q1

April 27, 2021

[1]:
```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from numpy import array, mean, append, std
from numpy.random import shuffle
from random import choice, randint
import matplotlib.pyplot as plt
```

[2]:
```
digits = load_digits()

X = digits['data']
target = digits['target']
```

Part A: Train a Linear Classifier on Digits Dataset

[3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size = .5)
```

[4]:
```
clf = SGDClassifier()
clf.fit(X_train, y_train)
```

[4]:
```
SGDClassifier()
```

Part B: Implement the Given Function

[5]:
```
# Tests classifier model M on 10**4 random samples of labeled data set (X,Y)
# (predictive attributes in X; labels in Y), each of size testSize, and
# returns an array of the mean classification accuracies for these samples

def testModel(M, X, y, testSize):

    accuracies = []
    indexes = array(list(range(0, testSize)))

    for _ in range(10**4):

        sample_X = []
        sample_y = []
```

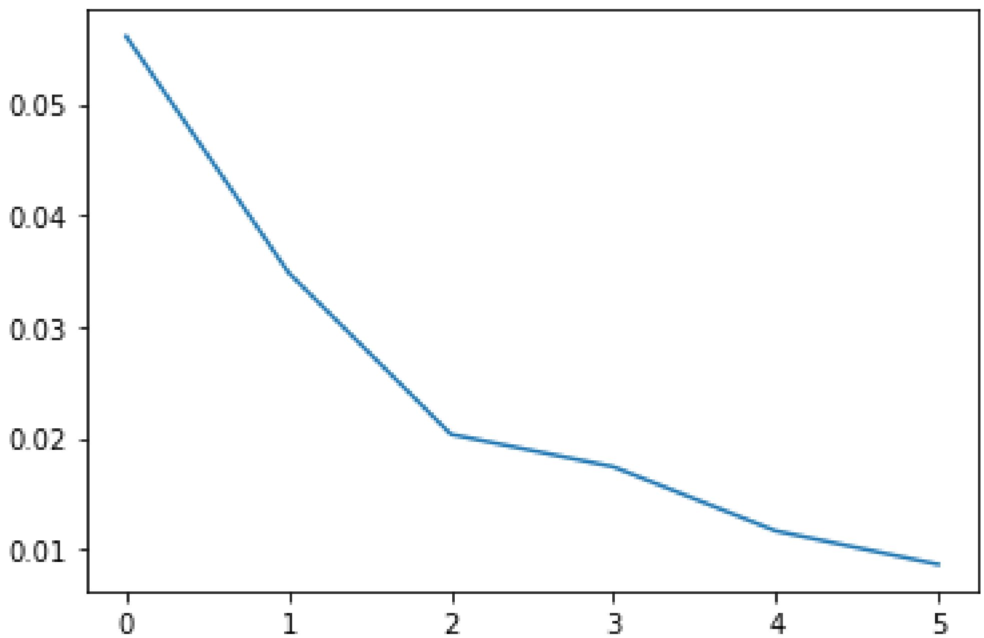
```python
        for _ in range(testSize):

            # shuffle the indices
            shuffle(indexes)

            sample_X.append(X[indexes[0]])
            sample_y.append(y[indexes[0]])

        sample_X = array(sample_X)
        sample_y = array(sample_y)

        # perform predictions on the test splits and get the accuracy
        accuracy = M.score(sample_X, sample_y)

        # append accuracy to array of accuracies
        accuracies.append(accuracy)

    return accuracies
```

[6]:
```python
deviations = []

for k in range(0, 5+1):

    num_rows = 24 * (2**k)

    test_X = X_test[0:num_rows]
    test_Y = y_test[0:num_rows]

    accuracies = testModel(clf, test_X, test_Y, len(test_X))

    print("\nMean (k={}): {:f}".format(k, round(mean(accuracies), 3)))
    print("Stdev (k={}): {:f}".format(k, round(std(accuracies), 3)))

    deviations.append(std(accuracies))

