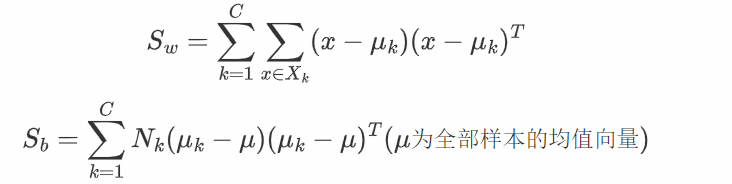
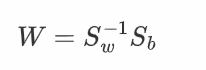
Fisher判别分析

1. 实验思路

定义如下类内散度矩阵和类间散度矩阵，将Fisher判别从二分类推广到C分类：

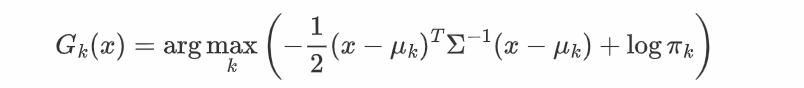


则投影矩阵W为：

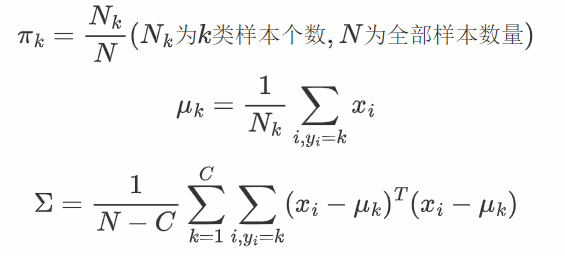


实际计算过程中为了保持数据稳定性，通常对S\_w进行SVD分解，还可以通过选过前k大的奇异向量来对数据进行降维

假设每一类样本服从高斯分布，且协方差矩阵相同，通过MAP估计可以得到决策面方程为：



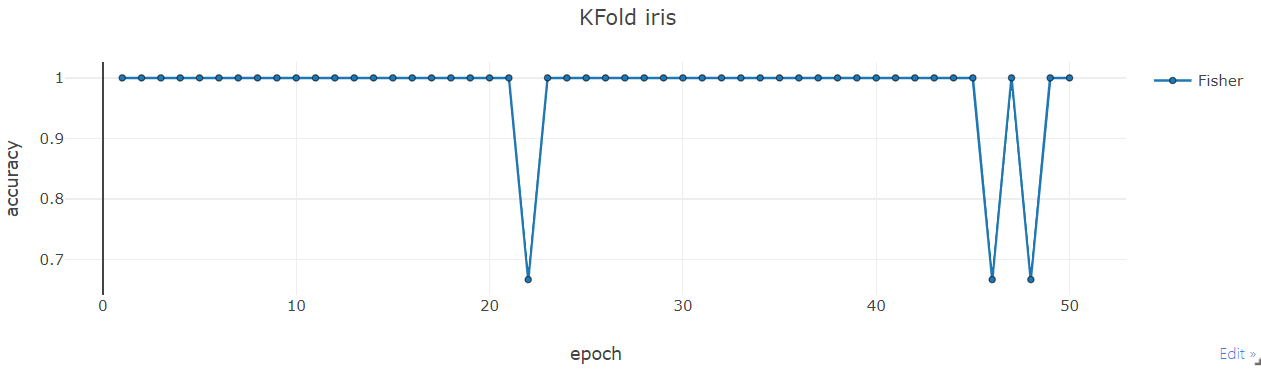
其中参数估计如下：

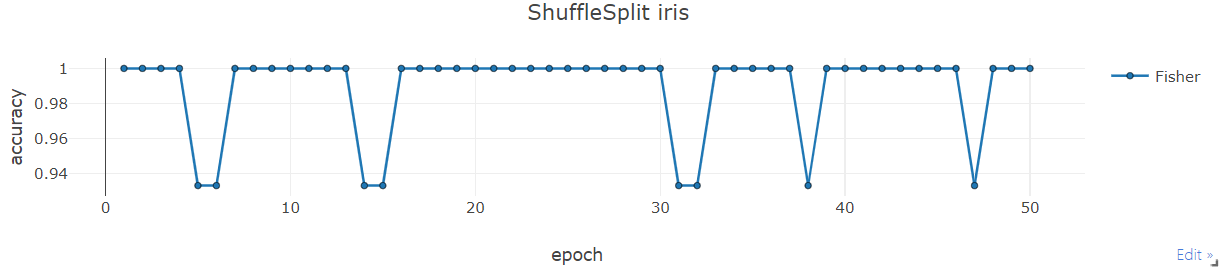


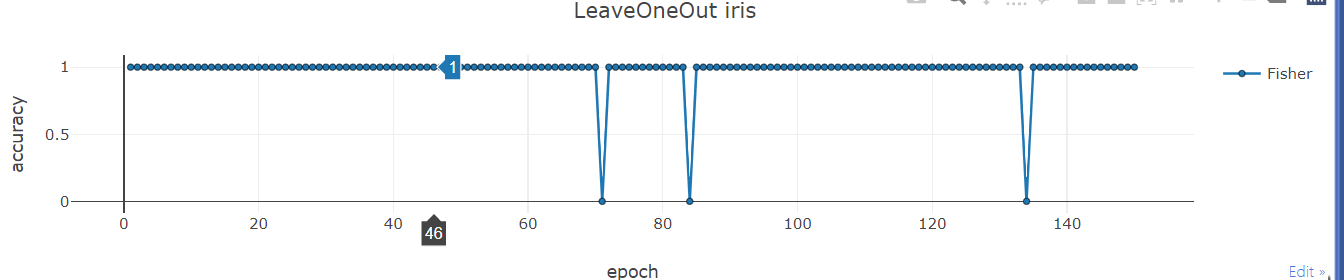
最后再iris, sonar数据集上对其进行K折交叉验证(k=50),和随机分配验证以及留一法验证，然后绘制出降维后的数据分布（二维）

