103590450 四資四 馬茂源

[1]
```
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
import operator
import pandas as pd
import itertools
```

[2]
```
class MyKNeighborsClassifier:

    def __init__(self, n_neighbors=3, **kwargs):
        self._k = n_neighbors
        self._X = self._y = None
        self.set_params(**kwargs)

    def get_params(self, deep=True):
        # suppose this estimator has parameters "alpha" and "recursive"
        return self.__dict__

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def fit(self, X, y):
        self._X = X
        self._y = y

    def _predict(self, x):
        distances = np.apply_along_axis(lambda x1: np.linalg.norm(x-x1),
                                        1, self._X)
        X_candidates = np.argsort(distances)[:self._k]
        y_candidates = self._y[X_candidates]
        return np.argmax(np.bincount(y_candidates.astype('int64')))

    def score(self, X, y_true):
        return accuracy_score(y_true, self.predict(X))

    def predict(self, X):
        return np.apply_along_axis(lambda x: self._predict(x), 1, X)
```

[3]
```
iris = load_iris()
feature_names = iris.feature_names.copy()
iris = pd.DataFrame(data=np.column_stack([iris.data, iris.target]),
```

```
                      columns=iris.feature_names+['target'])
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

1

In the lecture, we mentioned that the term λ ∑ $wi2i$ is a regularization term.

i. Use your own words to explain why it is so.

ii. If we implement two regression programs, one using λ ∑ $wi2i$ and theother one using λ ∑ $wi10i$, which one do you expect has lower bias? How about variance? Why? Assuming that we use the same λ for both programs.

針對第一小題，L2正規化的作用在於，在一個具有overfitting能力的model上我們在loss func後方追加L2 term，因此這個loss func在計算loss時不僅取決於每個預測值的誤差的總和(原項)，同時也取決於在λ的比例下w(係數們)的平方和，會這樣做的原因在於一個具有高次項(high polynomial degree)的model，在發生overfitting時高次項係數會開始變大(或者說極端化)，這是Model為了fit所有數值所造成的現象(為了所有point曲線開始扭曲)，扭曲意味著需要高次項的幫助，也就是說overfitting是因為某些係數(通常為高次項)極端化造成的，因此會造成w的平方和變大，變大不是我們想要的現象，就跟誤差一樣，因此我們可以將L2追加在loss func後面一起評估，這就是L2功能。

針對第二小題，假設相同λ，10次方和正規化這個應該會比平方和正規化產生更多限制的影響，因此正規化程度嚴重，也就是說model比起L2更不會有overfitting現象，換句話說L2跟L10相比L2的model會比較fit data，因此L10正規化的bias相較L2比較高但var比L2低，最後針對題目回答，L2應該會有比較低的bias。

2

We mentioned that the covariance matrix may be ill-conditioned. Find the(sample) covariance matrices for the three classes of the Iris dataset and compute the condition numbers for the covariance matrices. For simplicity, use the following as the condition number: κ(A) = |λmax/ λmin|, where λmax and λmin arethe largest and smallest eigenvalues of matrix A.

[4]
```
for tx in range(3):
    targetx = iris.loc[iris.target == tx, feature_names]
    targetx = targetx.apply(lambda x: x-x.mean())
    cov = [(targetx[f1]*targetx[f2]).mean()
           for f1, f2 in itertools.product(feature_names,
                                           feature_names)]
    cov = np.array(cov).reshape((len(feature_names),
                                 len(feature_names)))
    eigvals, _ = np.linalg.eig(cov)
    display('class %d'%(tx),
            'condition number:{}'.format(abs(eigvals.max() /
                                             eigvals.min())))
    display(pd.DataFrame(data=cov,
                         columns=feature_names,
                         index=feature_names))
```

'class 0''condition number:25.373917566328807'

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	0.121764	0.098292	0.015816	0.010336
sepal width (cm)	0.098292	0.142276	0.011448	0.011208
petal length (cm)	0.015816	0.011448	0.029504	0.005584
petal width (cm)	0.010336	0.011208	0.005584	0.011264

'class 1''condition number:49.83204972740381'

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	0.261104	0.08348	0.17924	0.054664
sepal width (cm)	0.083480	0.09650	0.08100	0.040380
petal length (cm)	0.179240	0.08100	0.21640	0.071640
petal width (cm)	0.054664	0.04038	0.07164	0.038324

In this problem, you are asked to use the Iris dataset to perform PCA dimensionality reduction before classification. Randomly draw 35 samples in each class to find the vectors $\boldsymbol{w}(j)$ for the largest two principal components. Recall that PCA is unsupervised; therefore, you need to use 35×3 = 105 data points to find the parameters of the PCA. Implement the 3-NN classifier to test the rest 15 samples in each class and record the accuracy. Repeat the drawing and the k-NN classification 10 times and compute the average accuracy and variance. For simplicity, use the Euclidean distance in the k-NN computation.

[5]
```
def randomly_draw(data):
    train = pd.DataFrame()
    for i in range(3):
        train = train.append(data[data.target == i].sample(35))

