BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM
KHOA CÔNG NGHỆ THÔNG TIN

HCMUTE

BÁO CÁO ĐỒ ÁN
MACHINE LEARNING

Giảng viên hướng dẫn: TS. Trần Nhật Quang

Sinh viên thực hiện:

Lê Văn Cường	**19110332**
Nguyễn Hiếu Đan	**19110345**
Bành Đăng Khoa	**19110378**
Tạ Bảo Minh	**19110399**
Nguyễn Hoàng Phúc	**19110052**

TP. Hồ Chí Minh, tháng 11 năm 2021

NHẬN XÉT CỦA GVHD

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

GIẢNG VIÊN

Trần Nhật Quang

Mục lục

CHƯƠNG 1: GIỚI THIỆU VỀ MACHINE LEARNING

1.1. Machine Learning là gì?

Machine Learning là một lĩnh vực nghiên cứu ra các phương pháp giúp cho máy tính có thể tự động học từ dữ liệu và sử dụng kinh nghiệm đó để giải quyết các bài toán phức tạp một cách đúng đắn mà khi sử dụng các phương pháp lập trình truyền thống thì sẽ rất mất thời gian và có thể không giải quyết được.

1.2. Tại sao cần phải sử dụng Machine Learning?

Trong cuộc sống chúng ta gặp rất nhiều vấn đề mà các phương pháp lập trình truyền thống không thể xử lý được hoặc có thể xử lý nhưng lại phải tiêu tốn một lượng lớn tài nguyên và thời gian để có thể xử lý được vấn đề ấy. Vì vậy mà Machine Learning ra đời để giải quyết những vấn đề đó.

So sánh Machine Learning và lập trình truyền thống:

Lập trình truyền thống:

- Xử lý bài toán đơn giản ít phức tạp một cách nhanh chóng và chính xác.
- Không thể tự động thông minh lên mà phải phụ thuộc vào sự nâng cấp trực tiếp của người lập trình.
- Bắt buộc người lập trình phải lập trình một cách tường minh.
- Không thể xử lý được một số bài toán phức tạp hoặc mất rất nhiều thời gian và tài nguyên.

Lập trình sử dụng Machine Learning

- Tốc độ huấn luyện dữ liệu và học tập có thể là khá lâu.
- Có thể tự học tập thông minh lên mà không cần tác động nâng cấp từ người lập trình
- Có thể giải quyết nhanh chóng hơn phương pháp lập trình thông thường ở một số bài toán phức tạp.

1.3. Một số khái niệm

- Training data**:** Dữ liệu được dùng để huấn luyện model khác.
- Test data: Dữ liệu được dùng để kiểm tra xem model hoạt động chính xác hay không.

- Sample hay còn được gọi là Example, Instance, data point: là một đối tượng trong tệp dữ liệu.
- Feature: là thuộc tính đặc điểm mà mỗi một đối tượng dữ liệu trong tập dữ liệu có.
- Label: nhãn của data mang kết quả mà máy cần phải cho ra kết quả sau khi đọc dữ liệu đó.
- Labeled data: dữ liệu có chứa label.
- Performance measure: Độ chính xác, khả năng dự đoán đúng.

1.4. Phân loại các thuật toán Machine Learning

1.4.1. Phân loại theo sự giám sát của con người

- Supervised learning sử dụng Labeled data:

Classification(Phân loại): Là phương pháp phân loại một tập dữ liệu thành các lớp khác nhau . Dữ liệu đầu ra là giá trị rời rạc là tên lớp mà dữ liệu thuộc về .

Regression: Là phương pháp cho phép dự đoán một kết quả liên tục dựa trên tập dữ liệu. Dữ liệu đầu ra là liên tục không rời rạc.

Một số thuật toán quan trọng:K- Nearest Neighbors, Logistic Regression, Stepwise Regression, Linear Classifier, Support Vector Machine (SVM), Kernel SVM, Decision Trees and Random Forest.

- Unsupervised learning sử dụng unlabeled data :

Clustering (Phân nhóm) là phương pháp gom các nhóm dữ liệu cùng đặc điểm tương đồng từ tập dữ liệu gốc.

Một số thuật toán quan trọng: K- Means, DBSCAN, Hierarchical Cluster Analysis(HCA)

Anomaly detection là phương pháp phát hiện dữ liệu bất thường trong một tập dữ liệu gốc bình thường.

Một số thuật toán quan trọng:One-Class SVM, Isolation Forest

Dimension reducing(rút gọn số chiều): thu gọn bỏ đi những feature không cần thiết.

Một số thuật toán quan trọng:Principal Component Analysis(PCA), Locally-Linear Embedding(LLE), Neighbor Embedding.

Association rule learning(Học theo mối quan hệ): Học dựa trên mối quan hệ giữa các feature.

Một số thuật toán quan trọng: Apriori, Eclat.

- Semi Supervised learning: là loại phương pháp học kết hợp cả hai loại trên vì dữ liệu có label thì ít và tốn kém nhưng lại chính xác hơn và nhanh hơn còn dữ liệu không có label thường có rất nhiều và thời gian học lâu hơn. Kết hợp chúng lại giúp cân bằng giữa chi phí và hiệu quả.
- Reinforcement learning: Không sử dụng label mà sử dụng reward là điểm cộng hoặc điểm trừ. Sau khi thuật toán thực hiện hành động thì sẽ cho reward để máy có thể nhận biết được hành động nên và không nên. Quá trình chạy rất nặng thường được ứng dụng vào game như alpha go.

1.4.2. Phân loại theo cách thức vừa học vừa thực hiện

- Online learning
- Batch learning

1.4.3. Phân loại theo sử dụng model hay không

- Instance-based learning: So sánh dữ liệu mới và dữ liệu cũ để xem nó giống nhóm nào bằng cách xem xét các feature không cần model
- Model-based learning: Sử dụng model để phân nhóm dữ liệu.

CHƯƠNG 2: THÁCH THỨC CỦA MACHINE LEARNING

2.1. Bad data(dữ liệu xấu)

2.1.1. Thiếu về lượng:

Xảy ra khi thuật toán không đủ dữ liệu có quá ít dữ liệu để huấn luyện nên thuật toán sẽ chạy không tốt.

2.1.2. Dữ liệu không có tính đại diện

Xảy ra khi dữ liệu không có mang tính đại diện cho dữ liệu thực tế gồm:

- Dữ liệu nhiễu (noise data): các dữ liệu quá khác biệt so với phần còn lại.
- Dữ liệu Sampling bias: Lấy mẫu không hợp lý mẫu không có tính đại trà ngẫu nhiên mà chỉ lấy ở một phần nhỏ có chung đặc điểm.

2.1.3. Thiếu dữ liệu (Missing data)

Trong tập dữ liệu có vài sample bị thiếu một vài feature null hoặc feature đó khác biệt quá so với phần còn lại. Cách xử lý phổ biến: bỏ luôn cột đó nếu mà feature đó thiếu quá nhiều, bỏ luôn cái sample bị thiếu, điền vào giá trị đó một giá trị trung bình hoặc trung vị

2.1.4. Irrelevant features

Là tập các feature không liên quan đến bài toán.

2.2. Bad Algorithms (Thuật toán xấu)

2.2.1. Overfitting

Xảy ra khi model cố gắng học tất cả sample trên tập dữ liệu điều này khiến cho những sample xấu rời ra cũng được học cho ra kết quả tập train rất tốt nhưng khi đưa vào thực tế lại thành xấu.

2.2.2. Underfitting

Ngược lại với Overfitting, Underfitting xảy ra khi học ít sample so với tập train điều này dẫn đến độ chính xác thấp trong tập train và tập set và cả tập ở thực tế.

2.3. Testing and hyperparameter turnning

2.3.1. Test Set

Là một tập dữ liệu kiểm tra mức độ đúng đắn hay độ hiệu quả của thuật toán, tập này không liên quan đến tập train.

2.3.2. Generalization error

Lỗi tổng quát hóa dữ liệu không thể học được dữ liệu mới nghĩa là khi đưa dữ liệu mới vào thì không thể học được cho ra kết quả đúng. Lỗi này càng nhỏ thì model càng tốt

2.3.3. Hyperparameter tuning

Hyperparameter là một siêu tham số được thêm vào trước khi thuật toán bắt đầu học.

Hyperparameter tuning là việc điều chỉnh tham số để model hoạt động tốt nhất.

2.3.4. Validation Set

Là 1 tập dữ liệu labeled được dùng để kiểm tra độ chính xác trong quá trình huấn luyện mô hình. Thường được lấy sao cho khái quát hết toàn bộ tập dữ liệu.

CHƯƠNG 3: TỔNG QUAN MỘT DỰ ÁN MACHINE LEARNING

Quá trình tổng quan một đồ án Machine Learning từ lúc bắt đầu đến khi kết thúc

1.

3.1. Các bước để thực hiện Machine Learning Project

3.1.1. Nhìn bức tranh tổng quát

Biết được yêu cầu của bài toán, biết đầu vào là gì, kết quả ta phải đưa ra được là gì là thứ đầu tiên cần phải làm để thực hiện Machine Learning Project. Chúng ta phải xác định rõ việc làm và mục đích để có thể hoạt động hiểu quả, biết được dữ liệu gì cần thu thập để có thể thu thập dữ liệu nhanh và tốt hơn, biết được khách hàng muốn gì để chọn thuật toán nhanh chậm, độ chính xác

3.1.2. Thu thập dữ liệu

Sau khi có bức tranh tổng quát ta phải thu thấp dữ liệu liên quán đến vân đề ta nhận được từ các nguồn như khách hàng, trên mạng hoặc một bên thứ 3 nào đó.

3.1.3. Xử lý dữ liệu lấy insights

Dùng các công cụ như excel… để mở tập dữ liệu. Khám phá và chỉnh sửa tập dữ liệu.

3.1.4. Tiền xử lý dữ liệu để máy có thể học

Tiền xử lý bằng ngôn ngữ lập trình biến đổi dữ liệu thành dạng máy có thể học được.

3.1.5. Huấn luyện và đánh giá model

Dùng thư viện hoặc thuật toán tự xây dựng để tìm ra model và xem xét đánh giá model nào tốt nhất để sử dụng

3.1.6. Tinh chỉnh model

Tinh chỉnh lại thuật toán hay model để có thể đạt được hiệu quả tốt hơn.

3.1.7. Kiểm tra model với dữ liệu thật

Thử model với dữ liệu thật để kiểm tra độ hiệu quả và tốc độ của nó

3.1.8. Triển khai, bảo trì, bảo dưỡng sản phẩm.

3.2. Chi tiết một số bước bằng code

3.2.1. Một số thư viện

- **Pandas:** Giúp chứa dữ liệu dạng mảng 2 chiều rất nhanh và dễ. Sau khi load vào thì sẽ lưu dưới dạng dataframe.
- **Numpy:** Thư viện toán học giúp xử lý ma trận số học một cách mạnh mẽ.
- **Matplotlib:** Thư viện hỗ trợ việc vẽ biểu đồ.
- **Sklearn:** Thư viện hỗ trợ các thuật toán Machine Learning giúp việc lập trình machine learning dễ dàng hơn.
- **Statistics:** Thư viện cung cấp các hàm để thống kê số học

3.2.2 Lấy dữ liệu

Lấy dữ liệu từ một số trang như laydulieu.com hay kaggle.com.

Sau khi lấy dữ liệu ta lưu trữ thành trang dưới dạng csv để lúc lập trình Python có thể đọc được dễ dàng.

Tiếp tục tinh chỉnh dữ liệu bằng excel như chuyển giá thành giá trị nhỏ hơn xóa 1 số dòng trống.

Sau đó thì đọc dữ liệu bằng hàm dưới đây với pd là pandas

pd.read_csv(r'NguyenHieuDan_19110345_tuan03_giamaytinh.csv')

3.2.3. Khám phá dữ liệu

Một số lệnh của pandas để xem dữ liệu:

- **Infor():** in ra thông tin của data vừa đọc.
- **Head(x):** in ra x dòng đầu tiên của dữ liệu vừa đọc.
- **raw_data['X'].value_counts():** đếm số dòng của cột X.
- **describe():** Mô tả tệp dữ liệu cho biết 1 số thông tin như đếm số dòng, giá trị max, min, median… của các cột.
- **loc và iloc:** dùng để truy xuất dữ liệu theo hàng hay cột.

Một số lệnh của Matplotlib để vẽ biểu đồ dữ liệu:

- *raw_data.plot(kind="scatter", y="GIÁ-NGHÌN ĐỒNG", x="RAM-GB", alpha=0.2)*

Hàm này có chức năng vẽ biểu đồ Scatter với các tham số là tên cột ở trong tệp dữ liệu các cột này phải chứa giá trị ở dạng số.

- *plt.savefig('figure/scatter_1_feat.png', format='png', dpi=300)*

Hàm này có chức năng lưu biểu đồ dưới dạng png.

- *from pandas.plotting import scatter_matrix*

features_to_plot = ["Ổ CỨNG-GB"]

scatter_matrix(raw_data[features_to_plot], figsize=(12, 8))

Những hàm này có tác dụng vẽ ra histogram của sample có cột Ổ CỨNG - GB.

Một số lệnh xem độ tương quan dữ liệu (correlation) :

- **raw_data.corr():** Hàm này cho biết độ tương quan tỉ lệ thuận tỉ lệ nghịch giữa các cột với nhau.

