/target",
                "Test data/target"], loc="best")
```



Hình 3.3. So sánh dự đoán cho n_neighsbor khác nhau bằng KNN regression

- Có thể thấy từ đồ thị, với K=1 mỗi điểm trong tập train có ảnh hưởng rõ ràng đối với sự dự đoán và giá trị dự đoán đi qua tất cả các điểm dữ liệu. Điều này dẫn đến việc dự đoán không ổn định. Khi số lượng neighbors càng lớn dẫn đến việc dự đoán mượt hơn nhưng điều này lại không fit train data.

3.2. Ưu điểm và khuyết điểm của KNN

Qua tìm hiểu và phân tích mô hình KNN ta rút ra được các ưu điểm và khuyết điểm.

3.1.1. Ưu điểm

- Độ phức tạp tính toán cho quá trình training là bằng 0. Nó không học được gì trong quá trình training. Nó lưu trữ tập dự liệu huấn luyện và chỉ học tại thời điểm nó dự đoán.

  => KNN nhanh hơn nhiều so với các thuật toán khác yêu cầu training trước.
- Vì thuật toán KNN không cần huấn luyện trước khi đưa ra dự đoán nên dữ liệu mới có thể được thêm liền mạch sẽ không ảnh hưởng đến độ chính xác của thuật toán.
- KNN rất dễ hiểu và dễ thực hiện. Chỉ có hai tham số quan trọng yêu cầu để triển khai, tức là giá trị của K và hàm khoảng cách.

  => Dễ tối ưu tham số cho mô hình.

3.1.2. Nhược điểm

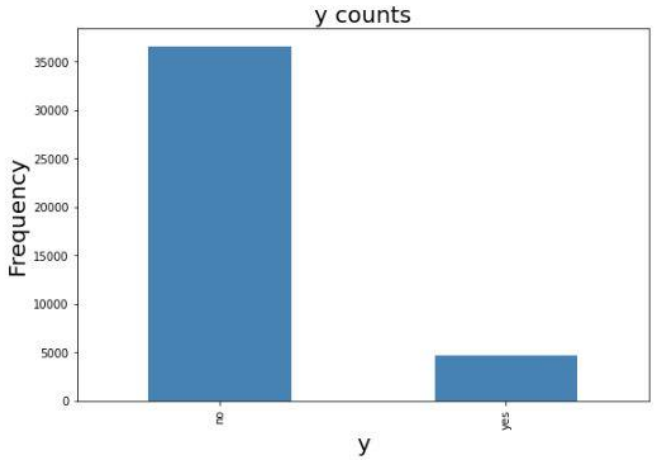
- Không hoạt động tốt với tập dữ liệu lớn, trong tập dữ liệu lớn, chi phí tính toán khoảng cách giữa điểm mới và mỗi điểm hiện có là rất lớn, điều này làm giảm hiệu suất của thuật toán.
- Không hoạt động tốt với bộ dữ liệu phức tạp (số chiều lớn), thuật toán sẽ trở nên khó khăn trong việc tính toán khoảng cách trong mỗi chiều.
- Cần scaling đặc trưng, cần thực hiện standardization và normalization trước khi áp dụng thuật toán KNN cho bất kỳ tập dữ liệu nào. Nếu không làm như vậy, KNN có thể tạo ra kết quả dự đoán sai.
- Nhạy cảm với dữ liệu nhiễu.