1. 实验结果

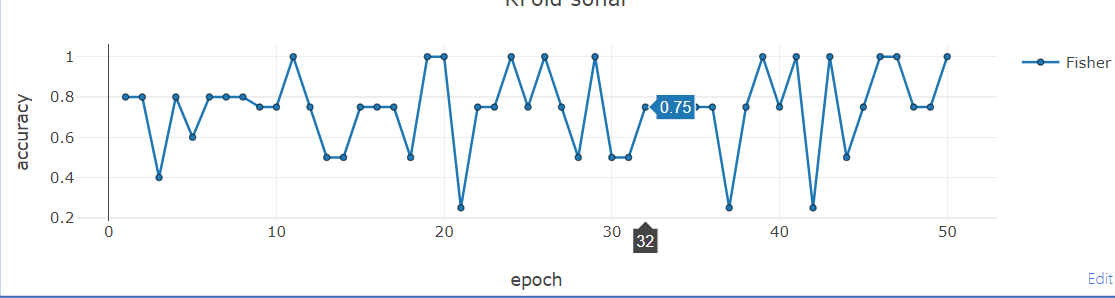
IRIS数据集验证结果：

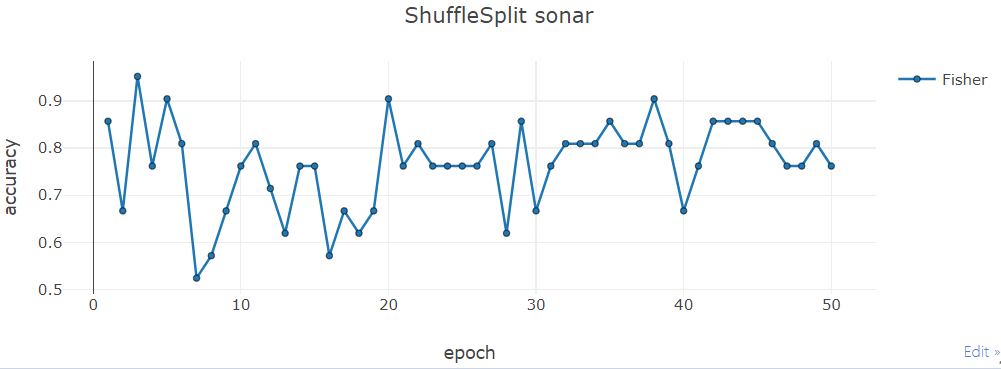


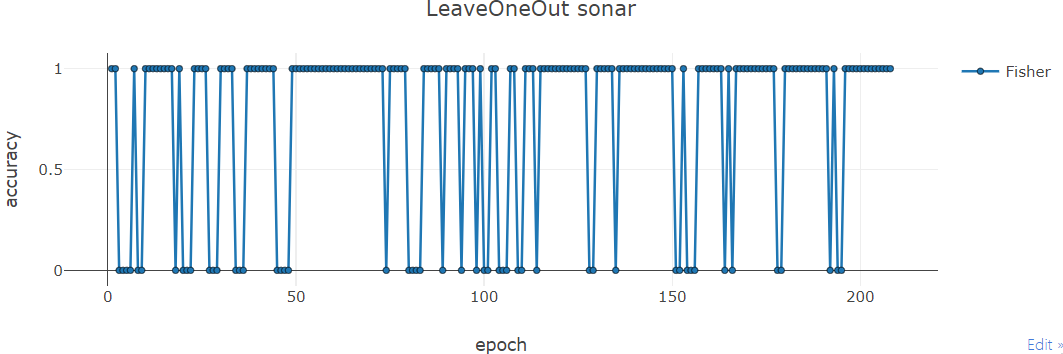




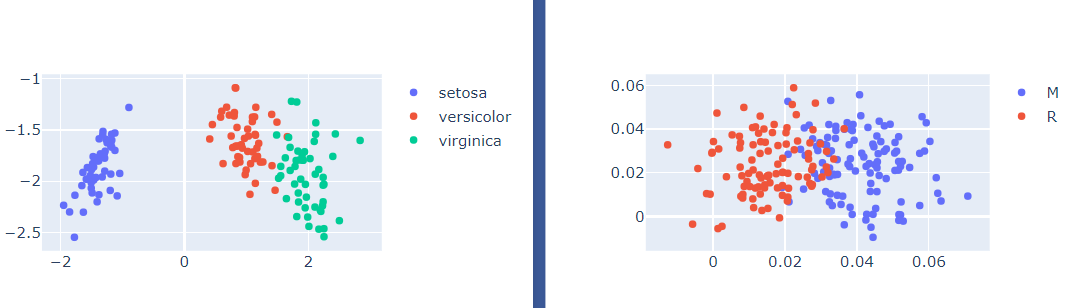
Sonar数据集验证结果：







Iris数据集和sonar数据降维后分布图：



1. 源代码(Python)

from sklearn.model\_selection import KFold, LeaveOneOut, ShuffleSplit

from sklearn.datasets import load\_iris

import pandas as pd

import numpy as np

import plotly.graph\_objects as go

from visdom import Visdom

# 启动Visdom绘图服务

viz = Visdom()

class Fisher:

def \_\_init\_\_(self, priors=None, n\_components=None):

self.S\_w = None

self.S\_b = None

self.label = None

self.priors = priors

self.mu\_i = None # 每一类的均值向量

self.mu = None

self.w = None

self.n\_components = n\_components

def fit(self, X, y):

X, y = np.asarray(X), np.asarray(y)

n\_samples, n\_features = X.shape

assert n\_samples >= 2

# 计算先验概率

if self.priors is None:

self.priors = np.bincount(y) / n\_samples

# 获取类别个数

self.labels, yidx = np.unique(y, return\_inverse=True)

# 计算每个均值向量

means = np.zeros((len(self.labels), n\_features))

