CÁC THUẬT TOÁN TRONG MACHINE LEARNING

I. KNN

1. Hồi quy KNN

- Tiền xử lý – Scaling các đặc trưng

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
```

- Thông thường

```
from sklearn import neighbors
model = neighbors.KNeighborsRegressor(n_neighbors = K)
```

Lưu ý: K có thể bằng 1, 2,....

- Xử lý test

```
test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(test_scaled)
```

- Dự đoán

```
predict = model.predict(test)
```

- Tìm K tốt nhất

```
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
model.fit(x_train,y_train)
model.best_params_
```

2. Phân lớp KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
# Đọc dữ liệu train and test dataset : Các bạn có thể tha khảo hai cách đọc dữ liệu từ bài thực hành 7 và bài Naive Bayes
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')

# Mô hình của dữ liệu
print('Shape of training data :', train_data.shape)
print('Shape of testing data :', test_data.shape)

# Phân chia dữ liệu X, Y của tập train
train_x = train_data.drop(columns=['Cột cần xóa'],axis=1)
train_y = train_data['Cột dữ liệu dự đoán - Phân lớp']

# Phân chia dữ liệu X, Y của tập test
test_x = test_data.drop(columns=['Cột cần xóa'],axis=1)
test_y = test_data['Cột dữ liệu dự đoán - Phân lớp']

#Tạo mô hình dự đoán
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html


model = KNeighborsClassifier()

# fit the model với dữ liệu training
model.fit(train_x,train_y)

# Số lượng n_neighbors: N-láng giềng
print('\nThe number of neighbors used to predict the target : ', model.n_neighbors)

# Dự đoán lại với tập training
predict_train = model.predict(train_x)
```

```
print('\nTarget on train data',predict_train)

# Accuray Score trên tập dữ liệu train
accuracy_train = accuracy_score(train_y,predict_train)
print('accuracy_score on train dataset : ', accuracy_train)

# Dự đoán lại với tập test
predict_test = model.predict(test_x)
print('Target on test data',predict_test)

# Accuracy Score Accuray Score trên tập dữ liệu test
accuracy_test = accuracy_score(test_y,predict_test)
print('accuracy_score on test dataset : ', accuracy_test)
```

II. Hồi quy Logistic

1. Phân lớp

a) Binomial Logistic Regression: Hai lớp

```
LogisticRegression(penalty='l2', tol=0.0001, C=1.0, fit_intercept=True, class_weight=None, solver='lbfgs', max_iter=100)
```

Trong đó:

- **tot**: là giá trị bao dung (tolarance) để dừng cập nhật gradient descent nếu khoảng thay đổi của hàm mất mát sau một bước huấn luyện nhỏ hơn tot.
- **max_iter**: là số lượt huấn luyện tối đa.
- **fit_intercept**: để qui định có sử dụng trọng số tự do (chính là không bị phụ thuộc vào dữ liệu) hay không.
- **solver**: là phương pháp để giải bài toán tối ưu đối với cross entropy. Trong đó có: liblinear, sag, saga, newton-cg. Đối với dữ liệu kích thước nhỏ thì liblinear sử dụng sẽ phù hợp hơn. Trái lại sag, saga có tốc độ huấn luyện nhanh hơn cho dữ liệu lớn.
- **penalty**: là dạng hàm được sử dụng làm thành phần điều chuẩn (regularization term).
- **C**: Hệ số nhân của thành phần điều chuẩn.
- **class_weight**: Trọng số được nhân thêm ở mỗi nhóm. Thường được sử dụng trong trường hợp mẫu mất cân bằng giữa các nhóm để dự báo nhóm thiểu số tốt hơn. Trọng số có tác dụng điều chỉnh mức độ phạt nếu dự báo sai một mẫu theo nhãn ground truth của mẫu. Nếu không được xác định thì ta hiểu trọng số nhóm là cân bằng.