Chương 4

THỰC NGHIỆM

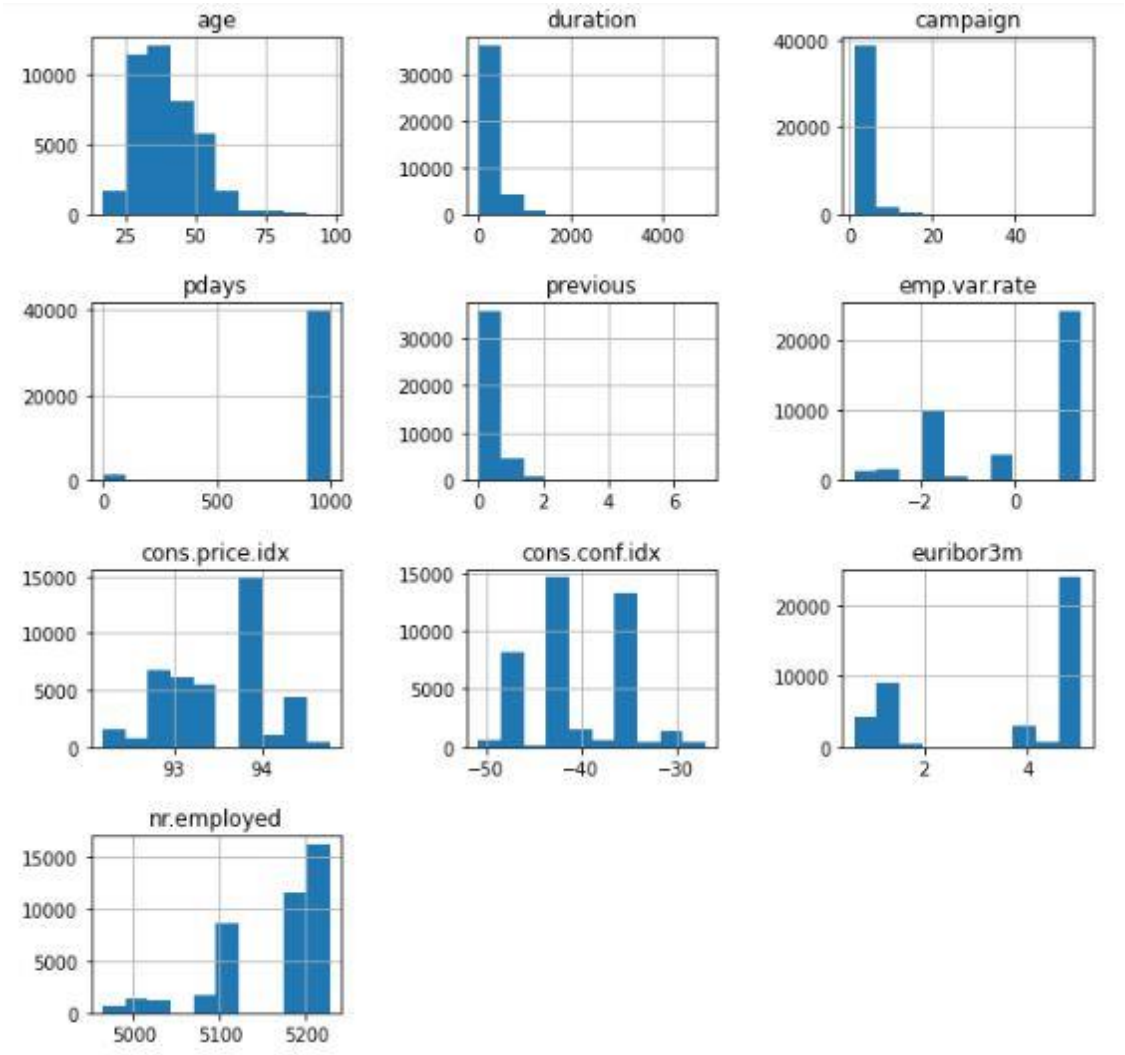
4.1. Mô tả dữ liệu:

- Data được lấy từ chiến dịch tiếp thị trực tiếp của Ngân hàng Bồ Đào Nha, là những cuộc gọi trực tiếp để xác nhận xem khác hàng của họ có đăng kí gửi tiền có kì hạn hay không (yes/no).
- Dữ liệu được bổ sung thêm 5 đặc điểm/ thuộc tính kinh tế và xã hội mới (các chỉ số trên phạm vi toàn quốc từ một quốc gia có dân số ~ 10 triệu người), được xuất bản bởi Banco de Portugal và công khai tại.
  https://www.bportugal.pt/estatisticasweb. Tạo bởi: Sérgio Moro (ISCTE-IUL), Paulo Cortez (Univ. Minho) và Paulo Rita (ISCTE-IUL) năm 2014.
- Records: 41188 cho bản dataset đầy đủ này.
- Số thuộc tính: 20+ thuộc tính.
- Output: 2 trường dữ liệu Yes/No.