3.2.4. Tiền xử lý dữ liệu

Loại bỏ những feature không cần thiết:

- **Drop(columns = [“X”,”Y”]) :** Truyền vào tên của feature X Y để loại bỏ các cột X Y ra khỏi tệp dữ liệu.

Chia tệp dữ liệu ra làm 2 tệp train set và test set:

- **Phương pháp 1: Lấy sample ngẫu nhiên**

from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(raw_data, test_size=0.2, random_state=42)

Những lệnh trên dùng để chia tệp dữ liệu raw_data thành 2 phần test_set và train_set một cách ngẫu nhiên tỉ lệ là 8/2.

- **Phương pháp 2: Lấy theo khoảng giá trị:**

raw_data["KHOẢNG GIÁ"] = pd.cut(raw_data["GIÁ-NGHÌN ĐỒNG"],
bins=[0, 5000, 10000, 20000, 30000, np.inf],
#labels=["<5 triệu", "5-10 triệu", "10-20 triệu", "20-30 triệu", "30-40 triệu", ">50 tỷ"])
labels=[5,10,20,30,100])

Hàm trên chia tệp dữ liệu ra thành các khoảng giá trị khác nhau.

Sau đó từ các khoảng giá trị đó ta lấy ngẫu nhiên như phương pháp 1 rồi ghép chúng lại với nhau để được 2 tập train set có đủ các khoảng giá trị.

Tách label ra khỏi hai tập dữ liệu train và test:

Ta tách bằng cách sao chép cột label ra 1 bảng mới và dùng hàm drop() để bỏ đi cột label ở 2 bảng train và test.

PineLine:

PipeLine xử lý những giá trị trống giá trị không phù hợp thành những dạng mà model có thể học được.

- **ColumnSelector:** class để biến đổi giá trị trong cột để thực hiện mục đích tiền xử lý dữ liệu.

class ColumnSelector(BaseEstimator, TransformerMixin):

def __init__(self, feature_names):

```
        self.feature_names = feature_names
    def fit(self, dataframe, labels=None):
        return self
    def transform(self, dataframe):
        return dataframe[self.feature_names].values
```

- **Chúng ta có 2 dạng dữ liệu là number và categorical**

Đầu tiên ta sẽ tách các cột theo dạng number và categorical riêng ra để xử lý tùy vào từng trường hợp:

num_feat_names = ['RAM-GB', 'Ổ CỨNG-GB',"MAX SIZE-INCH","MIN SIZE-INCH"] # =list(train_set.select_dtypes(include=[np.number]))

cat_feat_names = ['HÃNG',"TÌNH TRẠNG","BẢO HÀNH","CHIP","CARD MÀN HÌNH"]

+ Với dữ liệu là number:

```
num_pipeline = Pipeline([
    ('selector', ColumnSelector(num_feat_names)),
('imputer', SimpleImputer(missing_values=np.nan, strategy="median", copy=True)), # thêm vào những ô bị khuyết giá trị median
('std_scaler', StandardScaler(with_mean=True, with_std=True, copy=True)) # Scale features to zero mean and unit variance
])
```

Std_Scaler Giúp cho các giá trị feature không quá cách biệt giúp thuật toán chạy tốt hơn.

+ Với dữ liệu là categorical:

```
cat_pipeline = Pipeline([
    ('selector', ColumnSelector(cat_feat_names)),
    ('imputer', SimpleImputer(missing_values=np.nan, strategy="constant", fill_value = "NO INFO", copy=True)), # Điền vào ô khuyết giá trị NO INFO
```

('cat_encoder', OneHotEncoder()) # Chuyển dữ liệu thành dạng one hot vector

])

One hot vector là dạng dữ vector giúp cho khoảng cách giữa các giá trị khác nhau là 1.

3.3.5. Huấn luyện

- **Linear Regression model:**

Là thuật toàn học cho ra model là một đường thẳng. Không thể cho ra đường cong theo dữ liệu. Thuật toàn này là thuật toán Supervied learning đưa ra đầu ra là y liên tục dựa trên x là các feature. Để sử dụng thuật toán này ta có thể sử dụng LinearRegression của thư viện Sklearn:

from sklearn.linear_model import LinearRegression

model = LinearRegression()

Sau đó dùng hàm *model.fit(processed_train_set_val, train_set_labels)* để tiến hành huấn luyện nó.

Tiếp theo ta tính độ sai số của nó với r2score là độ chính xác của nó với 1 là tốt nhất và 0 là tệ nhất, rmse là sai số trung bình của nó càng lớn càng tệ.

def r2score_and_rmse(model, train_data, labels):

r2score = model.score(train_data, labels)

from sklearn.metrics import mean_squared_error

prediction = model.predict(train_data)

mse = mean_squared_error(labels, prediction)

rmse = np.sqrt(mse)

return r2score, rmse

Ta có hàm *Predict()* dùng in dự đoán giá trị sau khi đã học thường thì chúng ta in ra giá trị label của sample đó luôn để có thể so sánh đánh giá:

model.predict(processed_train_set_val[0:9]).round(decimals=1)

In ra kết quả dự đoán của tập train hàng 0 đến 9.

- **Decision Tree Regressor model:**

Là thuật toán học dựa trên cây quyết định. Cây quyết định gồm RootNode nut gốc, các nhánh đại diện cho quyết định của cây và nút lá là label. Hàm thực hiện giá trị ước lượng cho ra các giá trị là số thực nên nó sẽ liên tục. Mới đầu sẽ vào nút gốc au đó dần dần đi theo các đường quyết định chúng ta sẽ đến được nút là là kết luận của thuật toán.

Cũng có các hàm như thuật toán *LinearRegression model* ta có các bươc tương tự được trình bày ở trên.

- **Random Forest Regressor model:**

Là thuật toán tốt nhất cho đến khi neuron network ra đời. Dùng nhiều cây để giúp cho thuật toán học tốt hơn và tất nhiên sẽ hoạt động lâu hơn. Được cải tiền từ DecisionTreeRegressor. Sử dụng n_estimators để quyết định số lượng cây trong thuật toán.

model = RandomForestRegressor(n_estimators = 5, random_state=42)

Cũng có các hàm như thuật toán *LinearRegression model* ta có các bươc tương tự được trình bày ở trên.

- **Polinomial Regression model:**

Thuật toán này cho phép huấn luyện model không theo đường thẳng mà có thể là đường cong theo đa thức như đường thẳng linear thì sẽ là bậc 1 đường cong parabol là bậc 2..Như phương trình dưới đây

F(x) y = ax+b

F(x)y = ax^2 +b

Khác với các thuật toán trên chỉ cần dùng *fit( )* là được thuật toàn này cần phải thêm vài tham số khác với dgree là bậc của model

poly_feat_adder = PolynomialFeatures(degree = 2)

- **Support Vector Machine Regression**

3.3.6. Đánh giá mô hình

Chúng ta đánh giá mô hình sau mỗi lần huấn luyện như thế nào? Chắc chắn không thể dục tập train vì nó không đánh giá được khách quan vì đã học nó r, cũng không thể sài tập test vì phải để nó sau cùng. Vậy để đánh giá một cách có hiệu quả thì chúng ta phải tạo một tập mới là tập validation. Chúng ta có thể cắt

thủ công tập train thành 2 phần train và validation nhưng cũng có thể dùng cách khác là K-fold cross validation.

K-fold cross validation giúp cho chúng ta tách train set và validation mỗi lần chạy validation là mỗi lần tách khác nhau giúp tránh được overfitting và tốt hơn tách thủ công một lần. Hàm này có tham số CV là số phần bạn muôn tách từ train ví dụ như CV=5 thì train set sẽ chia thành 5 phần mỗi lần chạy validation là 1 phần.

nmse_scores = cross_val_score(model, processed_train_set_val, train_set_labels, cv=5, scoring='neg_mean_squared_error')

Nmse_scores là một mảng chứa 5 giá trị sai số trung bình của 5 lần chạy validation.

3.3.7. Tinh chỉnh model

Sau khi tìm được model tốt nhất ở bước trên ta tiếp tục thử các Hyperparameter để cho model của chúng ta tốt hơn nữa. Những hyperparameter này thường được tìm ra bằng trực giác kinh nghiệm, hoặc thử nhiều lần trước khi training.

Các phương pháp tìm kiếm Hyperparameter tốt nhất:

- Grid Search : đây là một hàm của Sklearn hoạt động dựa vào các giá trị muốn thử truyền vào sau đó tiến hành huấn luyện với các giá trị đó, rồi cho ra danh sách kết quả dựa vào đó ta có thể tìm được bộ hyperparameter tốt nhất.
- Random Search : đây là một hàm của Sklearn hoạt động dựa vào các giá trị muốn thử truyền vào nhưng không huấn luyện hết mà từ đó tiến hành chọn ngẫu nhiên một số đểhuấn luyện với các giá trị đó, rồi cho ra danh sách kết quả dựa vào đó ta có thể tìm được bộ hyperparameter tốt nhất.
- Ở phần này ta tiến hành fine-tune với RandomForest model :

{'bootstrap': [True], 'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]}

Ở đây ta train thử với số cây lần lượt là 3,10 và 30 cây số lượng feature tối đa lần lượt là 2,4,6,8 vậy số lần thử là 12 lần bên cạnh đó chúng ta có validate set với CV = 5 thì sẽ là 12 x 5 = 60 lần chạy

grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)

3.3.8. Phân tích và kiểm tra kết quả

Các bước này danh cho Random Forest

- **In ra model tốt nhất:** sử dụng hàm *best_model = search.best_estimator_*
- **In ra các feature quan trọng:** Random forest hỗ trợ cho ta biết feature nào quan trọng *best_model.feature_importances_*
- **Chạy bộ test :** Sử dụng tương tự bộ train để đưa ra các thông số đánh giá model.
- **Chỉnh sửa cuối :** Sau khi kiểm tra xong tiền hành xóa feature không cần thiết và chọn ra những thứ tốt nhất

CHƯƠNG 4: CLASSIFICATION

4.1. Định nghĩa

Classification hay phân loại trong Machine Learning thuộc loại supervised learning. Điều này có nghĩa là chương trình sửa dụng classification sẽ học từ dữ liệu được cung cấp và tạo ra một danh sách các observation(quan sát), mỗi observation là một danh sách các thuộc tính có thể đại diện cho rất nhiều thứ khác nhau như là: độ cao tòa nhà, số phòng…, các thuộc tính này thường được chia thành hai loại:

- Number(dạng số): độ cao tòa nà, số phòng..
- Categories(dạng thể loại): màu sắc, chủng loại hoa…

Sau đó dựa vào các observation và label của chúng mà tiến hành phân loại xử lý dữ liệu chúng ta có các tập quan sát. Nhờ đó khi có Model không có label được đưa vào thì chúng ta có observation của Model đó phân loại và đưa ra label của nó một cách chính xác.

4.2. Các dạng Classification

4.2.1 Binary Classification (Phân loại nhị phân)

Phân loại nhị phân là loại phân loại mà dữ liệu có 2 loại.

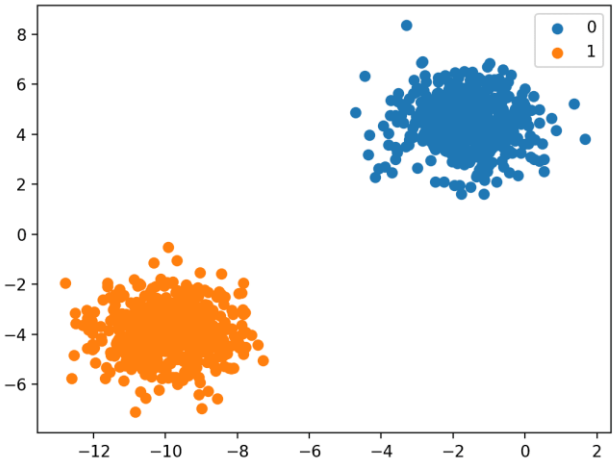
Ví dụ: Phát hiện thư rác(đúng hoặc không),Dự đoán tăng giảm cổ phiếu vào ngày mai (tăng hoặc giảm).

Thông thường thì label liên quan đến đến 2 lớp trạng thài là: bất thường và bình thường, với bình thường là 0 và bất thường là 1.

Đối với loại phân loại nhị phân này người ta thường dùng mô hình dự đoán phân phối Bernoulli là một phân phối rời rạc cho ra kết quả nhị phân là 0 hoặc 1.

Các thuật toán thường được sử dụng:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machine
- Naive Bayes



Ảnh 1: Phân tán của 1 tập dữ liệu phân loại nhị phân

4.2.2. Multi-Class Classification (Phân loại nhiều loại)

Phân loại nhiều loại là loại phân loại mà dữ liệu có nhiều hơn 2 loại.