np.add.at(means, yidx, X)

self.mu\_i = means / np.expand\_dims(np.bincount(y), 1)

# 计算总体均值向量

self.mu = np.dot(np.expand\_dims(self.priors, axis=0), self.mu\_i)

# 计算类内散度矩阵

covMatrix = [np.cov(X[y == group].T) for group in self.labels]

self.S\_w = sum(covMatrix) / len(covMatrix)

# 计算类间散度矩阵

self.S\_b = sum([sum(y == group)\*np.dot((self.mu\_i[idx, None] - self.mu).T, (self.mu\_i[idx, None] - self.mu))

for idx, group in enumerate(self.labels)]) / (n\_samples - 1)

# 计算投影矩阵

# SVD求Sw的逆矩阵

U, Sigma, V = np.linalg.svd(self.S\_w)

Sigma\_inv = np.linalg.inv(np.diag(Sigma))

Sw\_inv = np.dot(np.dot(V.T, Sigma\_inv), U.T)

Sw\_inv\_Sb = np.dot(Sw\_inv, self.S\_b)

# 求特征值和特征向量，并取实数部分

la, vectors = np.linalg.eig(Sw\_inv\_Sb)

la = np.real(la)

vectors = np.real(vectors)

# 特征值的下标从大到小排列

laIdx = np.argsort(-la)

# 默认选取(N-1)个特征值的下标

if self.n\_components == None:

self.n\_components = len(self.labels)-1

# 选取特征值和向量

lambda\_index = laIdx[:self.n\_components]

w = vectors[:, lambda\_index]

self.w = w

# 求出投影后的矩阵

def transform(self, X):

return np.dot(X, self.w)