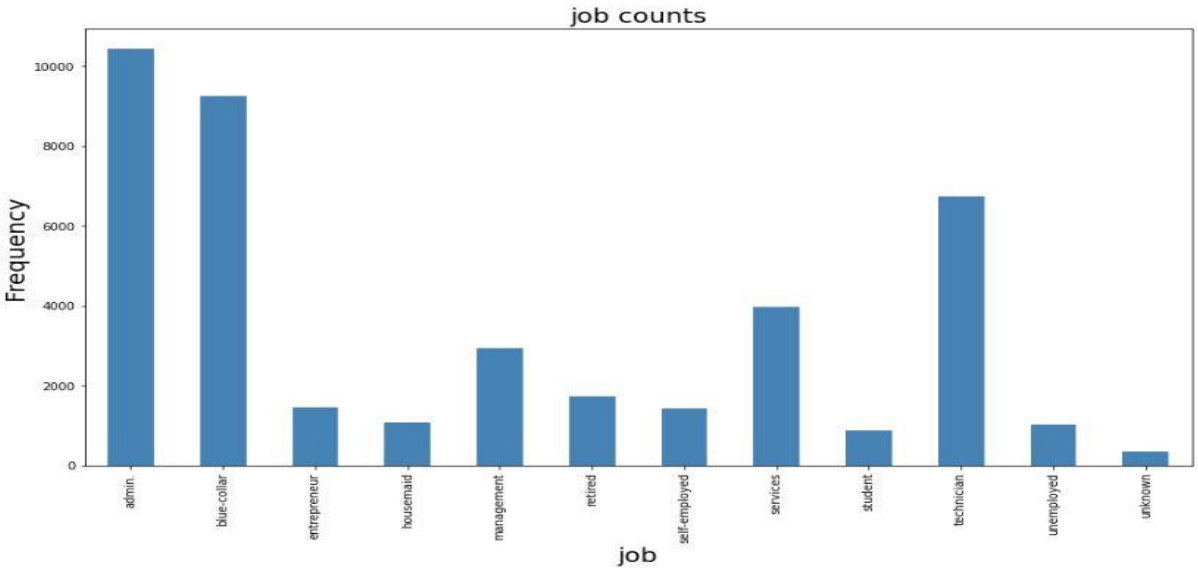
Hình 4.1. Phân bố phân loại của kết quả đầu ra

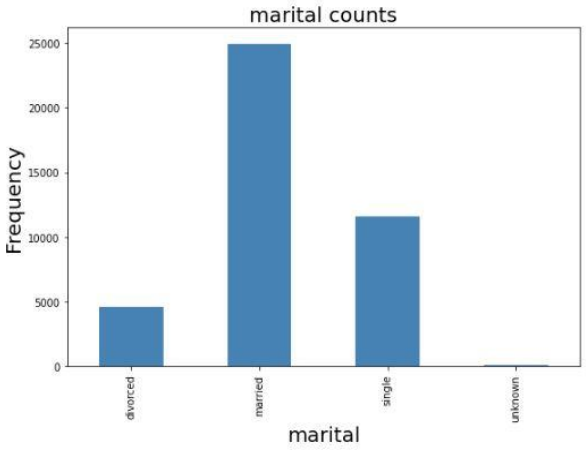
- **Sự phân bố của các trường thuộc tính liên tục**

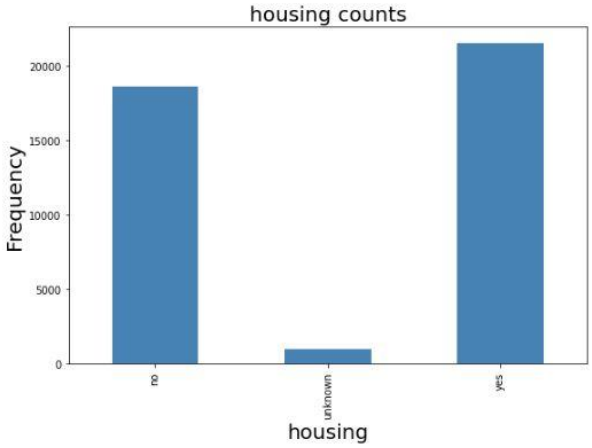


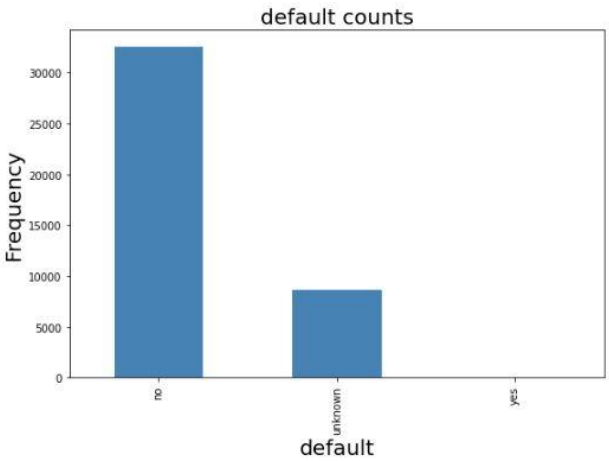
Hình 4.2. Phân bố dữ liệu thuộc tính liên tục