- Code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
completed_pl = Pipeline(
   steps=[
        ("preprocessor", preprocessor),
        ("classifier", LogisticRegression(penalty='l2', C=0.5, max_iter=200, class_weight=[0.3, 0.7]))
   ]
)

# training
completed_pl.fit(X_train, y_train)

# accuracy
y_train_pred = completed_pl.predict(X_train)
printf("Accuracy on train: {accuracy_score(list(y_train), list(y_train_pred)):.2f}")

y_pred = completed_pl.predict(X_test)
printf("Accuracy on test: {accuracy_score(list(y_test), list(y_pred)):.2f}")
```

b) Multinomial Logistic Regression: Hồi quy Logistic đa lớp

b.1) Đánh giá mô hình Hồi quy Logistic đa lớp

- Ví dụ 1: Truy vấn chéo với dữ liệu make_classification

```
# evaluate multinomial logistic regression model
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# define dataset
```

```
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, n_classes=3, random_state=1)
# define the multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
# define the model evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the model and collect the scores
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report the model performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

=> Kết quả: Mean Accuracy: 0.681 (0.042)

- Ví dụ 2: Đánh giá mô hình với dữ liệu make_classification

```
# make a prediction with a multinomial logistic regression model
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, n_classes=3, random_state=1)
# define the multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
# fit the model on the whole dataset
model.fit(X, y)
# define a single row of input data
row = [1.89149379, -0.39847585, 1.63856893, 0.01647165, 1.51892395, -3.52651223, 1.80998823, 0.58810926, -0.02542177, -0.52835426]
# predict the class label
yhat = model.predict([row])
# summarize the predicted class
print('Predicted Class: %d' % yhat[0])
# predict a multinomial probability distribution
yhat = model.predict_proba([row])
# summarize the predicted probabilities
print('Predicted Probabilities: %s' % yhat[0])
```

b.1) Tune Penalty (Điều chỉnh phạt) cho mô hình Hồi quy Logistic đa lớp

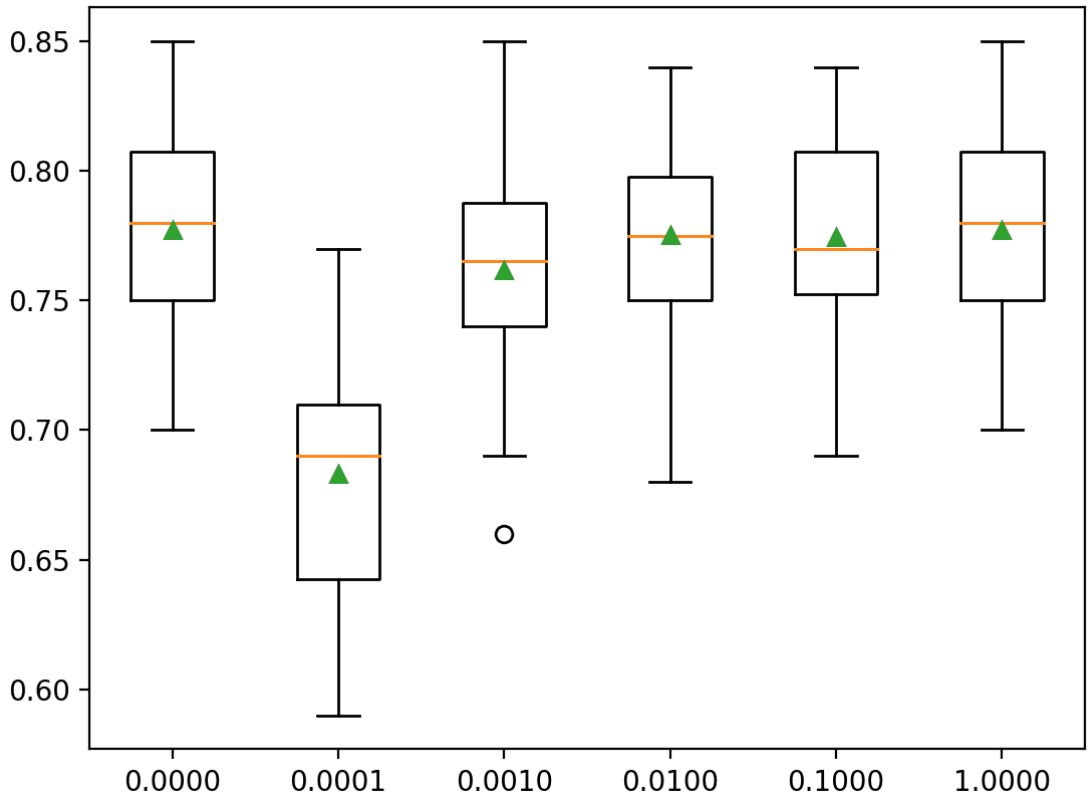
- Một siêu tham số quan trọng để điều chỉnh cho hồi quy logistic đa thức là thuật ngữ phạt (Penalty)
- Cấu trúc:

LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2', C=1.0)

- Code

```
# tune regularization for multinomial logistic regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot

# get the dataset
def get_dataset():
	X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1, n_classes=3)
	return X, y

# get a list of models to evaluate
def get_models():
	models = dict()
	for p in [0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]:
		# create name for model
		key = '%.4f' % p
		# turn off penalty in some cases
		if p == 0.0:
			# no penalty in this case
			models[key] = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='none')
		else:
```

```
                models[key] = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2', C=p)
        return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
        # define the evaluation procedure
        cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
        # evaluate the model
        scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
        return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
        # evaluate the model and collect the scores
        scores = evaluate_model(model, X, y)
        # store the results
        results.append(scores)
        names.append(name)
        # summarize progress along the way
        print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```



2. Hồi quy

- Hồi quy Logistic chỉ áp dụng cho bài toán phân lớp

III. Cây quyết định

1. Phân lớp

```
#Scaling các đặc trưng
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth = 3, min_samples_leaf = 5)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
```

- Tạo ma trận hỗn độn: Confusion matrix

```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

2. Hồi quy (Xem tuần 7)

```
DecisionTreeRegressor(*,
criterion='gini',
splitter='best',
max_depth=None,
min_samples_split=2,
min_samples_leaf=1,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None
)
```

- Trong đó:

- **criterion**: Là hàm số để đo lường chất lượng phân chia ở mỗi node. Có hai lựa chọn là gini và entropy.
- **max_depth**: Độ sâu tối đa cho một cây quyết định. Đối với mô hình bị quá khớp thì chúng ta cần giảm độ sâu và vị khớp thì gia tăng độ sâu.
- **min_samples_split**: Kích thước mẫu tối thiểu được yêu cầu để tiếp tục phân chia đối với node quyết định. Được sử dụng để tránh kích thước của node lá quá nhỏ nhằm giảm thiểu hiện tượng quá khớp.
- **max_features**: Số lượng các biến được lựa chọn để tìm kiếm ra biến phân chia tốt nhất ở mỗi lượt phân chia.
- **max_leaf_nodes**: Số lượng các node lá tối đa của cây quyết định. Thường được thiết lập khi muốn kiểm soát hiện tượng quá khớp.
- **min_impurity_decrease**: Chúng ta sẽ tiếp tục phân chia một node nếu như sự suy giảm của độ tinh khiết nếu phân chia lớn hơn ngưỡng này. im
- **min_impurity_split**: Ngưỡng dừng sớm để kiểm soát sự gia tăng của cây quyết định. Thường được sử dụng để tránh hiện tượng quá khớp. Chúng ta sẽ tiếp tục chia node nếu độ tinh khiết cao hơn ngưỡng này.

IV. Rừng ngẫu nhiên

1. Phân lớp

```
RandomForestClassifier(*,
n_estimators=100,
criterion='gini',
max_depth=None,
min_samples_split=2,
min_samples_leaf=1,
```

```
    max_features=None,
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    max_samples=None
    )
```

- Bản chất của mô hình rừng cây là sự kết hợp giữa nhiều cây quyết định được huấn luyện theo phương pháp lấy mẫu tái lập. Do đó các tham số của nó sẽ lấy các tham số thiết lập cây quyết định từ class DecisionTreeClassifier và tham số tạo mẫu dữ liệu từ BaggingClassifier. Đối với các tham số của mô hình cây quyết định, các bạn có thể xem giải thích ý nghĩa tại tuning siêu tham số cho mô hình cây quyết định. Ba tham số cho thiết lập mẫu dữ liệu bao gồm n_estimators, bootstrap, oob_score và max_samples.