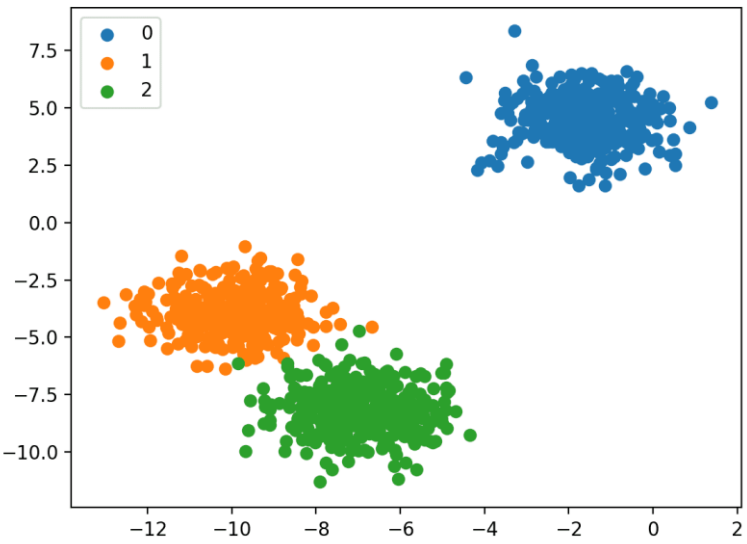
Ví dụ: Nhận diện khuôn mặt, nhận diện loại xe...

Phân loại nhiều loại này không quy về loại bất thường hay bình thường mà phân loại chúng về lớp mà ta đã biết. Số lượng lớp có thể là rất lớn tùy theo vấn đề mà chúng ta làm việc như việc nhận dạng khuôn mặt thế giới có 7 tỉ khuôn mặt khác nhau thì chúng ta có 7 tỉ lớp.

Đối với loại phân loại nhiều loại này người ta thường dùng mô hình dự đoán phân phối Multinoulli là một phân phối rời rạc cho ra kết quả dự đoán xác xuất đối với mỗi lớp.

Các thuật toán thường được sử dụng:

- k-Nearest Neighbors.
- Decision Trees.
- Naive Bayes.
- Random Forest.
- Gradient Boosting.



Ảnh 2: Phân tán của 1 tập dữ liệu phân loại nhiều loại

4.2.3. Multi-Label Classification (Phân loại nhiều nhãn)

Phân loại nhiều nhãn là loại phân loại mà dữ liệu có thể có nhiều hơn 1 loại nhãn.

Ví dụ: Nhận diện ảnh...

Phân loại nhiều nhãn cho ta biết danh sách các nhãn mà ta đã biết. Số lượng nhãn ta có thể có rất nhiều như phân loại ảnh, chúng ta có thể có một bức ảnh gồm rất nhiều đồ vật bên trong nên sẽ có nhiều nhãn hay thuộc về nhiều lớp.

Đối với loại phân loại nhiều loại này người ta thường dùng mô hình dự đoán phân phối Bernoulli Là một phân phối rời rạc cho ra kết quả dự đoán xác xuất đối với mỗi lớp.

Các thuật toán thường được sử dụng:

- Multi-label Decision Trees
- Multi-label Random Forests
- Multi-label Gradient Boosting

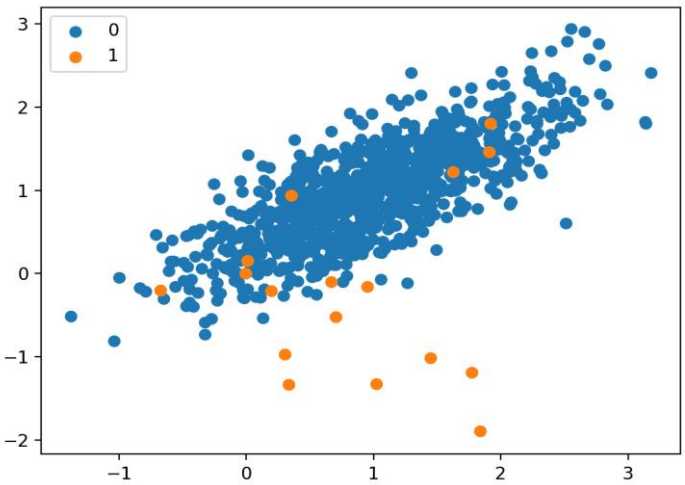
4.2.4. Imbalanced Classification (Phân loại không cân bằng)

Phân loại không cân bằng là loại phân loại mà dữ liệu chia thành các nhóm với số lượng không bằng nhau. Thông thường thì nhiệm vụ của nó cũng giống phân loại nhị phân chia ra thành 2 tập bất thường và bình thường.

Ví dụ: Phát hiện xâm nhập, hành vi bất thường trong bảo mật thông tin.

Phân loại không cân bằng cho ta đối tượng vào là bình thường hay bất thường so với dữ liệu đã được train. Kết quả này cho ta biết hành vi xâm nhập tác động vào máy tính của các đối tượng truy cập là binh thường hay bất thường để có biện pháp đối phó trong an toàn thông tin.

Đối với loại phân loại nhiều loại này người ta thường dùng mô hình dự đoán phân phối Bernoulli như phân phối nhị phân.



Ảnh 3 Phân tán của 1 tập dữ liệu phân loại không cân bằng

4.3. Quy trình

Bước 1: Chuẩn bị tập dữ liệu huấn luyện (dataset) và rút trích đặc trưng (feature extraction).

Chọn ra các feature tốt, loại bỏ các feature xấu , gây nhiễu. Tính toán số lượng feature sao cho phù hợp đảm bảo tốc độ chạy và độ chính xác của dữ liệu.

Bước 2: Xây dựng mô hình phân lớp (classifier model).

Đây là bước học hay còn gọi là training để tìm ra một hàm f(x) = y với x là các feature đầu vào.

Bước này chúng ta dùng các thuật toán supervised learning như: Logistic Regression, Naive Bayes....

Bước 3: Kiểm tra dữ liệu với mô hình (make prediction).

Đây là bước đưa dữ liệu mới vào để kiểm tra tính chính xác của mô hình có được ở bước 2.

Bước 4: Đánh giá mô hình phân lớp và chọn ra mô hình tốt nhất.

Bước này đánh giá mức độ lỗi sai sót đối với dữ liệu của model mà ta đã tìm được. Qua đó thay đổi tham số để có được model tốt nhất.

4.4. Classification with MNIST

4.4.1. MNIST dataset

Cơ sở dữ liệu MNIST (tiếng Anh: MNIST database, viết tắt từ Modified National Institute of Standards and Technology database) là một cơ sở dữ liệu lớn chứa các chữ số viết tay thường được sử dụng trong việc training nhận diện ảnh số. Ở bài tập em dùng fashion dataset.

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boot

4.4.2. Code

4.4.2.1. Linear Model , Binary Classification

- **Chuẩn bị dữ liệu và huấn luyện:**

Thêm thư viện Keras vào đồng thời lấy 2 tập dữ liệu train và test từ mnist

from tensorflow import keras

(X_train, y_train), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()

Chỉnh lại dữ liệu để tạo thành 1 tập dữ liệu mà 1 dữ liệu có 784 feature.

X_train = X_train.reshape(60000,784)

X_test = X_test.reshape(10000,784)

Chọn BINARY CLASSIFIER ở đây chỉ có 2 lớp là 5 hoặc khác 5.

```
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
```

Huấn luyện dữ liệu bằng cách sử dụng hàm fit với SGDC CLASSIFIER

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

- **Đánh giá dữ liệu:**

Thử nghiệm dự đoán bằng cách sử dụng hàm predict:

```
sgd_clf.predict([X_train[sample_id]])
```

Kiểm tra độ chính xác của model:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

Sử dụng như K-fold cross validation được giới thiệu ở chương trước.

Tiếp theo tạo ra một hàm luôn trả về false để kiểm tra mức độ cân bằng dữ liệu (Imbalance data) nghĩa là nhóm label sai có quá nhiều và dữ liệu chứa label đúng quá ít.

```
from sklearn.base import BaseEstimator
class DumpClassifier(BaseEstimator): # always return False (not-5 label)
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
no_5_model = DumpClassifier()
```

cross_val_score(no_5_model, X_train, y_train_5, cv=3, scoring="accuracy")

Confusion matrix: Cung cấp cho ta biết thuật toán nhận định dữ liệu như thế nào để biết được mức độ sai và loại sai của thuật toán như sau.

Predict Actual	Negative	Positive
Negative	True Negative	False Positive
Positive	False Negative	True Positive

Trong đó True Negative và False Negative càng lớn càng tốt và True Positive và False Positive càng bé càng tốt.

Bên cạnh đó có 2 chỉ số được tạo ra từ bảng này để đánh giá tùy theo mục đích của người lập trình mà ưu tiên chỉ số đó cao hơn

Precision : dùng nếu ưu tiên độ chính xác cao $\frac{True\ Positive}{True\ \ Positive + False\ Positive}$

Recall : dùng nếu ưu tiên tính đầy đủ $\frac{True\ Positive}{True\ \ Positive + False\ Negative}$

Code: *precision_score(y_train_5, y_train_pred)*

recall_score(y_train_5, y_train_pred)

F1 score:

Giá trị F1 score là giá trị cho ta biết thuật toán chạy đúng hay không, là sự tổng hợp của 2 giá trị trên được tính toán theo công thức: $2\ x\ \frac{Precision\ x\ \ recall}{Precision + recall}$