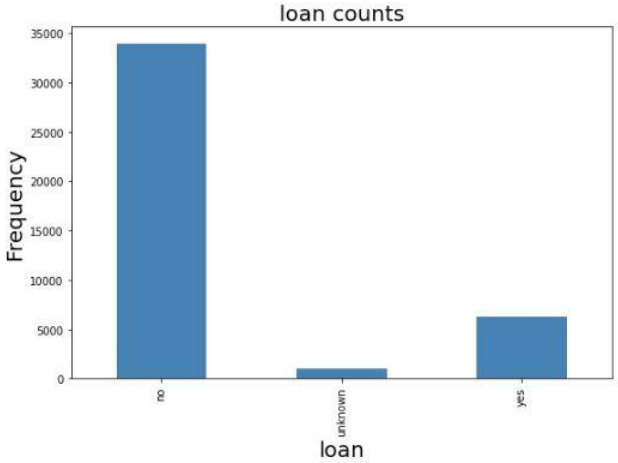
- **Sự phân bố của các trường thuộc tính rời rạc**

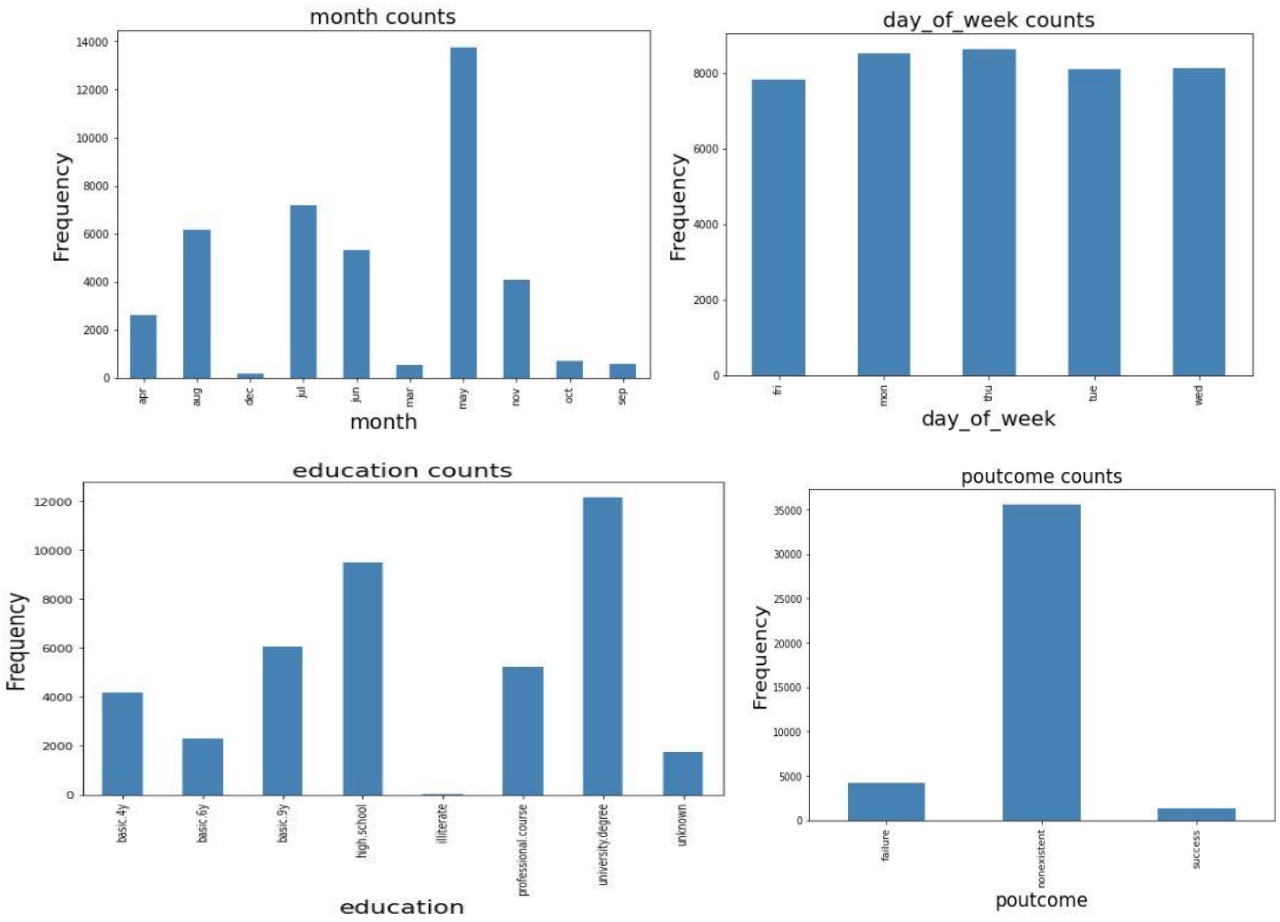






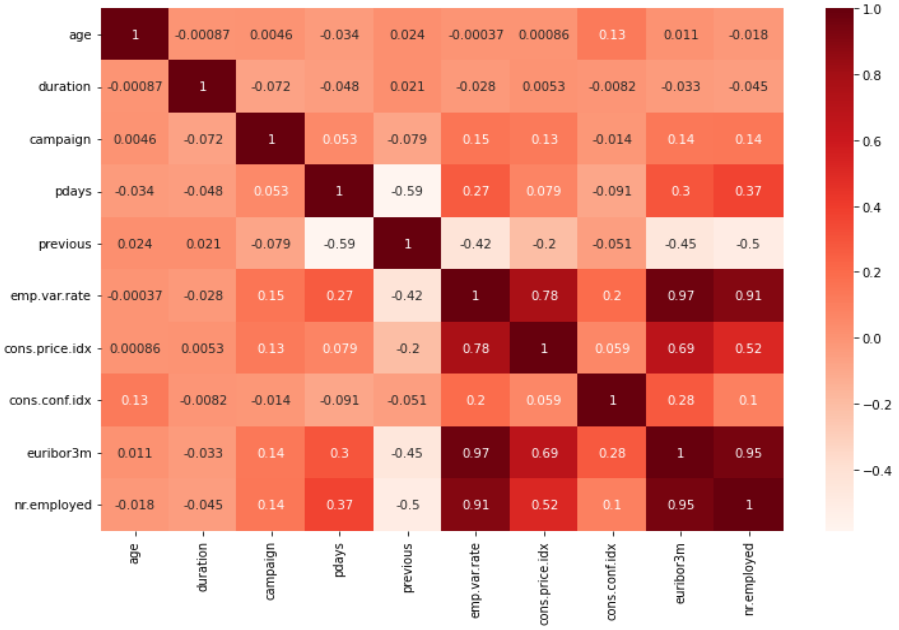






Hình 4.3. Phân bố dữ liệu các thuộc tính rời rạc

- **Tương quan dữ liệu**



Hình 4.4. Tương quan dữ liệu

- **Nhận xét:** Với những mô tả ban đầu về bộ dữ liệu, ta thấy còn tồn tại vài đặc trưng có độ chênh lệch cao so với những phần còn lại. Về output có sự chênh lệch hẳn về 1 bên (yes: 4640; no:36548). Nên trước khi tiến hành training cho mô hình KNN ta tiến hành xử lý dữ liệu và cân bằng dữ liệu, cụ thể sau khi cân bằng (yes: 27387; no:27387).