- **Trong đó:**

- **n_estimators** là số lượng các cây quyết định được sử dụng trong mô hình rừng cây.
- **bootstrap=True** tương ứng với sử dụng phương pháp lấy mẫu tái lập khi xây dựng các cây quyết định. Trái lại thì chúng ta sử dụng toàn bộ dữ liệu. Lưu ý rằng ở đây do chúng ta đã lựa chọn ngẫu nhiên ra biến khi xây dựng cây quyết định nên dù sử dụng chung một tập dữ liệu thì các cây quyết định vẫn khác nhau.
- **oob_score** chỉ có hiệu lực khi sử dụng khi sử dụng boostrap. Nếu oob_score=True thì sẽ tính toán thêm điểm số trên các mẫu nằm ngoài túi.
- **max_samples** là số lượng mẫu được sử dụng để huấn luyện mô hình cây quyết định. Mặc định max_samples=None thì chúng ta lấy ra các mẫu con có kích thước bằng với tập huấn luyện.

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# Huấn luyện mô hình trên tập train
rdf_clf = RandomForestClassifier(
   max_depth = 3,
   max_leaf_nodes = 16,
   min_samples_split = 10,
   min_samples_leaf = 10
)
```

```
rdf_clf.fit(X_train, y_train)


# Dự báo trên tập test
y_pred = rdf_clf.predict(X_test)
scores = accuracy_score(y_pred, y_test)
print('RandomForest Accuracy: {:.03f}'.format(scores))
```

2. Hồi quy (Xem tuần 7)

V. Naive Bayes

Lưu ý: Naive Bayes chỉ có bài toán phân lớp, không có hồi quy

1. Gaussian Naive Bayes

- Thông thường mô hình Gaussian Naive Bayes sẽ áp dụng trên dữ liệu đầu vào là những biến liên tục. Để xây dựng mô hình Gaussian Naive Bayes thì trong sklearn thì chúng ta sử dụng class sklearn.naive_bayes.GaussianNB. Bên dưới chúng ta sẽ thực hành huấn luyện mô hình này trên bộ dữ liệu iris.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB


X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)


gnb = GaussianNB()
gnb.fit(X_train, y_train)


y_pred = gnb.predict(X_test)
print(classification_report(y_pred, y_test))
```

2. Multinomial Naive Bayes

a) Bộ dữ liệu

- Bộ dữ liệu huấn luyện được trích lọc từ fetch_20newsgroups. fetch_20newsgroups bao gồm 20 chủ đề khác nhau. Tuy nhiên ở đây ta chỉ lấy ra 1183 văn bản thuộc hai chủ để là cơ đốc giáo có nhãn soc.religion.christian và đồ hoạ máy tính có nhãn comp.graphics.

```
from sklearn.datasets import fetch_20newsgroups


# Download bộ dữ liệu phân loại văn bản gồm 2 chủ đề  tôn giáo: 'soc.religion.christian', 'comp.graphics'].
categories = ['soc.religion.christian', 'comp.graphics']
twenty_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=42)
print('Total document: {}'.format(len(twenty_train.data)))
```

- Nội dung của object twenty_train sẽ bao gồm dữ liệu là văn bản được chứa trong list twenty_train.data và nhãn được chứa trong list twenty_train.target.

```
for i in range(5):
  print('----------------------------------> \n')
  print('Content: {} ...'.format(twenty_train.data[i][:100]))
  print('Target label: {}'.format(twenty_train.target[i]))
```

- Chúng ta không thể đưa trực tiếp dữ liệu là văn bản vào để huấn luyện mô hình mà cần phải mã hoá chúng dưới dạng véc tơ.