Với 1 là giá trị tốt nhất và 0 là xấu nhất

from sklearn.metrics import precision_score, recall_score

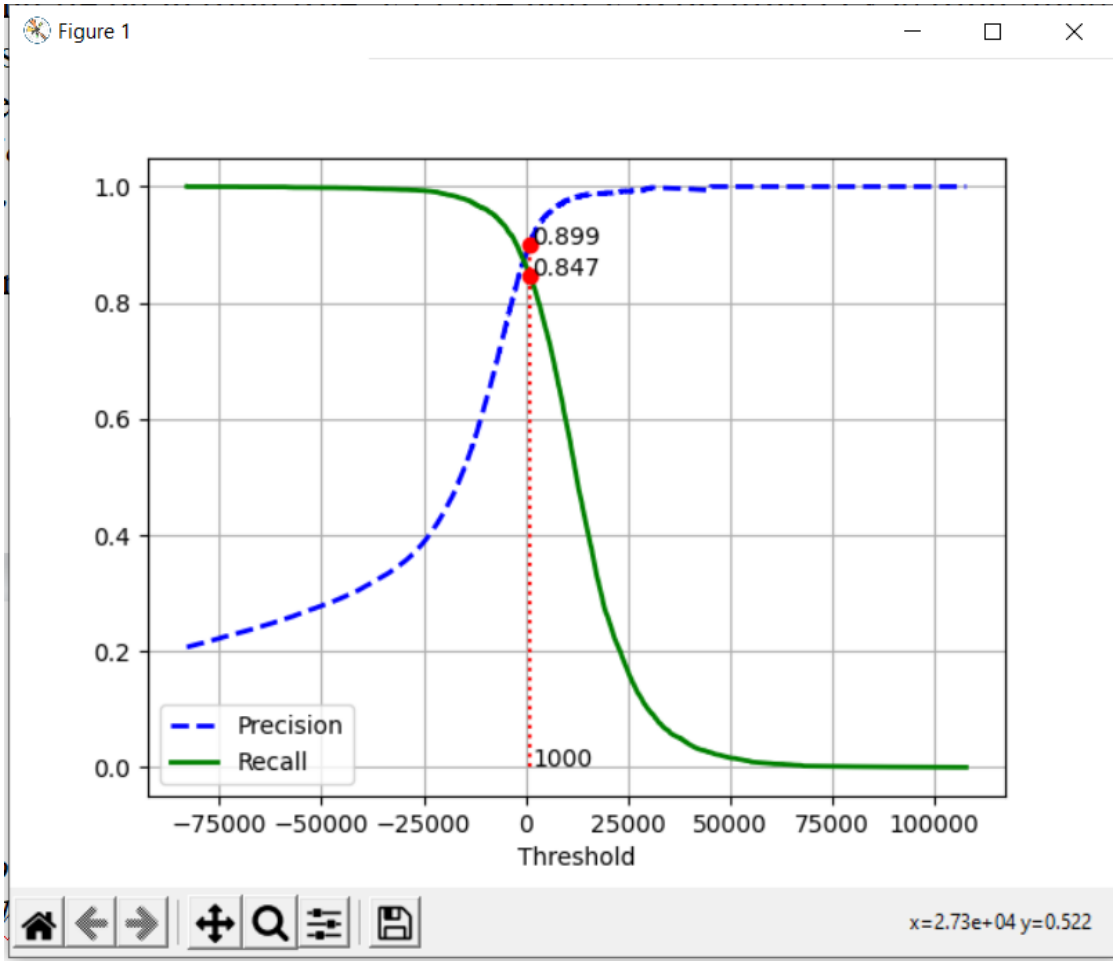
from sklearn.metrics import f1_score

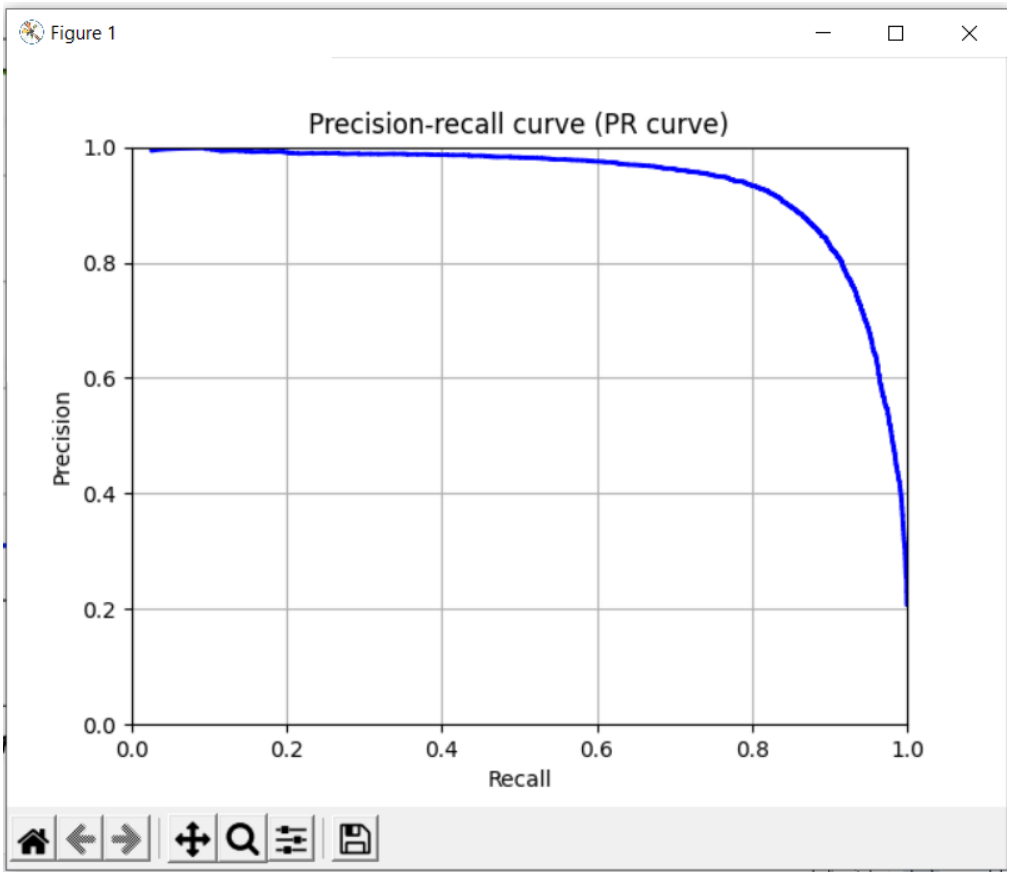
f1_score(y_train_5, y_train_pred)

Threshold:

Là giá trị ngưỡng để phân định true và false nhờ vào đó giúp ta xác định được các giá trị precision, recall và F1 ta có 2 loại sau:

Precision and recall curve: Dạng đồ thị trục x,y với trục y với trục x là gia trị của precision và recall còn trục y ;là giá trị của threshold

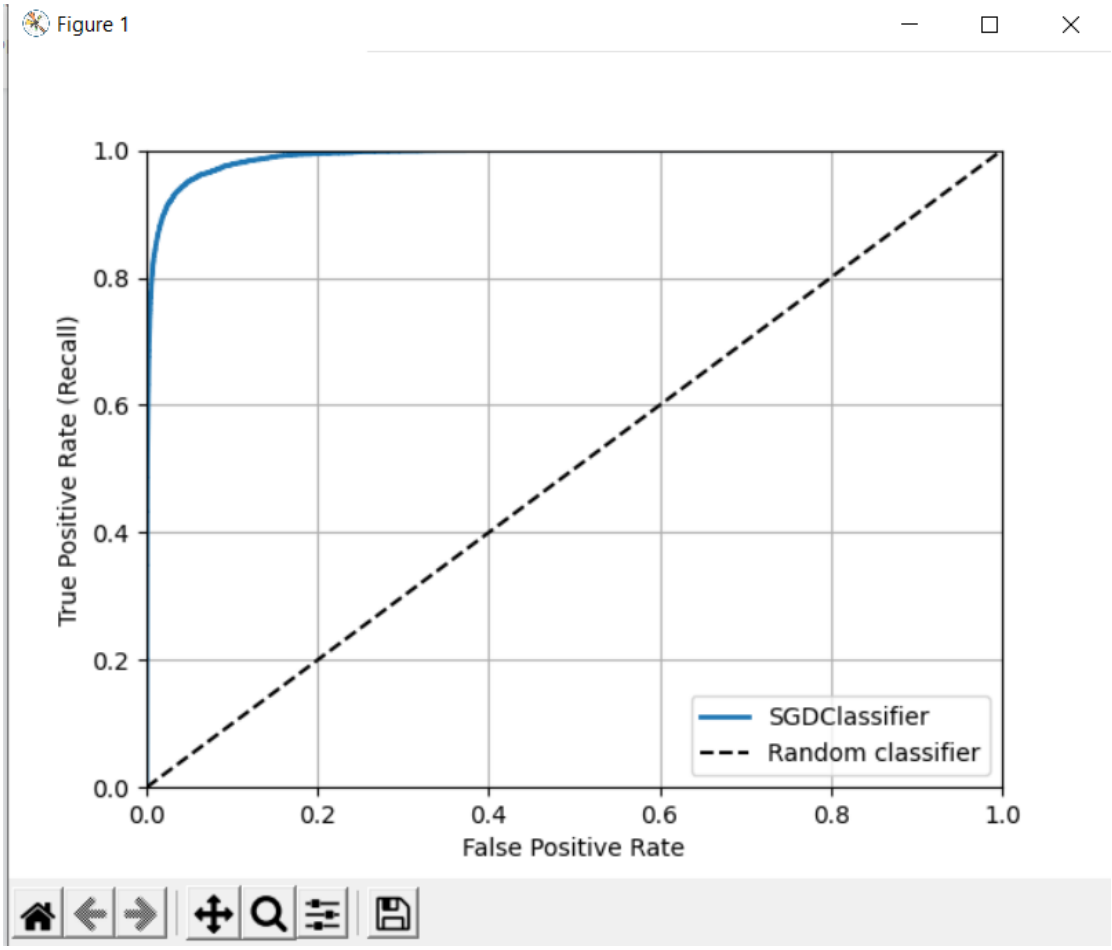




from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

ROC curve: Là dạng đồ thị trục x là FTR và trục y là FPR cho ta thấy mối liên hệ trực tiếp của chúng đồ thị này tốt nhất khi vùng bên dưới bằng 1 và xấu khi bé hơn 0.5. này gồm 2 thông số FTR là recall còn FPR tính theo :$\frac{False\ Positive}{True\ \ Negative + False\ Positive}$



y_scores_random = randomforest_probas[:, 1]

fprr, tprr, thresholdsr = roc_curve(y_train_5, y_scores_random)

4.5. Multiclass Classification

Bài trên ta có 2 label là 5 và khác 5 và chúng ta có thể nhận biết độ hiệu quả dựa trên các thông số liệt kê ở trên. Nếu thuật toán gặp trường hợp imbalance thì ta tiến hành các cách như bỏ bớt dữ liệu, biến đổi nó, thu thập thêm dữ liệu. Và trong trường hợp có nhiều class như NMIST trên thì có tới 10 class nên chúng ta phải sử dụng thuật toán khác thuộc loại multiclass Classification lúc này thì đầu ra có thể là nhiều label chứ ko phải là 2 như binary classification.

4.5.1 One versus all

Bài toán được đưa về dạng nhiều bài toán Binary classification nghĩa là nếu bạn có n class thì nó sẽ tạo ra n class classifier và mỗi class ta có class A và không phải class A. Khi chạy thì thuật toán sẽ trả về cac con số là tỉ lệ đúng là class A chúng ta so sánh các con số với nhau để được class gốc của nó.

Phương pháp này có ưu điểm là nhanh nhưng vẫn bị imbalance.

4.5.2. One versus One

Bài toán đưa về so sánh từng cặp class kiểm tra lần lượt ví dụ ta có 3 classifier là A, B, C thì sẽ so sánh A - B , B - C, A - C như vậy ta có thể tránh được tình trạng imbalance. Mặc dù nhiều nhưng vẫn có thể nhanh hơn OvA vì cần ít dữ liệu hơn. Tùy theo thực tế mà ta chọn cách.

4.5.3. SGDClassifier

Theo Sklearn thì nó sẽ chạy mặc định OvA cho SGD và OvO cho SVM.

sample_scores = sgd_clf.decision_function([X_train[sample_id]])

Dòng này trả lại kết quả là một dãy các điểm số ứng với mỗi class

class_with_max_score = np.argmax(sample_scores)

Dòng này trả lại class có số điểm cao nhất.

ova_clf = OneVsRestClassifier(SGDClassifier(random_state=42))

Cài đặt Ova cho SGDClassifier.

ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))

Cài đặt OvO cho SGDClassifier.

4.5.4. Đánh giá

Multi classification có thể sử dụng Accuracy vì không còn bị imbalance nữa.

Nên sử dụng Feature Scaling khi có chênh lớn giữa các feature hiệu suất có thể tăng rõ rệt. Feature Scaling rất phù hợp với thuật toán SGD còn RandomForest thì không ảnh hưởng.

4.5.5. Phân tích lỗi

Với 10 class thì chúng ta sẽ có 1 confusion matrix với 10 hàng và 10 cột. Sau khi in ra thì sẽ nhìn vào đường chéo để đánh giá nó tốt hay không. Với đường chéo chính càng đỏ đậm càng tôt và xung quanh càng đen đậm càng tốt. Đối với ma trận số thì số ở đường chéo chính càng lớn càng tốt và ở những ô còn lại càng nhỏ càng tốt.

Sau đó chúng ta có thể đánh giá được và đưa ra chỉnh sửa cho tốt. Ở đây thì số 8 dễ nhận nhầm với số theo model thầy hướng dẫn.

Các cách giải quyết: Thêm nhiều dữ liệu hơn, viết hàm nhận thêm đặc điểm, tiền xử lý ảnh bằng một số hàm có sẵn như OpenCV, Pillow…

4.6. Multilabel và Multioutput Classification

4.6.1. Multilabel

Là bài toán khi đầu vào là dữ liệu có nhiều hơn 1 label. Ví dụ: 1 tấm hình có nhiều vật thể cần nhận diện. Đầu ra của nó không phải là một giá trị mà là 1 vector binary tương ứng với thứ tự của label 1 là có và 0 là không.

y_train_large = (y_train >= 7)

y_train_odd = (y_train % 2 == 1)

y_multilabel = np.c_[y_train_large, y_train_odd]

Đây là một trường hợp multilabel với 2 label là >= 7 và số chẵn.

Thường dùng KNeighborsClassifier để xử lý multilabel.

knn_clf = KNeighborsClassifier()

knn_clf.fit(X_train, y_multilabel)

F1_Score của multilabel là con số trung bình cộng của tổng tất cả class.

4.6.2. Multioutput

Là bài toán đầu ra là một vectơ bất kì có thể là binary vector hoặc các con số đại diện cho những thứ khác, còn lại tương tự Multilabel.

CHƯƠNG 5: TRAINING

5.1. Linear Regression

5.1.1. Hypothesis function

Hypothesis function là model mô tả dữ liệu (ví dụ như đường thẳng) được dùng để dự đoán label

Hypothesis function của Linear Regression có n feature :$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.

Với $[\theta_0 , \theta_1, \theta_2,\ldots, \theta_n]$ là các tham số cần tìm trong quá trình training

5.1.2. Cost functions

Cost functions là hàm dùng tính toán sai số của các hypothesis functions, từ đó chọn ra hypothesis functions tốt nhất

MSE cost functions: $MSE(X,h_\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(X^i) - y^i)^2$ Là bình phương sai số trung bình của các sample

Cost function càng thấp thì chứng tỏ Hypothesis function càng tốt

Có 2 cách tìm θ để cost function đạt được giá trị thấp nhất:

- Normal equation: Giải đạo hàm của cost function để ra tìm θ làm cho cost function đạt cực tiểu

$$\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)$$

Trong đó:

- X là ma trận các giá trị các feature.
- y là label
- X^T là ma trận chuyển vị của X
- $(X^T X)^{-1}$ là ma trận nghịch đảo của ma trận $X^T X$

-Gradient descent: Tạo giá trị θ ngẫu nhiêu để tính cost function, sau đó dựa vào giá trị đạo hàm của cost function để tăng hoặc giảm θ. Thực hiện nhiều lần để tìm được cost function nhỏ nhất

5.1.3.Gradient descent

Sau mỗi lần lặp, $\theta = \theta - \nabla$

Trong đó, ∇ là learning rate dùng để tăng hoặc giảm tốc độ thay đổi của θ

∇ được xác định trong từng thuật toán khác nhau

-Batch gradient descent: Chạy trên tất cả sample

hay dạng vector

$$J'_{\theta_j} = \frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{m}\sum_{i=1}^{m} (x^{(i)}\theta - y^{(i)}) x_j^{(i)}$$

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

-Stochastic gradient descent: Chỉ chọn ra 1 sample ngẫu nhiêu

$$J'_{\theta_j} = \frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{1} \sum_{i=random}^{1} (x^{(i)}\theta - y^{(i)}) x_j{}^{(i)}$$

-Mini-Batch gradient descent: chọn ra một tập các sample: 1≤ k ≤ m

$$J'_{\theta_j} = \frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{k} \sum_{i=1}^{k} (x^{(i)}\theta - y^{(i)}) x_j{}^{(i)}$$

5.2. Polynomial regression

Hypothesis function của Polynomial regression models bậc n với 1 feature có dạng: $\mathbf{y = a_0 + a_1x_1 + a_2x_1^2 + \ldots + a_nx_1^n}$ giống của linear Regression $\mathbf{y = a_0 + a_1x_1}$ nên ta chỉ cần xem mỗi $\mathbf{x_1^k}$ là một feature riêng biệt. Như vậy, Polynomial regression **m** feature trở thành linear regression có **n.m** feature rồi giải như là linear regression