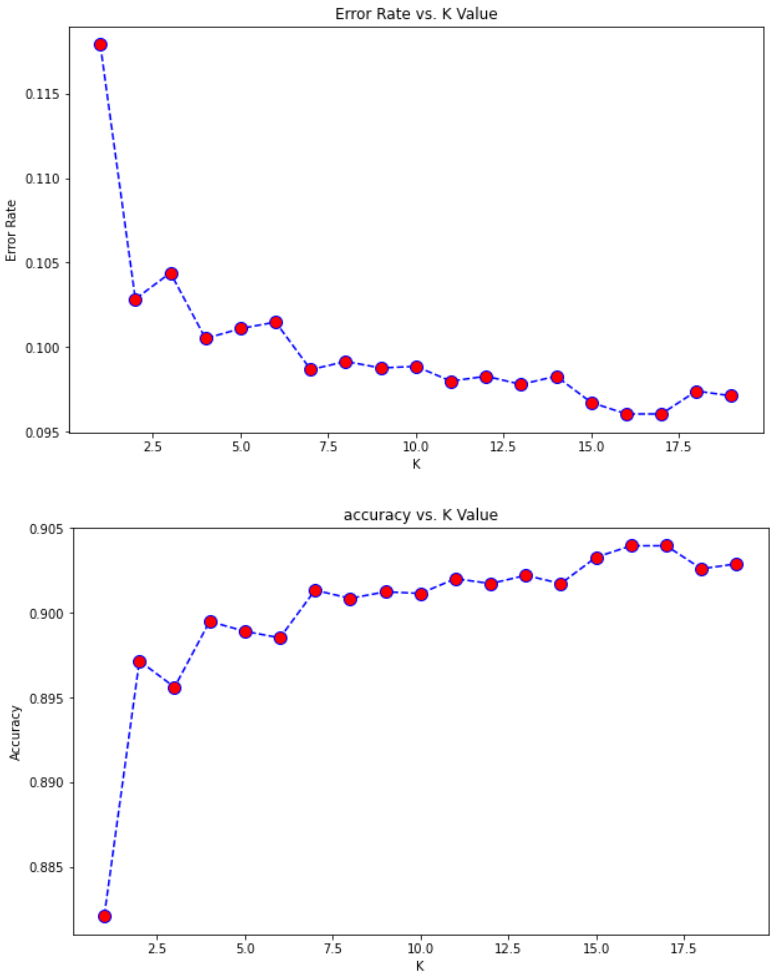
4.2. Kết quả thực nghiệm

- **Với số K = 20 (ngẫu nhiên)**. Tiến hành so sánh trên auc_roc, ROC càng cao thì mô hình càng chính xác trong việc phân loại các lớp.
  - **Kết quả thu được:**

Thuật toán / Kết quả	KNN (k=20)
Train Accuracy	0.589
Test Accuracy	0.901
Train ROC	0.588
Test ROC	0.585
F1	0.282
Precision	0.695
Recall	0.177

➢ **Nhận xét:** Qua bảng kết quả có thể thấy được với K ngẫu nhiên (20) recall không cao (0.177). Model không thật sự tốt mặc dù accuracy trong tập test khá cao(90%).

- **Thử trên các trường hợp K khác nhau**



Hình 4.5. Kết quả độ lỗi và accuracy trên K khác nhau

- **Kết quả thử nghiệm với K có độ lỗi nhỏ nhất (K=15)**

Thuật toán / Kết quả	KNN (k=15)
Train Accuracy	0.926
Test Accuracy	0.873
Train ROC	0.926
Test ROC	0.719
F1	0.475
Precision	0.436
Recall	0.52

➢ **Nhận xét:** So với mô hình mặc định, tuy accuracy trên tập test có giảm (**còn 0,873**), nhưng recall đã tăng lên đáng kể (0.17->**0.52**). Mô hình dự đoán đúng nhiều trường hợp positive hơn.

4.3. Tuning model

- **Grid Search**

```
param_grid = {'n_neighbors': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
              'weights': ["uniform", "distance"],
              'metric': ['euclidean', 'manhattan', 'minkowski']
             }
```

- **Kết quả sau khi tuning với best_param = 'metric':** 'euclidean', **'n_neighbors':** 15, **'weights':** 'distance'}.