- Trong sklearn để xử lý văn bản, mã hoá kí tự sang số (tokenizing) và lọc bỏ từ dừng (stopwords) chúng ta hoàn toàn có thể sử dụng sklearn.feature_extraction.text.CountVectorizer. Class này sẽ giúp xây dựng một từ điển và biến đổi một văn bản thành một véc tơ đặc trưng theo tần suất xuất hiện các từ trong từ điển.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
X_train_counts.shape
```

- Ma trận X_train_counts thu được là một ma trận tần suất có số dòng bằng số lượng văn bản và số cột bằng kích thước của từ điển. Mỗi một dòng là một véc tơ tần suất xuất hiện của các từ trong từ điển. Những tần suất này được sắp xếp theo thứ tự khớp với thứ tự của các từ mà nó thống kê trong từ điển.

b) Huấn luyện mô hình Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
```

```
import numpy as np

# Khởi tạo mô hình
mnb_clf = MultinomialNB()

# Cross-validation với số K-Fold = 5 và thực hiện 1 lần cross-validation
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=1)
scores = cross_val_score(mnb_clf, X_train_counts, twenty_train.target, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: {:.03f}, Standard Deviation Accuracy: {:.03f}'.format(np.mean(scores), np.std(scores)))
```

- Độ chính xác đạt được 98.9% là rất cao, đồng thời độ biến động của độ chính xác chỉ 0.6% khi thực hiện cross-validation. Điều đó cho thấy mô hình Multinomial Naive Bayes khá hiệu quả trong tác vụ phân loại văn bản.

c) Dự báo cho một văn bản mới

- Để dự báo cho một nội dung văn bản mới chúng ta cần phải trải qua hai bước:

+ B1: Mã hoá văn bản sang véc tơ tần suất.

+ B2: Dự báo trên véc tơ tần suất.

- Tất cả những bước này được thực hiện khá dễ dàng trên sklearn.

```
# Mã hoá câu văn sang véc tơ tần suất
docs_new = ['God is my love', 'OpenGL on the GPU is fast']
X_new_counts = count_vect.transform(docs_new)

# Dự báo
mnb_clf.fit(X_train_counts, twenty_train.target)
predicted = mnb_clf.predict(X_new_counts)

for doc, category in zip(docs_new, predicted):
  print('%r => %s' % (doc, twenty_train.target_names[category]))
```

VI. Hồi quy tuyến tính (Linear regression)

- Hồi quy tuyến tính chỉ áp dụng cho bài toán hồi quy

- Xem tuần 7

VII. SVM

1. Hồi quy (SVR)

- Xem tuần 7

2. Phân lớp (SVC)

- sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True, probability=False,tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, random_state=None)

- kernel có thể là: “linear”, “rbf”, “sigmoid” hoặc ”poly”

- Code:

```
from sklearn import svm
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
support = svm.LinearSVC(random_state=20)

# Train the model using the training sets and check score on test dataset
support.fit(train_x, train_y)
predicted= support.predict(test_x)
score=accuracy_score(test_y,predicted)
print("Your Model Accuracy is", score)
```

a) Ví dụ về bài toán SVM

- Tiếp theo chúng ta sẽ cùng sử dụng SVM để phân loại bộ dữ liệu iris.

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()

X = iris["data"]
```

```
y = (iris["target"] == 2).astype(np.int8) # 1 if virginica, 0 else
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
svm_pl = Pipeline((
   ("scaler", StandardScaler()),
   ("linear_svc", SVC(C=1, kernel="linear", probability = True))
  )
)
svm_pl.fit(X, y)
scores = cross_val_score(svm_pl, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: {:.03f}, Standard Deviation Accuracy: {:.03f}'.format(np.mean(scores), np.std(scores)))
```

=> Kết quả: Mean Accuracy: 0.960, Standard Deviation Accuracy: 0.050

- Dự báo cho một quan sát mới