    test = data[~(iris.isin(train).all(axis=1))].copy()
    return train.as_matrix(), test.as_matrix()
```

[6]
```
def pca(train_X, test_X):
    cov = train_X.T.dot(train_X)
    eigvalue, loading = np.linalg.eig(cov)
    # display(eigvalue, loading)
    new_train_X, new_test_X = (train_X.dot(loading[:, :2]),
                               test_X.dot(loading[:, :2]))
    return new_train_X, new_test_X
```

[7]
```
# train, test = randomly_draw(iris)
```

[8]
```
# train_X, train_y = train[:, :-1], train[:, -1]
# test_X, test_y = test[:, :-1], test[:, -1]
```

[9]
```
# new_train_X, new_test_X = pca(train_X, test_X)
```

[10]
```
acc = []
for i in range(10):
    train, test = randomly_draw(iris)
    train_X, train_y = train[:, :-1], train[:, -1]
    test_X, test_y = test[:, :-1], test[:, -1]
    train_X, test_X = pca(train_X, test_X)
    model = MyKNeighborsClassifier()
    model.fit(train_X, train_y)
    acc.append(model.score(test_X, test_y))
print('average acc:{}'.format(np.mean(acc)))
print('variance of acc:{}'.format(np.var(acc)))
```

```
average acc:0.9666666666666666
variance of acc:0.0002222222222222214
```

Following the general steps of problem 3, but use the FA approach for dimensionality reduction. For simplicity, you may assume □ = 0 and use the LS solutions.

[16]
```
acc = []
for i in range(10):
    train, test = randomly_draw(iris)
    train_X,  train_y = train[:, :-1], train[:, -1]
    test_X, test_y = test[:, :-1], test[:, -1]

    cov = np.cov(train_X, rowvar=False)
    eigvalue, eigvertor= np.linalg.eig(cov)

    V = (eigvertor[:, :2]).dot(np.diag(eigvalue[:2])**0.5)
    Vbar = np.linalg.inv(V.T.dot(V)).dot(V.T)
    #display(V, Vbar)
    #break
    train_X = train_X.dot(Vbar.T)
    test_X = test_X.dot(Vbar.T)
    model = MyKNeighborsClassifier()
    model.fit(train_X, train_y)
    acc.append(model.score(test_X, test_y))
print('average acc:{}'.format(np.mean(acc)))
print('variance of acc:{}'.format(np.var(acc)))
```

```
average acc:0.9622222222222222
variance of acc:0.0006962962962962956
```

5

Repeat problem 3 by using LDA as the reduction method. Remember to compute the parameters for each class in order to use LDA.

[12]
```
def lda(data):
    Sw = np.zeros((4, 4))
    for i in range(3):
        xinDi = data[data[:,-1] == i][:, :-1]
        mi = xinDi.mean(axis=0)
        Si = np.zeros((4, 4))
        for row in xinDi:
            row, mv = row.reshape(4,1), mi.reshape(4,1)
            Si += (row-mv).dot((row-mv).T)
        Sw += Si

    Sb = np.zeros((4, 4))
    m = data[:, :-1].mean(axis=0).reshape(4, 1)
    for i in range(3):
        xinDi = data[data[:,-1] == i][:, :-1]
        mi = xinDi.mean(axis=0).reshape(4, 1)
```

```
            n = xinDi.shape[0]
            Sb += n*(mi-m).dot((mi-m).T)

        #display(Sw, Sb)
        lda_eigvalue, lda_eignvector = np.linalg.eig(
                                        np.linalg.inv(Sw).dot(Sb))
        #display(lda_eigvalue.real, lda_eignvector.real)
        return lda_eigvalue, lda_eignvector
```

[13]
```
data = iris.as_matrix()
lda_eigvalue, lda_eignvector = lda(data)
X, y = data[:, :-1].dot(lda_eignvector[:, :2]), data[:, -1]
display(lda_eigvalue.real, lda_eignvector.real)
```

```
array([ 3.22719578e+01,  2.77566864e-01, -6.73276390e-16, -6.73276390e-
16])array([[-0.20490976,  0.00898234,  0.27709585,  0.27709585],
       [-0.38714331,  0.58899857, -0.386313  , -0.386313  ],
       [ 0.54648218, -0.25428655, -0.438815  , -0.438815  ],
       [ 0.71378517,  0.76703217,  0.66441477,  0.66441477]])
```

[14]
```
#check Av = λv
# for i in range(len(lda_eigvalue)):
#     eigv = lda_eignvector[:, i].reshape(4,1)
#     np.testing.assert_array_almost_equal(np.linalg.inv(Sw).dot(Sb).dot(
#                                          lda_eigvalue[i] * eigv,
#                                          decimal=6, err_msg='', verbose
```

[15]
```
acc = []
for i in range(10):
    train, test = randomly_draw(iris)

    lda_eigvalue, lda_eignvector = lda(train)
    train_X, train_y = (train[:, :-1].dot(lda_eignvector[:, :2]),
                        train[:, -1])
    test_X, test_y = (test[:, :-1].dot(lda_eignvector[:, :2]),
                      test[:, -1])

    model = MyKNeighborsClassifier()
    model.fit(train_X, train_y)
    acc.append(model.score(test_X, test_y))

print('average acc:{}'.format(np.mean(acc)))
print('variance of acc:{}'.format(np.var(acc)))
```

```
average acc:0.9733333333333334
variance of acc:0.00017777777777777708
```