Thuật toán / Kết quả	KNN (tuning)
Train Accuracy	1
Test Accuracy	0.903
Train ROC	1
Test ROC	0.658
F1	0.441
Precision	0.61
Recall	0.345

➢ **Nhận xét:** Có thể thấy sau khi tuning ta có thể chọn ra được các parameters đem lại accuracy trên tập train và tập test khá cao, nhưng recall và test_roc không mấy khả quan, model còn khá yếu.

4.4. So sánh với các mô hình khác

4.4.1. So sánh kết quả với model khác

- **Không tuning:**

Thuật toán / Kết quả	KNN	Decision Tree	Random Forest	SVM	Logistic Regression
Train Accuracy	0.589	1	1	0.889	0.908
Test Accuracy	0.901	0.887	0.912	0.892	0.91
Train ROC	0.588	1	1	0.526	0.684
Test ROC	0.585	0.725	0.733	0.521	0.684
F1	0.282	0.504	0.559	0.085	0.492
Precision	0.695	0.492	0.628	0.626	0.662
Recall	0.177	0.517	0.503	0.045	0.392

➢ **Nhận xét:**

- Các mô hình sở hữu test acc khá tương đồng và cao nhất là RF.
- Test ROC của RF là cao nhất trong khi đó KNN khá lép vế.
- Recall KNN tệ thứ 2 sau SVM.

- **Turning GridSearch:**

Thuật toán / Kết quả	KNN	Decision Tree	Random Forest	SVM	Logistic Regression
Train Accuracy	1	0.92	0.932	0.902	0.908
Test Accuracy	0.903	0.914	0.912	0.902	0.905
Train ROC	1	0.759	0.742	0.715	0.776
Test ROC	0.658	0.738	0.678	0.705	0.76
F1	0.441	0.568	0.488	0.504	0.573
Precision	0.61	0.638	0.688	0.571	0.572
Recall	0.345	0.511	0.378	0.453	0.575

➢ **Nhận xét:**

- Test ROC của KNN đã tăng lên nhiều nhất.
- Recall của KNN được tăng lên nhiều nhất so với các model khác.

- Mô hình sở hữu Test Accuracy cao nhất: Decision Tree.
- Mô hình sở hữu Recall cao nhất: Logistic Regression.
- Mô hình sở hữu Test ROC cao nhất: Logistic Regression.

- **Bagging Base Model để tìm kết quả hiệu quả nhất (tìm hiểu thêm)**
  - **KNN**

Thuật toán / Kết quả	Bagging KNN
Train Accuracy	0.926
Test Accuracy	0.875
Train ROC	0.926
Test ROC	0.713
F1	0.471
Precision	0.441
Recall	0.506

  - **Decision Tree**

Thuật toán / Kết quả	Bagging DT
Train Accuracy	0.924
Test Accuracy	0.917
Train ROC	0.784
Test ROC	0.76
F1	0.6
Precision	0.642
Recall	0.556

➢ **Nhận xét:**

- Các mô hình cải thiện rất nhiều khi dùng với Bagging.
- Recall và Test ROC của KNN tăng lên đáng kể.
- Bagging với Decision Tree là mô hình phù hợp nhất với bài toán này sau khi thực hiện tất cả các so sánh dựa trên recall, roc và accuracy.

4.4.2. So sánh time và bộ nhớ giữa các model

❖ **Time:**

- Hai giá trị kết quả bao gồm:
  - Training time: thời gian để train một model.
  - Testing time: thời gian test một model.
- Cách thức so sánh: so sánh thời gian trung bình 10 classification model (default parameters)/1 thuật toán,trên bộ data được split với test size=0.5

➢ Kết quả khi chạy với 50% số lượng data:

Môhình	KNN	LR	SVM	GNB	DT	RF
Training time (S)	0.002119	0.183751	8.859340	0.009916	1.228011	14.484036
Testing time (S)	10.704676	0.001218	9.161111	0.008848	0.004216	0.303394

➢ Kết quả khi chạy với 100% sốlượng data:

Môhình	KNN	LR	SVM	GNB	DT	RF
Training time (S)	0.002269	0.133436	1.002881	0.005495	0.408153	5.157030
Testing time (S)	3.839734	0.000827	1.123359	0.004656	0.001741	0.116973

➢ Kết quả khi chạy 100% số lượng data với pca số chiều từ 25->13:

Môhình	KNN	LR	SVM	GNB	DT	RF
Training time (S)	0.002269	0.049367	8.068048	0.006840	0.508534	8.067495
Testing time (S)	3.413046	0.000864	6.752073	0.005540	0.003467	0.313101

- **Nhận xét:**
  - KNN cho thấy tốc độ training rất nhanh, nhanh nhất trong các thuật toán và không phụ thuộc vào cả số lượng dât và số chiều data. Nguyên nhân do KNN là thuật toán "lười" nên trong quá trình training KNN chỉ mất thời gian để lưu toàn bộ dữ liệu ngoài ra không thực hiện việc tính toán khác .