```
# Dự báo nhãn
svm_pl.predict(np.array([[1.2, 3.3, 2.2, 4.5]]))
array([1], dtype=int8)
# Dự báo xác suất, chỉ được khi probability trong SVC() được set True.
svm_pl.predict_proba(np.array([[3.2, 3.0, 4.2, 4.5]]))
array([[3.0000009e-14, 1.0000000e+00]])
```

b) Bài toán SVM cho dữ liệu dạng phi tuyến

- Mặc dù SVM có kết quả khá tốt cho bài toán phân loại nhưng có một số tình huống dữ liệu là phức tạp và yêu cầu chúng ta phải thực hiện các phép biến đổi phi tuyến đối với biến đầu vào để tạo thành những đường biên phức tạp hơn.

- Kỹ thuật chuẩn hoá đa thức (polynormial) được áp dụng để tạo ra những biến bậc cao sẽ hữu ích trong những tình huống này:

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
svm_ply_pl = Pipeline((
   ("poly_features", PolynomialFeatures(degree=3)),
   ("scaler", StandardScaler()),
   ("linear_svc", SVC(C=1, kernel="linear", probability = True))
  )
)
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(svm_ply_pl, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: {:.03f}, Standard Deviation Accuracy: {:.03f}'.format(np.mean(scores), np.std(scores)))
```

⇨ Kết quả: Mean Accuracy: 0.969, Standard Deviation Accuracy: 0.041

- Như vậy sau khi áp dụng chuẩn hoá đa thức thì độ chính xác đã tăng lên từ 0.96 lên 0.969. Đây là một trong những kỹ thuật thường được áp dụng để giúp cải thiện độ chính xác cho SVM.

- Trên thực thế thì kỹ thuật chuẩn hoá đa thức cũng tương tự như việc sử dụng kernel poly trong module SVC. Lưu ý rằng mặc dù kỹ thuật chuẩn hoá đa thức thường mang lại sự cải tiến đáng kể về độ chính xác cho mô hình nhưng số lượng biến mà nó tạo ra bao gồm những biến tích chéo (dạng $x_1^p x_2^q$) và biến bậc cao (dạng x_1^l) là rất lớn.

- Do đó sẽ dễ xảy ra hiện tượng quá khớp và đồng thời gia tăng chi phí huấn luyện và tính toán => Tiếp theo ta sẽ thực hành tuning kernel trong SVM.

c) Sử dụng kernel SVM

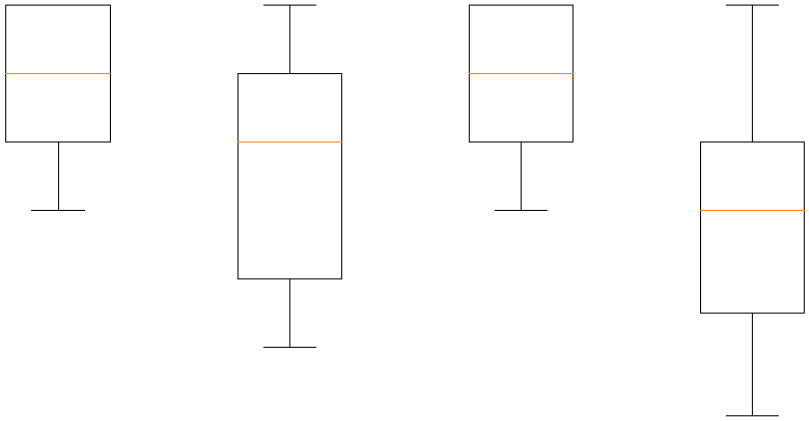
- Khi huấn luyện mô hình SVM chúng ta cần thử với nhiều kernels khác nhau để tìm ra kernel tốt nhất cho bộ dữ liệu huấn luyện. Các kernel phổ biến đó là: linear, poly, rbf, sigmoid như đã được giới thiệu ở trên.

- Ngoài ra nếu mô hình gặp hiện tượng quá khớp thì chúng ta cần điều chỉnh giảm hệ số của mô hình SVM để gia tăng ảnh hưởng của thành phần kiểm soát.