5.3. Underfitting và overfitting

Có 2 cách để xác định model sau khi trên có bị underfitting hoặc overfitting là Cross –validation và learning curves

Đặc điểm của cross validation:

- Underfitting: kết quả trên tập training và validation đều không tốt, xảy ra khi model quá đơn giản so với dữ liệu

-Overfitting: kết quả trên tập training tốt nhưng trên tập validation lại không tốt, có thể do model quá phức tạp so với dữ liệu

Underfitting dễ phát hiện nhưng overfitting rất khó phát hiện. Do dó, phải sử dụng learning curves để vẽ ra performance của model

Đặc điểm của learning curves:

- Underfitting: MSE của 2 tập training và validation đều cao, khoảng cách giữa 2 đường nhỏ

- Overfitting: Overfitting: MSE của tập training thấp ,của validation cao, khoảng cách giữa 2 đường lớn

5.4. Regularization

Ý tưởng: Sử dụng model phức tạp để đảm bảo không bị underfitting rồi thêm ràng buộc để overfitting trở thành good-fitting bằng cách cộng thêm 1 regularization term cho cost function

$$J(\theta) = MSE(\theta) + regularization\ term$$

5.4.1.Ridge regression

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \quad {\theta_i}^2$$

Trong đó α dùng để hằng số được chọn trước để điều chỉnh hiệu quả của $regularization\ term$,α càng lớn thì hiệu quả càng lớn

Để $\alpha \frac{1}{2} \sum_{i=1}^{n} \quad {\theta_i}^2$ càng nhỏ thì θ phải càng nhỏ (<1)

5.4.2. Lasso regression

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \quad |\theta_i|$$

Với θ nhỏ hơn 1 thì $|\theta_i| > {\theta_i}^2$ do đó Lasso regression có hiệu quả tốt hơn Ridge regression

5.4.3.Elastic net

Kết hợp cả 2 công thức trên ta được:

$$J(\theta) = MSE(\theta) + r\alpha \frac{1}{2} \sum_{i=1}^{n} \quad |\theta_i| + \frac{1-r}{2} \alpha \frac{1}{2} \sum_{i=1}^{n} \quad {\theta_i}^2$$

Trong đó, r dùng để điều chỉnh hiệu quả giữa Ridge và Lasso:r càng lớn thì hiểu quả của lasso càng lớn,r =1 thì chỉ dùng lasso, r =0 chỉ dùng Ridge

5.5. Early stopping

Một số thuật toán như Stochastic, mini-batch khi thực hiện train quá lâu thì gặp phải những model có validation loss giảm tới cực tiểu rồi lại tăng lên mặc dù training loss vẫn giảm.

Để giải quyết vấn đề này thì ta sẽ dừng thuật toán lại khi validation loss bắt đầu tăng

5.6. Logistic Regression

Là thuật toán Classification với giá trị đầu ra là 0(false) hoặc 1(true)

$$\hat{p} = \sigma(t) = \frac{1}{1 + e^{-t}}$$

Cost function của 1 sample: $c(\theta) = \{- log\ log\ (\hat{p})\ \ if\ y = 1\ - log\ log\ (1 - \hat{p})\ \ if\ y = 0$

Nếu y =1

- Nếu $\hat{p}$ >> 0.5 thì dự đoán là 1, $c(\theta)$ nhỏ
- Nếu $\hat{p}$ << 0.5 thì dự đoán là 0, $c(\theta)$ lớn

Nếu y=0

- Nếu $\hat{p}$ >> 0.5 thì dự đoán là 1, $c(\theta)$ lớn
- Nếu $\hat{p}$ << 0.5 thì dự đoán là 0, $c(\theta)$ nhỏ

Cost function cho nhiều sample

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \quad \left[y^{(i)}\ log\ log\ \left(\hat{p}^{(i)}\right) + log\ \left(1 - y^{(i)}\right) log\ \left(1 - \hat{p}^{(i)}\right) \right]$$

Dùng gradient descent để giải:

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\quad\left(\sigma(x^{(i)}\theta)-y^{(i)}\right)x_j{}^{(i)}$$

5.7. Softmax Regression

Là thuật toán giống logistic nhưng đầu ra có nhiều class hơn

$$\hat{p}_k=h_{\theta^{(k)}}(x)=\frac{exp\ exp\left(s_k(x)\right)}{\sum_{j=1}^{K}\quad exp\ exp\ \left(s_j(x)\right)}$$

Để xác định 1 sample thuộc về class nào trong k class ta cần tính $\hat{p}_1, \hat{p}_2,\ldots, \hat{p}_k$. Giá trị của $\hat{p}$ nào lớn nhất thì sample thuộc về class đó

Cost function:

$$J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}\quad\sum_{k=1}^{K}\quad y_k{}^{(i)}\ log\ log\ (\hat{p}_k{}^{(i)})$$

Dùng gradient descent để giải:

$$\nabla_{\boldsymbol{\theta}^{(k)}}J(\boldsymbol{\Theta})=\frac{1}{m}\sum_{i\ =\ 1}^{m}\left(\hat{p}_k^{(i)}-y_k^{(i)}\right)\mathbf{x}^{(i)}$$

CHƯƠNG 6: SUPPORT VECTOR MACHINE

6.1. Giới thiệu

Là thuật toán rất mạnh và linh hoạt trong Machine Learning. Có khả năng thực hiện linear hoặc nonlinear classification, regression, và thậm chí là phát hiện outliner. Support Vector Machine là một trong những thuật toán phổ biến nhất trong Machine Learning. Support Vector Machine hoạt động ổn khi classification bộ dữ liệu phức tạp vừa và nhỏ.

Support Vector là các samples nằm trên margin

6.2. Ý tưởng

Cố gắng tìm một cái "street" rộng nhất có thể giữ các class trong bộ dữ liệu.

6.3. Linear SVM Classification

6.3.1. Mô tả

SVM chỉ có thể phân được 2 class, có nghĩa là label chỉ là 0 hoặc 1. Model sẽ tính ra một decision boundary đại diện là đường thẳng để chia 2 class ra làm 2 phần.

Lưu ý:SVM thì nhạy cảm với features scaling. Để đạt được kết quả tốt nhất thì nên features scaling trước khi huấn luyện model

6.3.2. Code

Ta sẽ thử phân loại 2 loại bông trong iris dataset

Load iris dataset

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]
setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]  # use only 2 classes: setosa, versicolor
y = y[setosa_or_versicolor]
```

Huấn luyện SVM

```
from sklearn.svm import LinearSVC # faster than SVC on large datasets
svm_clf = LinearSVC(C=np.inf) # C: larger => less regularization. loss = 'hinge': standard loss
```

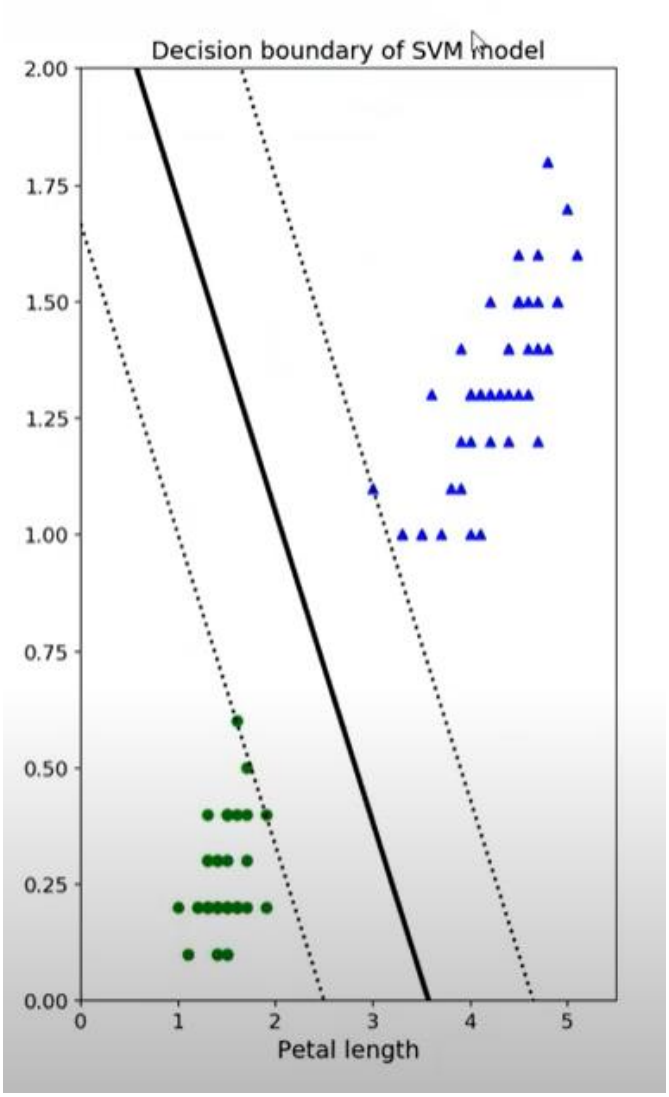
svm_clf.fit(X, y)

svm_clf.predict(X) # Predicted labels

svm_clf.decision_function(X) # Scores of samples

Kết quả: Sau khi huấn luyện xong ta sẽ nhận được models có hình dạng như hình dưới. Một decision boundary với 2 margin chia bộ dữ liệu ban đầu thành 2 phần tách biệt.

Lưu ý: parameter C trong SVC càng lớn thì sẽ chống overfitting càng mạnh

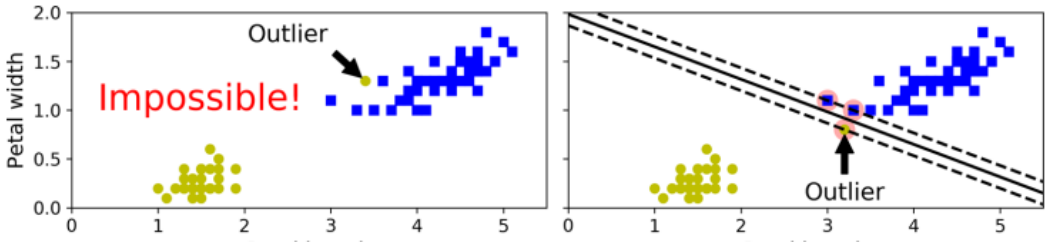


6.4. Soft Margin Classification

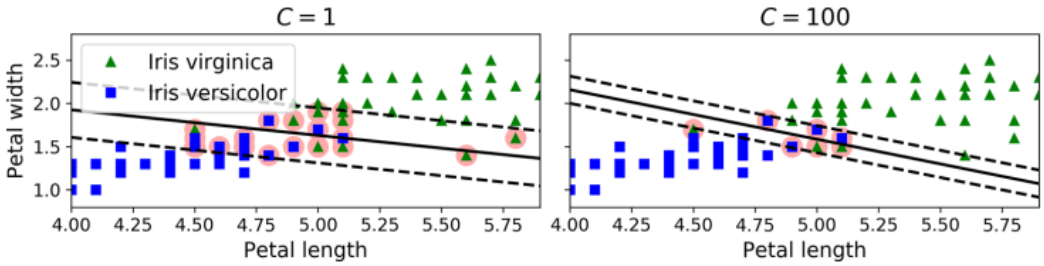
6.4.1. Khái niệm

Mặc định, Linear SVM classification sẽ dùng Hard margin (nghĩa là tất cả samples của 1 class không nằm trong area của class khác). Việc áp đặt nghiêm

ngặt như vậy sẽ chỉ hiệu quả đối với bộ dữ liệu tuyến tính và nhạy cảm với bộ dữ liệu có outliner.



Như hình trên, nếu ta dùng hard margin thì sẽ không tìm được decision boundary do trong bộ dữ liệu có outliner. Để tránh vấn đề trên thì ta cần một model linh hoạt hơn. Đó là soft margin. Soft margin vẫn giữ ý tưởng là tìm được "street" rộng nhất có thể, nhưng phải cân bằng giữ độ rộng và vi phạm margin (dành cho các outliner). Ta sử dụng soft margin trong SVC model bằng cách điều chỉnh giá trị của parameter C và loss.



Khi SVC model bị overfitting, ta có thể regularization bằng cách giảm C.

6.4.2. Code

Load iris dataset

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]
setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]  # use only 2 classes: setosa, versicolor
y = y[setosa_or_versicolor]
```

Huấn luyện SVM với soft margin

```
from sklearn.svm import LinearSVC
svm_clf = LinearSVC(C=1, loss="hinge", random_state=42)
```

```
svm_clf.fit(X, y)
svm_clf.predict(X) # Predicted labels
svm_clf.decision_function(X) # Scores of samples
```

6.5. Nonlinear SVM classification

Mặc dù linear SVM classification hoạt động hiệu quả và ổn với nhiều trường hợp. Tuy nhiên dataset không phải lúc nào cũng cũng tuyến tính để ta tách ra dễ dàng. Để xử lý tập dữ liệu phi tuyến tính thì ta thêm nhiều feature.