  - Tuy nhiên tốc độ testing của KNN lại thấp hơn rất nhiều so với tốc độ training nhanh chóng của nó và thấp nhất trong các model.

- Tốc độ testing của KNN còn tăng khi tăng số lượng data và số chiều data cụ thể từ 3.14s→10.7s khi tăng số lượng, từ 3.84s→10.7s khi tăng số chiều. Nguyên nhân do phải tính toán khoảng cách từ điểm test đến nhiều điểm trong tập train và tính toán phức tạp hơn với số chiều nhiều hơn khi test.

❖ **Bộ nhớ:**

- Cách thức: tính dung lượng của một model sau khi fit với data train.

➢ Kết quả sau khi fit 100% dữ liệu train:

Môhình	KNN	LR	SVM	GNB	DT	RF
Dung lượng (MB)	10.866	0.0018	5	0.0022	0.426	50.68

➢ Kết quả sau khi fit 50% dữ liệu train:

Môhình	KNN	LR	SVM	GNB	DT	RF
Dung lượng (MB)	5.43	0.0018	1.304	0.0022	0.292	32.085

- **Nhận xét:**
  - 4 thuật toán thay đổi dung lượng khi thay đổi số lượng dữ liệu train là: KNN, SVM, DT và RF.
  - KNN là thuật toán có model chiếm nhiều bộ nhớ đứng thứ 2 và tăng gấp đôi dung lượng khi tăng gấp đôi dữ liệu, vì nó lưu toàn bộ dữ liệu train => KNN không phải thuật toán ưu thế về không gian.

4.5. Kết luận

- Với việc thực nghiệm trên bộ data Bank+Marketing, kết quả mong muốn trả về cao nhất sẽ là recall, ROC, dưới đó sẽ là Accuracy. Các số liệu kết quả cho thấy KNN tỏ ra không hiệu quả với các tập data lớn và phức tạp. KNN sử dụng nhiều tài nguyên bộ nhớ và cũng tốn nhiều thời gian hơn so với các giải thuật khác để đưa ra kết quả. Về khía cạnh độ chính xác KNN cũng không quá vượt trội hơn so với các thuật toán khác. Nhưng bù lại KNN có tính đơn giản nên

việc tối ưu tham số dễ dàng hơn và đạt hiệu quả cao hơn so với các mô hình khác và KNN làm việc hiệu quả trên dữ liệu dễ dàng hơn.

- Đồ án cho chúng em cơ hội tìm hiểu nhiều hơn về thuật toán, các cách xử lí khi gặp phải dữ liệu lớn và mất cân bằng. Từ nhược điểm của KNN, chúng em có dịp tìm ra các cách để có thể cải thiện performance, nâng cao nhất có thể hiệu suất của mô hình. Cảm ơn thầy và các anh chị đã luôn giảng dạy để nhóm chúng em có thêm nhiều kiến thức. Nhóm xin chân thành cảm ơn.

TÀI LIỆU THAM KHẢO

[1]https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[2]https://machinelearningcoban.com/2017/01/08/knn/?fbclid=IwAR1xvXpPXcqqGP8z2vJn8mKg75Wjyij7tukFow3NCPwdRL0r3RS9Jq7l--E

[3] http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html?fbclid=IwAR1YZeb41cDpkTh2j3t3XBc8j9n5vohmxKTOmb4q0xjKexWWsmyMpln6NhM

[4] https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[5] Python Data Science Handbook: Essential Tools for Working with Data

[6] Introduction to Machine Learning with Python: A Guide for Data Scientists