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

kernels = ['linear', 'poly', 'rbf', 'sigmoid']
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
all_scores = []
# Đánh giá toàn bộ các mô hình trên tập K-Fold đã chia
for kernel in kernels:
  svm_kn_pl = Pipeline((
     ("scaler", StandardScaler()),
     ("linear_svc", SVC(C=1, kernel=kernel, probability = True))
   )
  )
  scores = cross_val_score(svm_kn_pl, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

```
  all_scores.append((kernel, scores))
import matplotlib.pyplot as plt
# Draw bboxplot
plt.figure(figsize=(16, 8))
plt.boxplot([score[1] for score in all_scores])
plt.xlabel('Scale', fontsize=16)
plt.ylabel('cm', fontsize=16)
plt.xticks(np.arange(len(kernels))+1, kernels, rotation=45, fontsize=16)
plt.title("Scores Metrics", fontsize=18)
```

⇨ Kết quả Text(0.5, 1.0, 'Scores Metrics')



- Như vậy ta có thể thấy các kernel hiệu quả chính là rbf và linear khi cùng có giá trị trung vị vào khoảng 0.97 và cao hơn mức trung bình của kernel kém nhất là Sigmoid là 0.07 điểm. Đây là một mức cải thiện khá đáng kể cho một bài toán phân loại nhị phân.

d) tuning siêu tham số cho một kernel

- Đối với mỗi một dạng hàm kernel, căn cứ vào phương trình của chúng ta có thể xác định được những siêu tham số cần tuning.

- Chẳng hạn như đối với danh sách các kernel được cung cấp ở mục 5.1 thì chúng ta có thể tuning các tham số như sau:

- kernel tuyến tính: tham số C.
- kernel đa thức: tham số C, γ (coef0), d (degree).
- kernel RBF: tham số C, γ.
- kernel sigmoid: tham số C, γ, d.

- Công thức tổng quát của một mô hình SVC:

```
sklearn.svm.SVC(*,
  C=1.0,
  kernel='rbf',
  degree=3,
```

```
    gamma='scale',
    coef0=0.0,
    class_weight=None,
    decision_function_shape='ovr',
    random_state=None
)
```

- Trong class SVC của sklearn thì hệ số γ tương ứng với đối số coef0, hệ số bậc đa thức d là đối số degree, trọng số C của hàm chi phí chính là đối số C và loại kernel là đối số kernel.

- Ngoài ra trong trường hợp mẫu bị mất cân bằng nghiêm trọng thì chúng ta thiết lập **class_weight** để phạt nặng hơn những trường hợp mẫu thiểu số.

- **decision_function_shape** là đối số cho phép chúng ta cấu hình kết quả xác suất dự báo trả về là theo phương pháp **one-vs-rest** hay **one-vs-one**.

- Nếu theo phương pháp one-vs-rest thì mô hình phân loại gồm C nhãn sẽ được chia thành C bài toán phân loại con, mỗi một bài toán tương ứng với một dự báo xác suất thuộc về nhãn i.
- Còn đối với bài toán one-vs-one chúng ta sẽ tìm cách xây dựng C x (C-1) mô hình phân loại cho một cặp nhãn (i, j) bất kỳ.
- Đối với bài toán phân loại nhị phân thì **decision_function_shape** = 'ovr' tương ứng với dự báo xác suất tương ứng với nhãn (0, 1)

- Bên dưới là một ví dụ mẫu về cách tuning tham số trên GridSearch đối với mô hình SVM.

```
from sklearn.model_selection import GridSearchCV
parameters = {
   'clf__kernel':['linear', 'rbf', 'poly', 'sigmoid'],  # Các dạng hàm kernel
   'clf__C':[0.05, 1, 100], # Trọng số của phạt phân loại sai
   'clf__coef0': [2, 4], # Tương ứng với tham số gamma của đa thức
   'clf__degree': [1, 2, 3] # Bậc d của đa thức
}



pipeline = Pipeline(
   steps=[("clf", SVC())]
)
gscv = GridSearchCV(pipeline, parameters, cv=5, n_jobs=12, scoring='accuracy', return_train_score=True, error_score=0, verbose=3)
gscv.fit(X, y)
```