# 预测分类情况，出分类概率

def predict\_prob(self, X):

# 求整体协方差的逆

cov = self.S\_w

U, Sigma, V = np.linalg.svd(cov)

Sigma\_inv = np.linalg.inv(np.diag(Sigma))

cov\_inv = np.dot(np.dot(V.T, Sigma\_inv), U.T)

# 线性判别函数值

value = np.log(np.expand\_dims(self.priors, axis=0)) - \

0.5\*np.multiply(np.dot(self.mu\_i, cov\_inv).T, self.mu\_i.T).sum(axis=0).reshape(1, -1) + \

np.dot(np.dot(X, cov\_inv), self.mu\_i.T)

return value/np.expand\_dims(value.sum(axis=1), 1)

# 预测分类情况，出具体分类值

def predict(self, X):

pValue = self.predict\_prob(X)

labelIdx = np.argmax(pValue, axis=1)

return np.array([self.labels[i] for i in labelIdx])

# 准备数据集

iris\_X, iris\_y = load\_iris(return\_X\_y=True)

df = pd.read\_csv("sonar.csv", header=None)

df[60] = df[60].replace({"M":0, "R":1})

features = [i for i in range(60)]

sonar\_X, sonar\_y = df[features].values, df[60].values

# 计算准确率

def predict\_score(X, y, model):

res = model.predict(X)

return (res == y).sum() \* 1.0 / y.shape[0]

# 交叉验证

def cross\_val\_score(model, X, y, splitFun):

score\_list = list()

for train\_idx, test\_idx in splitFun.split(X):

train\_X, train\_y = X[train\_idx], y[train\_idx]

test\_X, test\_y = X[test\_idx], y[test\_idx]

model.fit(train\_X, train\_y)

score\_list.append(predict\_score(test\_X, test\_y, model=model))

return score\_list

# 对模型进行交叉验证并可视化结果

def val\_plot\_Modules(modules: dict, X, y, n\_splits=50, name=""):

assert len(modules) > 0, "Least have one model in modules!"

legend\_name, modules = list(modules.keys()), list(modules.values())

model\_cnt = len(modules)

splitFn\_list = [KFold(n\_splits=n\_splits, shuffle=True),

LeaveOneOut(), ShuffleSplit(n\_splits)]

result = [list() for \_ in range(model\_cnt)]

for i in range(model\_cnt):

for fn in splitFn\_list:

result[i].append(cross\_val\_score(modules[i], X, y, fn))

title\_name = ["KFold", "LeaveOneOut", "ShuffleSplit"]

for i in range(3):

data\_len = len(result[0][i])

if model\_cnt > 1:

plot\_y = [[result[idx][i][k] for idx in range(model\_cnt)]

for k in range(data\_len)]

plot\_x = [[k for \_ in range(model\_cnt)] for k in range(data\_len)]

else:

plot\_y = result[0][i]

plot\_x = np.arange(1, data\_len + 1, 1)

viz.line(Y=plot\_y, X=plot\_x, win=title\_name[i] + name,

opts=dict(title=title\_name[i] + name, markers=True, markersize=5,

legend=legend\_name,

xlabel="epoch", ylabel="accuracy"))

# 绘制降维后的数据分布图

def plot\_Fisher\_Projection(X, y, labels:list, name="projection"):

model = Fisher(n\_components=2)

model.fit(X, y)

X\_tran = model.transform(X)

classes = len(labels)

x1 = [X\_tran[y==i, 1] for i in range(classes)]

x0 = [X\_tran[y==i, 0] for i in range(classes)]

sc = list()

for i in range(classes):

sc.append(go.Scatter(x=x0[i], y=x1[i], mode="markers", name=labels[i]))

fig = go.Figure(sc)

viz.plotlyplot(fig, win=name)

val\_plot\_Modules(dict(Fisher=Fisher()), iris\_X, iris\_y, name=" iris")

val\_plot\_Modules(dict(Fisher=Fisher()), sonar\_X, sonar\_y, name=" sonar")

iris\_labels = ['setosa', 'versicolor', 'virginica']

sonar\_labels = ['M', 'R']

plot\_Fisher\_Projection(iris\_X, iris\_y, iris\_labels, "iris")

plot\_Fisher\_Projection(sonar\_X, sonar\_y, sonar\_labels, "sonar")