6.5.1. Polynomial Kernel

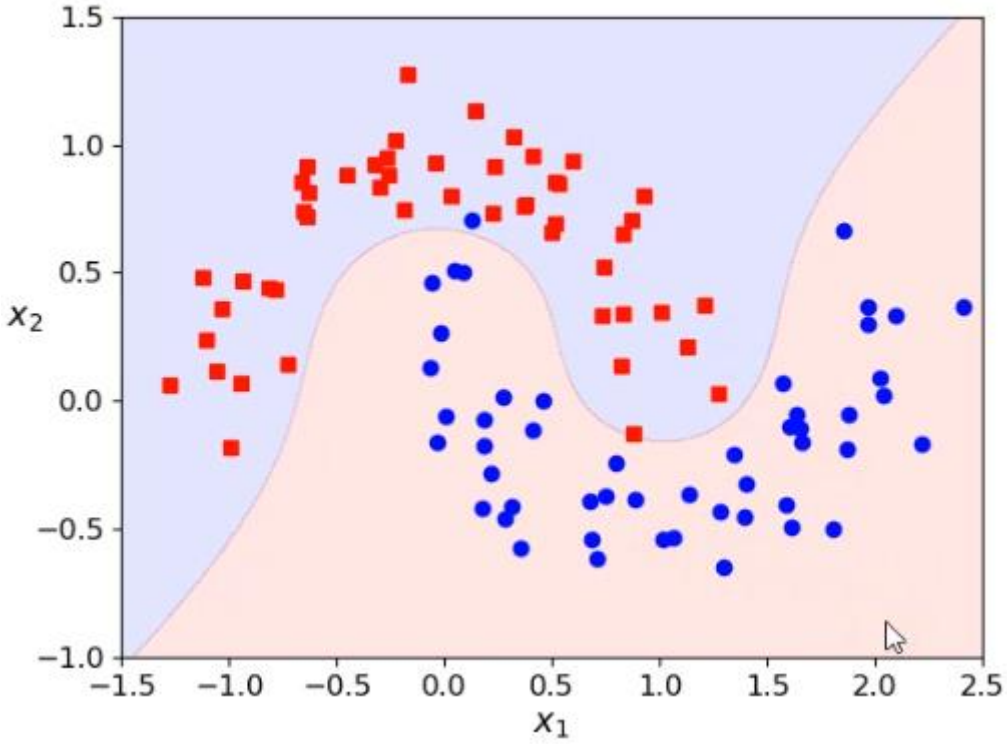
6.5.1.1 Ý tưởng

Đơn giản là thêm feature mới sau đó thực hiện linearSVM như trên. Nhược điểm của polynomial là nếu bậc của model thấp thì không thể giải quyết được dataset phức tạp, nếu bậc của model cao, sẽ tạo ra một lượng features khổng lồ, làm cho model chạy rất chậm.

6.5.1.2.Code

```
from sklearn.preprocessing import PolynomialFeatures
polynomial_svm_clf = Pipeline([
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scaler", StandardScaler()),
    ("svm_clf", LinearSVC(C=40, loss="hinge", random_state=42)) ])
polynomial_svm_clf.fit(X, y)
```

Kết quả



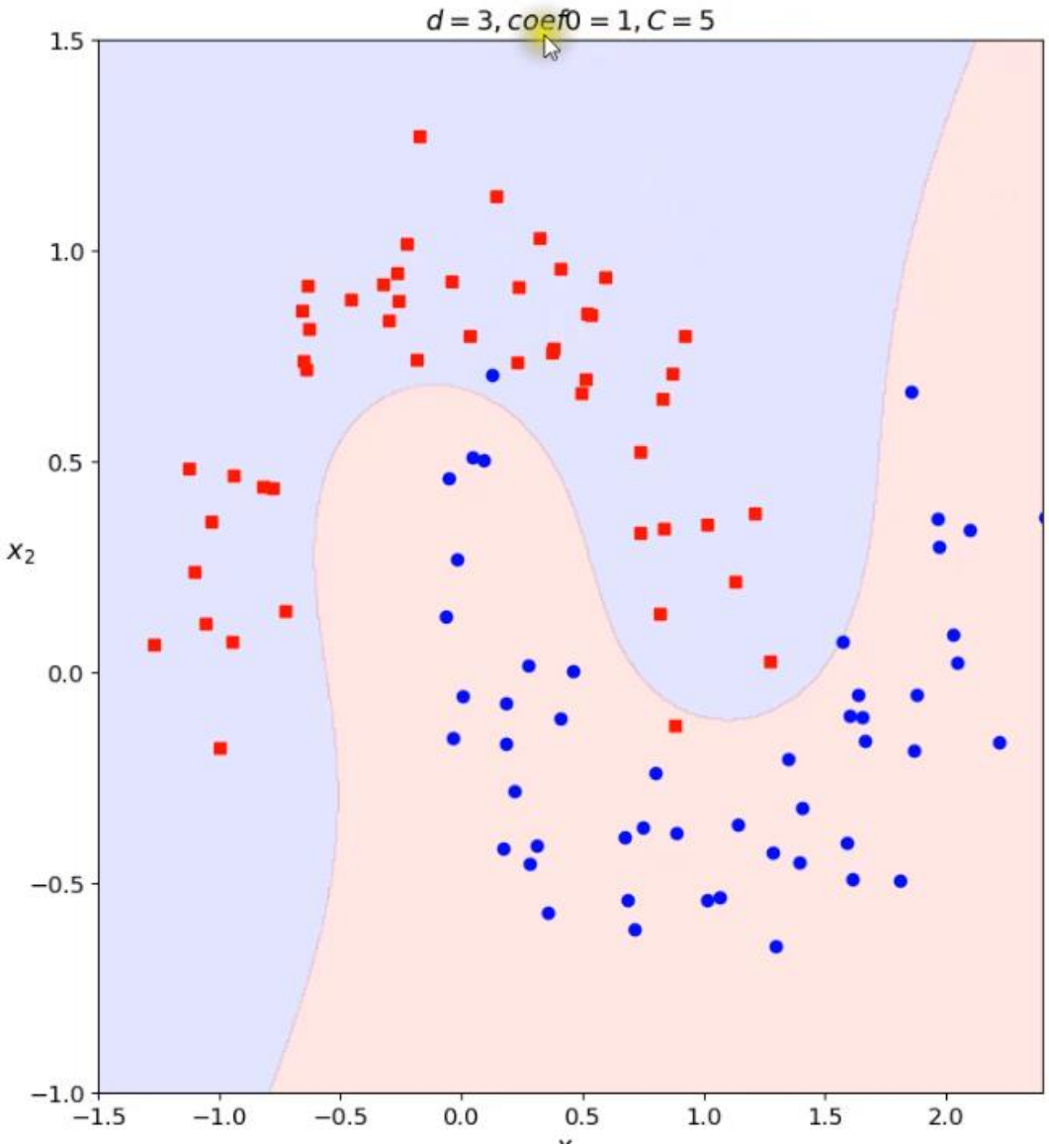
6.5.2. Kernel trick

Để khắc phục cho việc bậc cao của polynomial làm chậm model. Kernel trick cho kết quả tương tự polynomial mà không cần phải thêm features, đồng nghĩa với việc kernel trick chạy nhanh hơn polynomial nhiều.

Code:

```
poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))  ])
poly_kernel_svm_clf.fit(X, y)
```

Kết quả



6.5.3. Similarity Features

Một giải pháp nữa cho nonlinear SVM classification đó là Similarity features. Thêm features mới, features mới là khoảng cách của sample đó tới landmark

Ý tưởng: tìm một landmark, sau đó thêm features là khoảng cách của sample đó tới landmark

Gaussian RBF

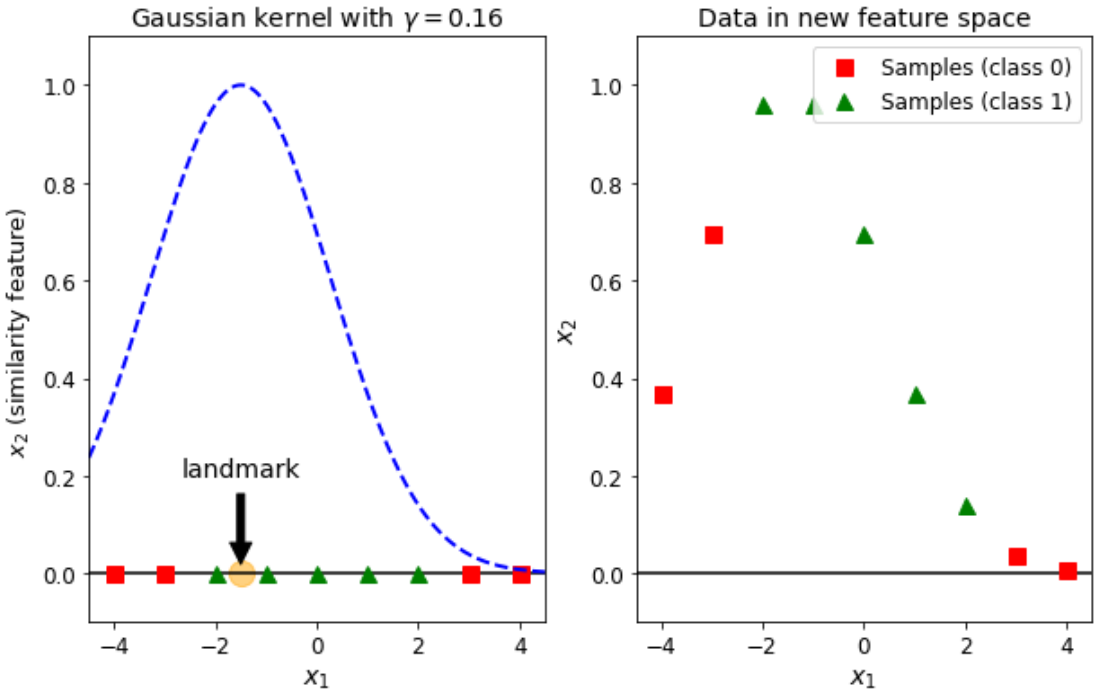
$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma\|\mathbf{x} - \ell\|^2\right)$$

Code:

Tạo ra bộ dữ liệu 1 chiều

```
X_1D = np.array([-4, -3, -2, -1, 0, 1, 2, 3, 4]).reshape(-1,1)
y = np.array([0, 0, 1, 1, 1, 1, 1, 0, 0]) # 2 classes
gamma = 0.16
landmark = np.array([-1.5])
def gaussian_rbf(x, landmark, gamma):
    return np.exp(-gamma * np.linalg.norm(x - landmark, axis=1)**2)
X_2D = np.c_[X_1D, gaussian_rbf(X_1D, landmark, gamma)]
```

Kết quả



6.6. SVM Regression

Ngoài classification thì SVM còn có thể dùng để regression. Trái với classification, SVM regression sẽ tìm model fit nhiều samples với margin nhỏ nhất có thể. SVM regression có một parameter gọi là epsilon. Epsilon càng thấp thì margin càng nhỏ. Trong trường hợp model bị overfitting thì ta nên tăng epsilon lên

Code:

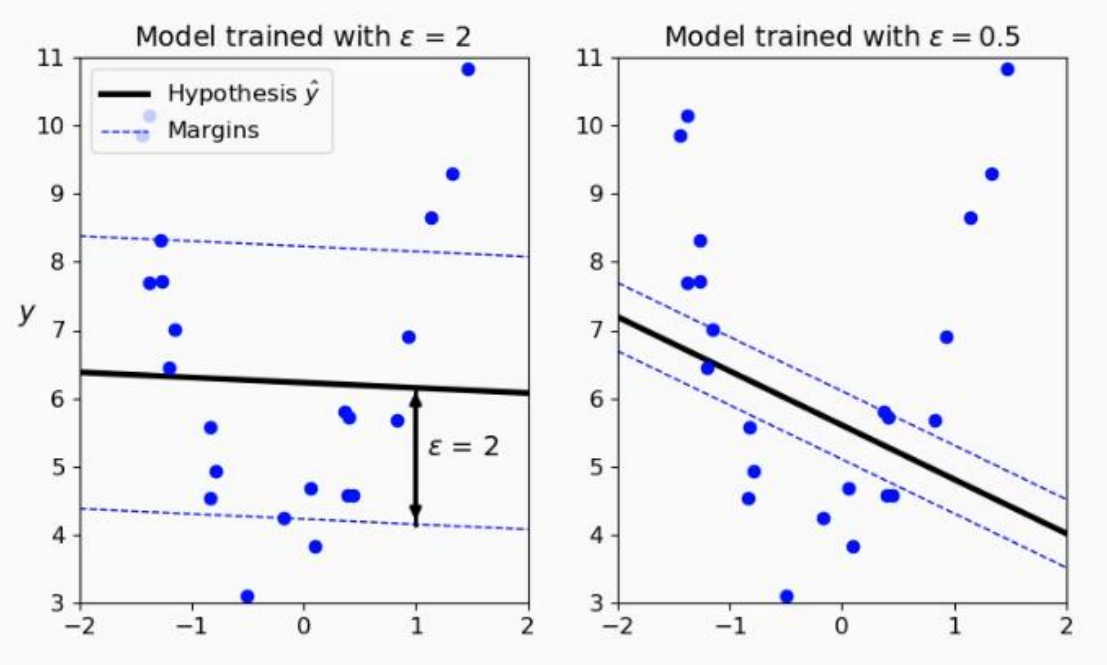
Tạo dữ liệu

```
np.random.seed(42)
m = 30
```

```
X = 4*np.random.rand(m, 1) -2
y = (4 + 3*X**2 + X + np.random.randn(m, 1)).ravel()

Huấn luyện SVM regression
from sklearn.svm import LinearSVR
svm_reg1 = LinearSVR(epsilon=2, random_state=42)
svm_reg1.fit(X, y)
svm_reg2 = LinearSVR(epsilon=0.5, random_state=42)
svm_reg2.fit(X, y)
```

Kết quả



6.6.1. Polynomial Regression

Tương tự như Polynomial Classification, đôi khi bộ dữ liệu không phải lúc nào cũng tuyến tính. Để khắc phục vấn đề này, ta sẽ dùng thuật toán polynomial, thêm features bậc cao và huấn luyện như linear regression.

