Loughborough University

COP528 Applied Machine Learning (LABs)

LAB Day 06. Support Vector Machine & kNN

Introduction

The aim of this session is to understand SVM and kNN. The experiment includes **1** tutorial example and **3** tasks. Please follow the tutorial to learn the basic functions and then complete the tasks.

Tutorial 01. Understand the SVM code

Run and understand the code below how to use SVM to make the classification. If you have two features for the task, you could plot the class boundary by using the following code.

(1) Load a dataset of two features for the task

```
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

iris = datasets.load_iris()# Select 2 features / variables
X = iris.data[:, :2] # we only take the first two features.
y = iris.target
feature_names = iris.feature_names[:2]
classes = iris.target_names
```

(2) Functions for drawing the boundaries of a class.

```
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

(3) Classification and visualisation with SVM and kNN.

```
# SVM
model = svm.SVC(kernel="poly", degree=3)
clf = model.fit(X, y)
fig, ax = plt.subplots()
# title for the plots
title = ('Decision surface of Polynomial Degree 3 SVC')
# Set-up grid for plotting.
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_ylabel("{}".format(feature_names[0]))
ax.set_xlabel("{}".format(feature_names[1]))
ax.set_xticks(())
ax.set_yticks(())
ax.set_title(title)
plt.show()
print(f'SVM accuracy_score:{accuracy_score(y, clf.predict(X))}')

# kNN
clf_kNN = neighbors.KNeighborsClassifier(n_neighbors=3, weights='distance')
clf_kNN.fit(X, y)
y_pred=clf_kNN.predict(X)
fig, ax = plt.subplots()
# title for the plots
title = ('Decision surface of kNN')
# Set-up grid for plotting.
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

plot_contours(ax, clf_kNN, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_ylabel("{}".format(feature_names[0]))
ax.set_xlabel("{}".format(feature_names[1]))
ax.set_xticks(())
ax.set_yticks(())
ax.set_title(title)
plt.show()
print(f'kNN accuracy_score:{accuracy_score(y, y_pred)}')
```

Task 01. SVM experiment with changing the kernel

Please read the manual of SVM (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html), change the kernel to RBF and Linear to find the classification boundaries.

*Solutions Example

```
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

iris = datasets.load_iris()# Select 2 features / variables
X = iris.data[:, :2] # we only take the first two features.
y = iris.target
print(X[0:5])
feature_names = iris.feature_names[:2]
classes = iris.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.3,
                                    random_state=0)
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy
def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
# The classification SVC model
for kernel in ['linear','poly','rbf']:
    model = svm.SVC(kernel=kernel, degree=3)
    clf = model.fit(X_train, y_train)
    fig, ax = plt.subplots()
    # title for the plots
    title = (f'Decision surface of SVM with {kernel}')
    # Set-up grid for plotting.
    X0, X1 = X[:, 0], X[:, 1]
    xx, yy = make_meshgrid(X0, X1)
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_ylabel("{}".format(feature_names[0]))
    ax.set_xlabel("{}".format(feature_names[1]))
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    plt.show()
    pred_test = model.predict(X_test)
    test_f1 = metrics.f1_score(y_test, pred_test, average='macro')
    test_acc = metrics.accuracy_score(y_test, pred_test)
    print(f'{kernel} test F1:{test_f1}, test acc:{test_acc}')
```

Task 02. SVM experiment of wine dataset

Use SVM with several popular kernels to work on the wine dataset. Please review the method of splitting the dataset into a training set, a validation set and a test set in Day2 (hint: Day02-tutorial06). Please use threefold cross-validation to evaluate the classification effect.

(1) Loading the dataset.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm, datasets

wine = load_wine()
X = wine.data
y = wine.target
```

*Solutions Example

```
from sklearn.datasets import load_wine
import numpy as np
from sklearn import svm
import numpy as np
from sklearn.model_selection import KFold
import numpy as np
import numpy as np
from sklearn.metrics import accuracy_score

wine = load_wine()
X = wine.data
y = wine.target
kf = KFold(n_splits=3,shuffle=True)
kf.get_n_splits(X)
kernels=['linear','poly','rbf']
for trainandval_index, test_index in kf.split(X):

    X_trainandval, X_test = X[trainandval_index], X[test_index]
    y_trainandval, y_test = y[trainandval_index], y[test_index]
    #svm
    kf_2 = KFold(n_splits=3,shuffle=True)
    kf_2.get_n_splits(X_trainandval)

    record=np.zeros(3)
    accuracy_score_Records=[]
    for train_index, val_index in kf_2.split(X_trainandval):
        X_train, X_val = X_trainandval[train_index], X_trainandval[val_index]
        y_train, y_val = y_trainandval[train_index], y_trainandval[val_index]
        record_index=0
        for kernel in kernels:
            model = svm.SVC(kernel=kernel, degree=3,C=1)
            clf = model.fit(X_train, y_train)
            y_pred = clf.predict(X_val)
            record[record_index]=accuracy_score(y_val, y_pred)
            record_index=record_index+1
        accuracy_score_Records.append(record)
    accuracy_score_Records=np.array(accuracy_score_Records)
    print(f"'linear','poly','rbf' \n {accuracy_score_Records.mean(axis=0) }")

max_index=np.where(accuracy_score_Records.mean(axis=0)==np.max(accuracy_score_Records.mean(axis=0)))
    print(f'the best model is{kernels[max_index[0].item()]}')
    model_best = svm.SVC(kernel=kernels[max_index[0].item()], degree=3,C=1)
    clf_best = model_best.fit(X_trainandval, y_trainandval)
    print(f'the accuracy_score is {accuracy_score(y_test, clf_best.predict(X_test))}')
```

[COP528]

Task 03. kNN experiment of wine dataset

Use kNN to make the classification on the wine dataset. Compare the results with the SVM methods. Additional: Please compare the classification results using the standardised data with those using the original data to appreciate the usefulness of standardisation (Optional).

(1) Loading the dataset.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm, datasets

wine = load_wine()
X = wine.data
y = wine.target
```

*Solutions Example

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm, datasets

wine = load_wine()
X = wine.data
y = wine.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)

print(X_train.shape, y_train.shape,X_test.shape, y_test.shape)

#knn
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors, datasets
import numpy as np
from sklearn.metrics import accuracy_score

n_neighbors=3
for weights in ["uniform", "distance"]:
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X_train, y_train)
    y_pred=clf.predict(X_test)

    print(f'the accuracy_score before Normalisation:{accuracy_score(y_test, y_pred)}')

# Pre-processing of data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_Norm=scaler.transform(X)
X_Norm_train, X_Norm_test, y_train, y_test = train_test_split(X_Norm, y, test_size=0.3, random_state=0)
for weights in ["uniform", "distance"]:
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X_Norm_train, y_train)
    y_pred=clf.predict(X_Norm_test)
    print(f'after Normalisation:{accuracy_score(y_test, y_pred)}')
```