-

VIII. Hồi quy Đa thức (Polynomial Regression)

- Hồi quy Đa thức chỉ áp dụng cho bài toán hồi quy
- Xem tuần 7

VIII. Hồi quy Logistic Đa thức (Polynomial Logistic Regression)

- Ví dụ 1:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape
poly = PolynomialFeatures(degree = 2, interaction_only=False, include_bias=False)
X_poly = poly.fit_transform(X_train)
X_poly.shape
lr = LogisticRegression()
lr.fit(X_poly,y_train)
lr.score(poly.transform(X_test), y_test)
pipe = Pipeline([('polynomial_features',poly), ('logistic_regression',lr)])
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Ví dụ 2:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

y = np.array([[1,0,1,0,1],[0,1,0,1,0]]).transpose()

num_features = 2
```

```
x = np.arange(1,num_features * y.shape[0] + 1).reshape((num_features,y.shape[0])).transpose() / 10

plt.figure()
plt.plot(x[np.where(y[:,0] == 1)[0],0], x[np.where(y[:,0] == 1)[0],1], 'b+')
plt.plot(x[np.where(y[:,0] == 0)[0],0], x[np.where(y[:,0] == 0)[0],1], 'ro')
plt.title('θ1 Vs θ2');

def spline(x):

    x1 = (x[:,0] - x[:,1]).reshape((1,x.shape[0])).transpose()
    x2 = (x[:,0] - np.power(x[:,1],2)).reshape((1,x.shape[0])).transpose()
    x3 = (x[:,0] - np.power(x[:,1],3)).reshape((1,x.shape[0])).transpose()

    return np.concatenate((x1,x2,x3), axis=1)

s = lambda x: (x-np.mean(x,axis=0))/(np.max(x,axis=0)-np.min(x,axis=0))
g = lambda z: 1 / (1 + np.exp(-z))
h = lambda x, theta: g(x @ theta);
p = lambda x, theta: np.argmax(g(Xs @ theta), axis=1)

def gradient_descent_regularization(Xs,y,h,lp,alpha,num_iter):

    num_k = y.shape[1]
    num_features = Xs.shape[1]
    theta = np.zeros((num_features,num_k))
    J_history = np.zeros((num_iter,num_k))
    for k in range(num_k):

        k_y = y[:,k].reshape(1,len(y)).transpose()
        k_theta = np.zeros((num_features,1))
        for i in range(num_iter):
```

```
        J_reg = lp*(k_theta[1:].transpose() @ k_theta[1:])/(2*len(k_y))
        J = np.sum((-k_y*np.log(h(Xs,k_theta)) - (1-k_y)*np.log(1-h(Xs,k_theta)))/len(k_y)) + J_reg

        grad_reg = lp*k_theta/len(y)
        grad_reg[0] = 0
        grad = (Xs.transpose() @ (h(Xs,k_theta)-k_y))/len(k_y) + grad_reg

        k_theta = k_theta - alpha*grad

        J_history[i,k] = J

      theta[:,k] = k_theta[:,0]

   return (theta, J_history)

lp = 1e-7
alpha = 7.1
num_iter = 5000

print('Spline: y = θ0 + θ1 (x1 - x2) + θ2 (x1 - x2^2) + θ3 (x1 - x2^3)')
Xm = spline(x)
print('X model')
print(Xm.shape)
print(Xm)

Xs = np.concatenate((np.ones((y.shape[0],1)), s(Xm)), axis=1)

theta, J_history = gradient_descent_regularization(Xs,y,h,lp,alpha,num_iter)

print('Predictions')
print(p(Xs,theta))
print('Expected')
```

```
print(y[:,1])
print('Success rate %')
print(np.mean(y[:,1] == p(Xs,theta)) * 100)

plt.figure()
for k in range(y.shape[1]):
    plt.plot(J_history[:,k])
plt.title('J(θ)');

plt.figure()
plt.plot(x[np.where(y[:,0] == 1)[0],0], y[np.where(y[:,0] == 1)[0]], 'b+')
plt.plot(x[np.where(y[:,0] == 0)[0],0], y[np.where(y[:,0] == 0)[0]], 'ro')
plt.plot(x[:,0], h(Xs,theta[:,0]), 'b-')
plt.plot(x[:,0], h(Xs,theta[:,1]), 'r-')
plt.title('Training data y = f(θ1)')
```