Code:

```
svm_poly_reg1 = SVR(kernel="poly", degree=2, epsilon=0.2, C=0.01, gamma="scale")
svm_poly_reg1.fit(X, y)
```

svm_poly_reg2 = SVR(kernel="poly", degree=2, epsilon=0.2, C=1, gamma="scale")

svm_poly_reg2.fit(X, y)

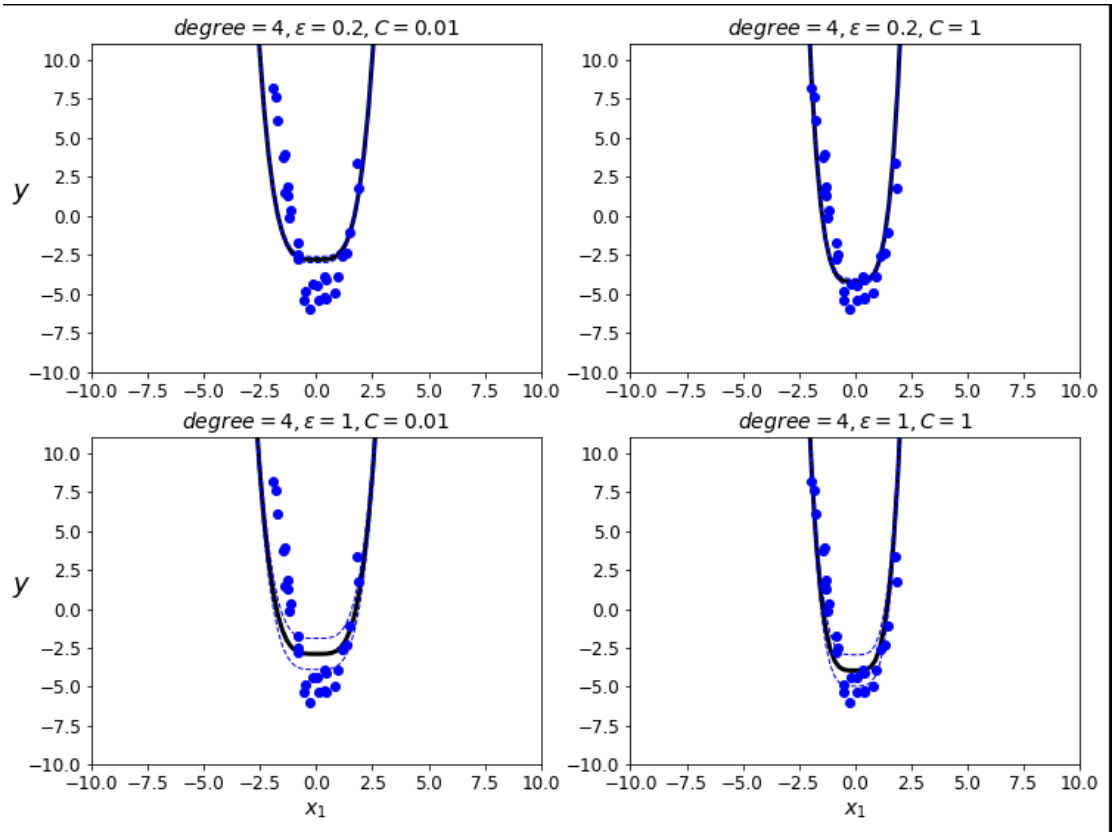
svm_poly_reg3 = SVR(kernel="poly", degree=2, epsilon=1, C=0.01, gamma="scale")

svm_poly_reg3.fit(X, y)

svm_poly_reg4 = SVR(kernel="poly", degree=2, epsilon=1, C=1, gamma="scale")

svm_poly_reg4.fit(X, y)

Kết quả



6.7. Math behind SVM

Hypothesis function

$$\hat{y} = \begin{cases} 0 \text{ if } \mathbf{xw} + b < 0 \\ 1 \text{ if } \mathbf{xw} + b \geq 0 \end{cases}$$

where $h(\mathbf{x}) = \mathbf{xw} + b$ is the **decision function**.

Primal problem là bài toán gốc, dual problem là vấn đề gần với primal problem, solution của dual problem là chặn dưới của solution của primal problem.

Giải dual problem thay vì giải trực tiếp primal problem vì đạt được kết quả gần giống như primal vì dual problem giải ra solution nhanh hơn, sử dụng được kernel trick trong bộ dữ liệu lớn hơn.

CHƯƠNG 7: DECISION TREE

7.1. Giới thiệu

Như SVM, Decision Tree là một thuật toán mạnh và linh hoạt trong Machine Learning mà có thể sử dụng cả trong classification và regression. Decision Tree có khả năng xử lý những bộ dữ liệu phức tạp.

7.2. Decision Tree Classification

Ý tưởng: Decision tree chia bộ dữ liệu thành nhiều node khác nhau

Mỗi node sẽ có 1 giá trị gọi là Gini(impurity) dùng để đo độ tinh khiết của node.

Dùng thuật toán CART algorithm để huấn luyện decision tree

+ Chọn 1 features k và threshold t_k (điểm để tách dữ liệu thành 2 bộ dữ liệu nhỏ)

+ Dừng lại khi đến độ sâu quy định và impurity không thể giảm thêm.

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

G Left/right để đo gini của subnet trái và phải

M left/right là số lượng samples của subnet trái và phải

Code:

Load dữ liệu

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target # ['SETOSA', 'VERSICOLOR', 'VIRGINICA']
```

Huấn luyện decision tree

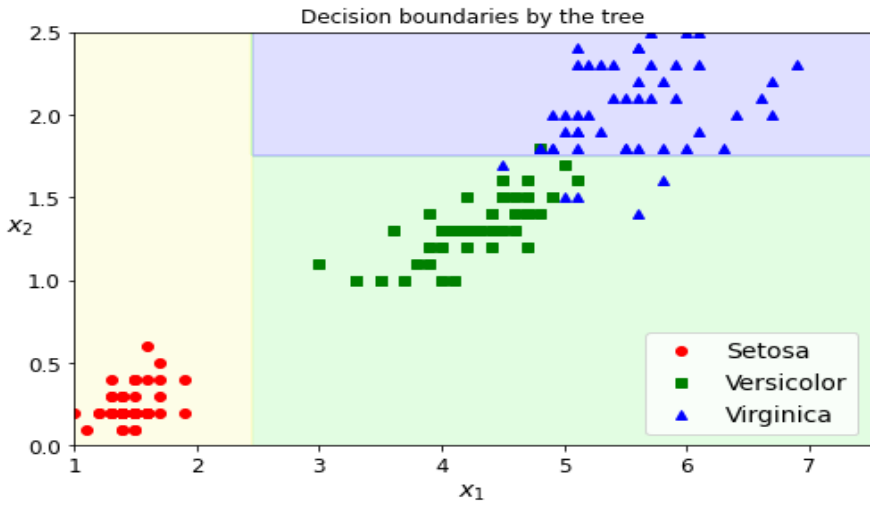
```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
# Try predicting class of a sample:
```

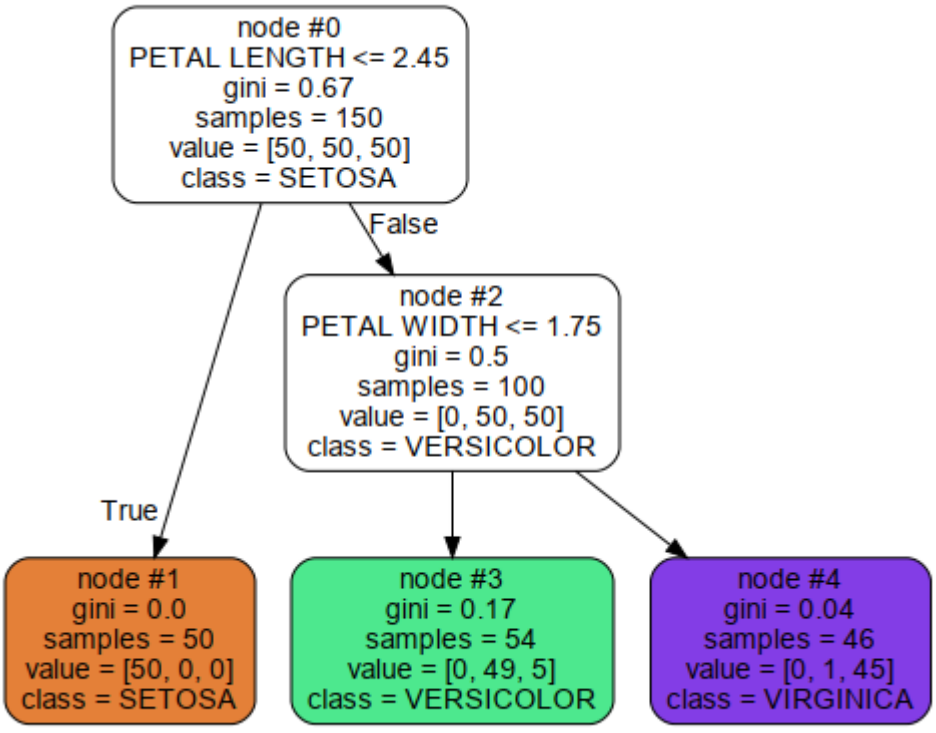
a_sample = [5, 2.5]

tree_clf.predict([a_sample])

tree_clf.predict_proba([a_sample]) # class probabilities

Kết quả:





7.3 Parametric và non-parametric models

Parametric: đã được xác định trước tham số của parameter trước khi huấn luyện

Non-parametric: chưa xác định trước tham số của parameter trước khi huấn luyện.

Decision tree là non-parametric models. Vì thế ta cần xác định các parameter của decision tree trước khi huấn luyện như:

+ Max_depth: độ sâu của tree

+ Min_samples_split: số lượng sample tối thiểu để tiếp tục tách

+ Min_sample_leaf: số lượng sample tối thiểu của một nút lá

+ Max_leaf_nodes: số lượng nút lá tối đa của một tree

+ Max_features: số lượng features sẽ dùng

Nếu decision tree bị overfitting, ta có thể điều chỉnh các parameter này để khắc phục vấn đề

Giảm max_depth

Tăng min_sample_split

Giảm max leaf node

Giảm max_feature

Tăng min_sample_leafe

7.4. Decision Tree Regression

Tương tự như decision tree classification, nhưng thay vì dự đoán từng class của mỗi node. Nó sẽ dự đoán value.

Ý tưởng: Model sẽ vẽ một cái cây đi qua các điểm của bộ dữ liệu

Ta vẫn sẽ dùng thuật toán CART để huấn luyện decision tree regression, nhưng sẽ thay đổi gini thành MSE

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \dfrac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

m left/right là số lượng samples của subnet trái và phải

MSE left/right là sai số trung bình của subnet trái, phải.

Code:

Tạo dữ liệu

np.random.seed(42)

m = 200

X = np.random.rand(m, 1)

y = 4*(X - 0.5)**2

y = y + np.random.randn(m, 1)/10

Huấn luyện decision tree regression

from sklearn.tree import DecisionTreeRegressor

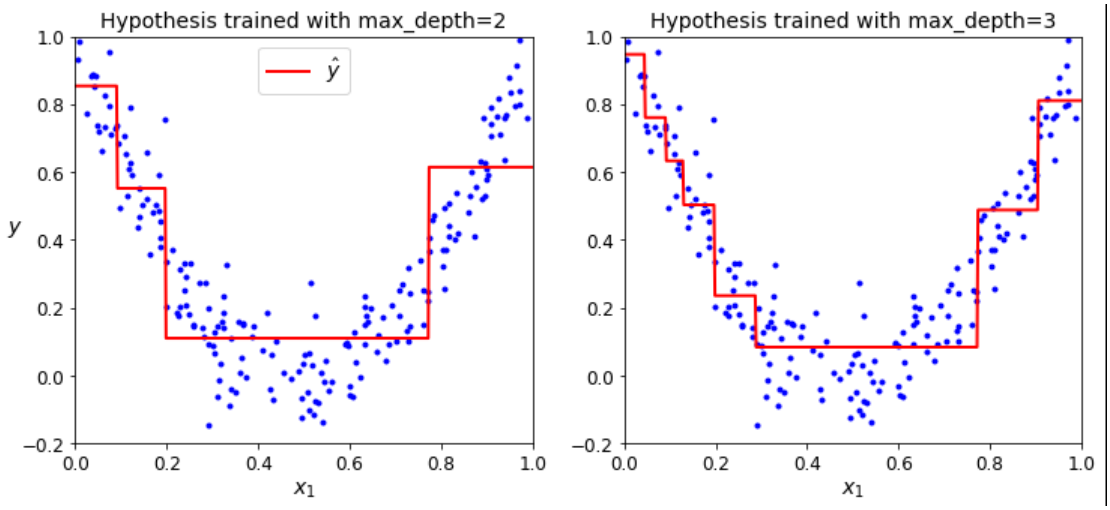
tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)

tree_reg1.fit(X, y)

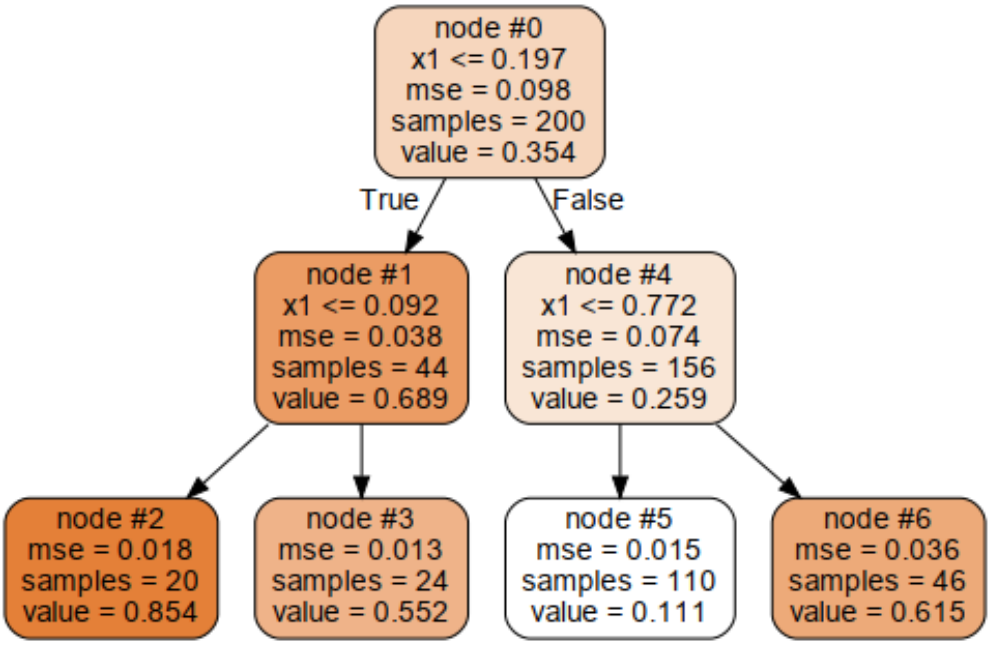
tree_reg2 = DecisionTreeRegressor(max_depth=3, random_state=42)

tree_reg2.fit(X, y)

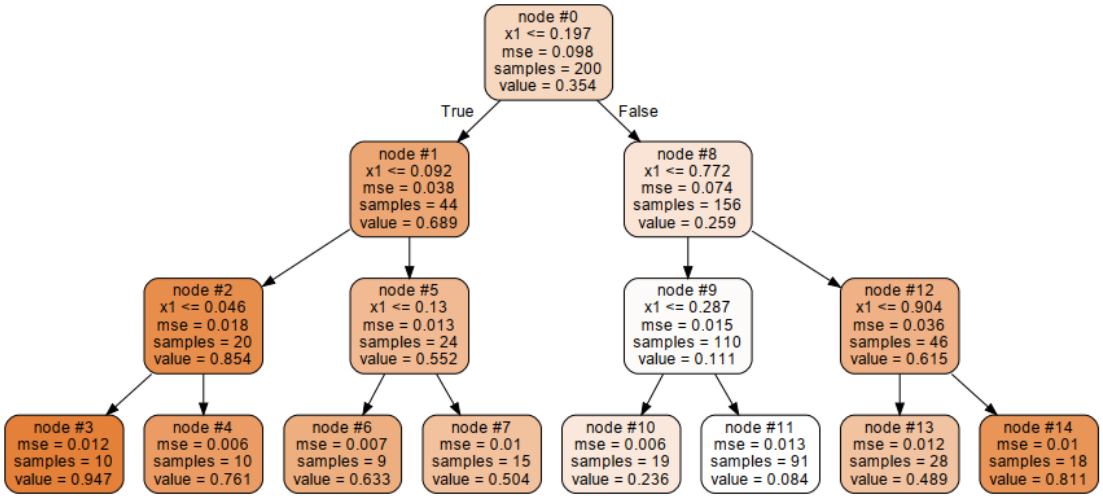
Kết quả:



Node của max_depth = 2



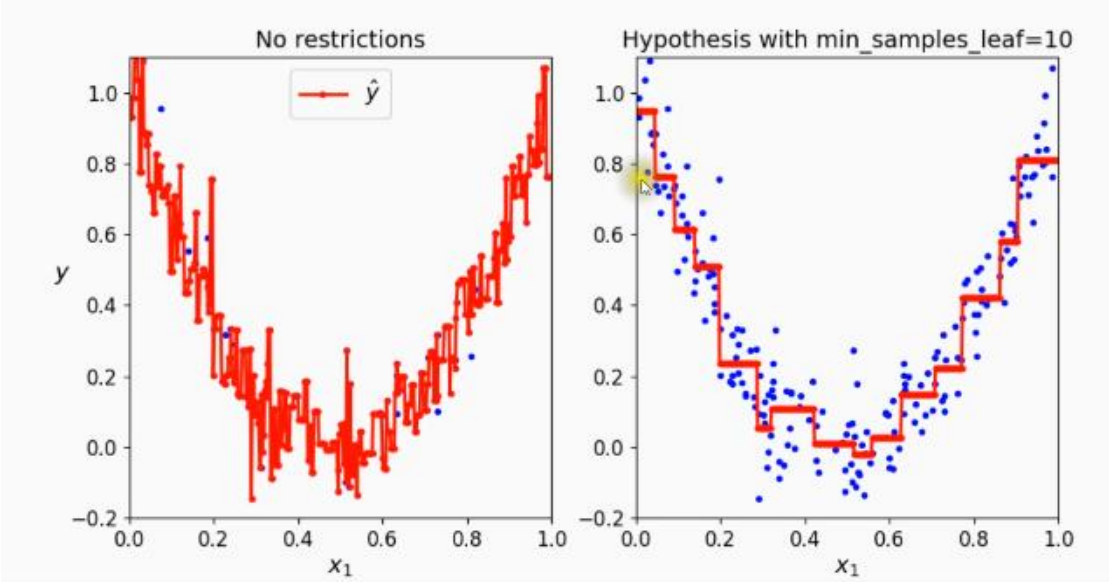
Node của max_depth = 3



7.5. Regularization

Hình dưới mô tả hình bên trái là khi huấn luyện không hạn chế và hình bên phải huấn luyện với regularization, cụ thể là kiểm soát số lượng sample tối thiểu của một nút lá.

Ta có thể dễ dàng thấy hình bên phải bị overfitting khi huấn luyện decision tree không có regularization



Vì thế regularization thì cần thiết khi huấn luyện decision tree. Huấn luyện decision tree với regularization khá dễ, ta chỉ cần chỉnh tham số của các parameter như max_depth, m, min_samples_split, min_sample_leaf, max_leaf_nodes, max_features.

Ví dụ: regularization decision tree khi điều chỉnh min_sample_leaf

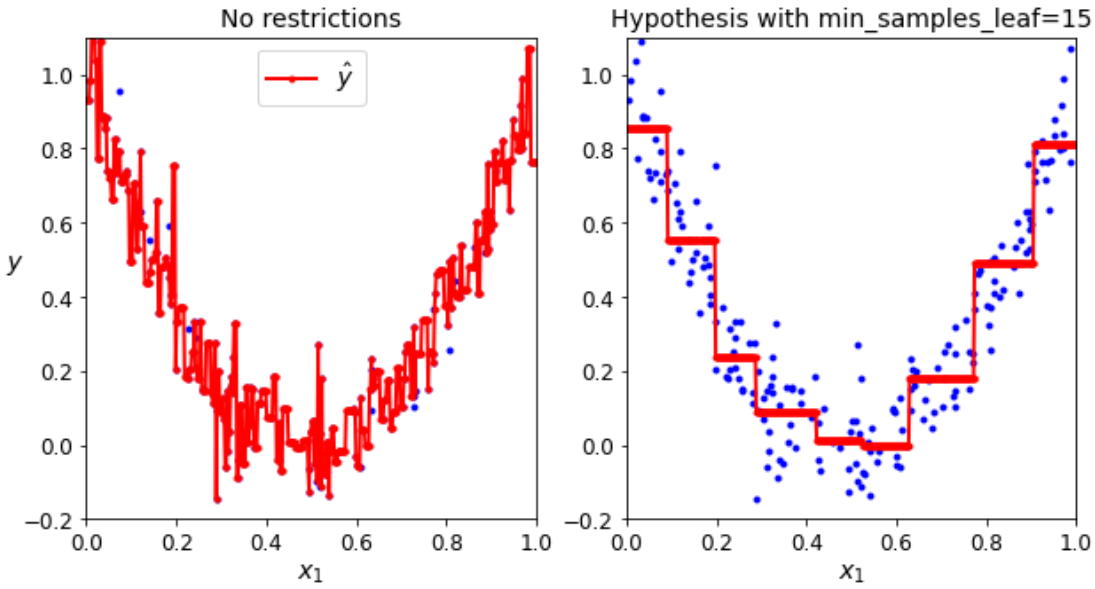
tree_reg1 = DecisionTreeRegressor(random_state=42)

tree_reg1.fit(X, y)

tree_reg2 = DecisionTreeRegressor(min_samples_leaf=15,random_state=42)

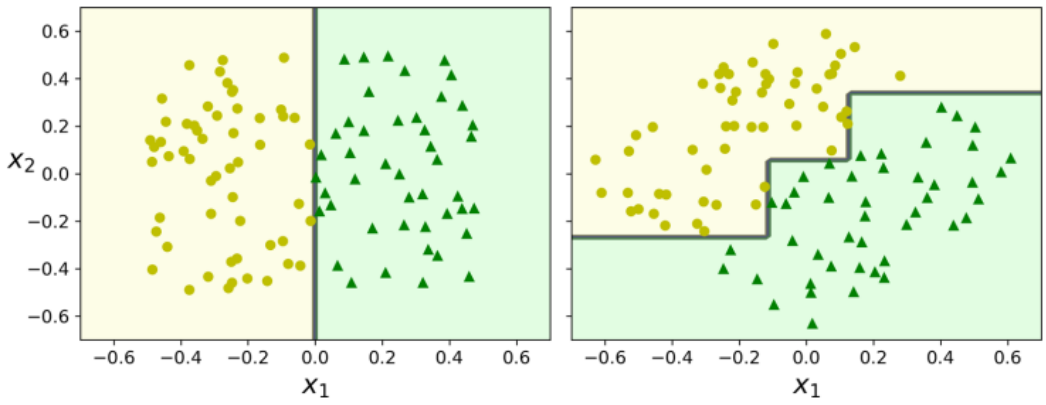
tree_reg2.fit(X, y)

Kết quả

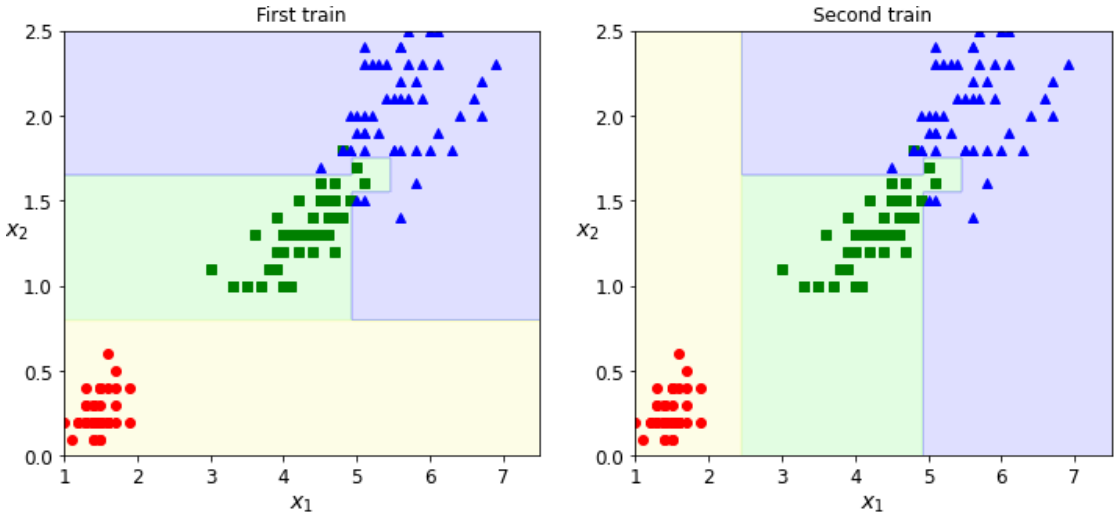


7.6. Không ổn định

Decision tree thì dễ hiểu, dễ cài đặt, linh hoạt và mạnh mẽ. Tuy nhiên decision tree có một vài nhược điểm đó là các decision boundary của decision tree thường song song hoặc vuông góc với trục tọa độ. Điều đó làm decision tree nhạy cảm với việc xoay dữ liệu do decision tree không biểu diễn được các đường chéo.



Và sau mỗi lần huấn luyện khác nhau thì model của decision tree cho 2 kết quả khác biệt.


First train
Second train
x_1
x_2
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7