plt.plot(list(range(6)), deviations)
```

```
Mean (k=0): 0.917000
Stdev (k=0): 0.056000

Mean (k=1): 0.938000
Stdev (k=1): 0.035000

Mean (k=2): 0.959000
Stdev (k=2): 0.020000
```

```
Mean (k=3): 0.937000
Stdev (k=3): 0.017000

Mean (k=4): 0.945000
Stdev (k=4): 0.012000

Mean (k=5): 0.939000
Stdev (k=5): 0.009000
```

[6]: [<matplotlib.lines.Line2D at 0x22d0c8b3e80>]



q2

April 27, 2021

[1]:
```
from numpy import mean
from math import sqrt
import matplotlib.pyplot as plt
```

Part A: How Do We Expect the Expected Value and Stdev to Vary as a Function of Test Size?

To begin, consider that each individual trial (X_i) is a Bernoulli random variable with parameter c (where c is considered the probability of a correct classification for a single sample). Then we have

$$E[X_i] = cVar(X_i) = c(1-c)$$

$$\implies Var(NX) = N^2 \cdot Var(X)$$

$$\sigma = \sqrt{var(X)}$$

Taking the average of N Bernoulli RV's each w/ classification accuracy c (probability correct) we have that the expected value among the N bernoulli trials is

$$E[\frac{1}{N}\sum_{i=1}^{N} X_i] = \frac{1}{N}\sum_{i=1}^{N} E[X_i] = \frac{Nc}{N} = c$$

For the standard deviation:

$$\frac{1}{N^2}\sum_{i=1}^{N} var(X) = \frac{1}{N^2}Nc(1-c) = \frac{c(1-c)}{N}$$

and standard deviation (square root of variance)

$$\sigma = \sqrt{\frac{c(1-c)}{N}}$$

So the standard deviation increases by a factor of $\frac{1}{\sqrt{N}}$

Part B: Apply Theoretical Results to Experimental (Above)

[2]:

```
# arrays from previous problem
means = [.958917, .937329, .916595, .900986, .895698, .907546]
stdevs = [.040264, .035297, .028471, .021488, .015385, .010430]

# average accuracy (experimental)
c = mean(means)

stdev_theoretical_predictions = []
for k in range(0, 5 + 1):

    test_size = 24 * (2**k)
    stdev_theoretical_predictions.append(sqrt((1/test_size) * c * (1-c)))
```

[3]:

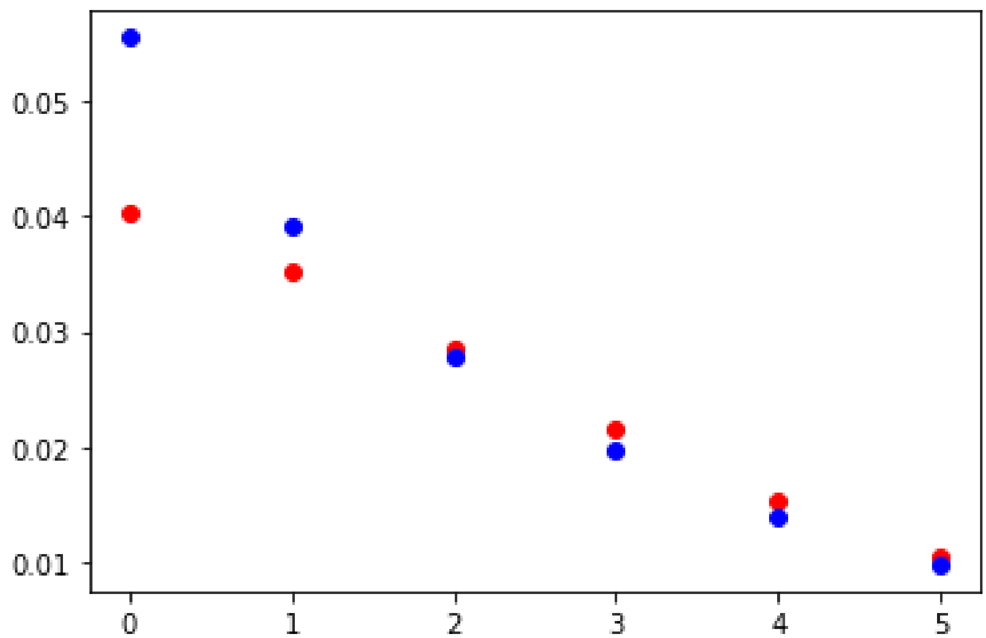
```
# do these conclusions agree?
print(stdev_theoretical_predictions)
print(stdevs)

# looks like they agree for the most part but we can plot against one another
```

```
[0.05553145568553371 4, 0.03926666888440115, 0.027765727842766857,
0.019633334442200574, 0.013882863921383428, 0.009816667221100287]
[0.040264, 0.035297, 0.028471, 0.021488, 0.015385, 0.01043]
```

[4]:

```
plt.scatter(list(range(0, 5+1)), stdevs, c="red")
plt.scatter(list(range(0, 5+1)), stdev_theoretical_predictions, c="blue")
```

[4]: <matplotlib.collections.PathCollection at 0x26da9b08a00>



```
[ ]:
```

q3

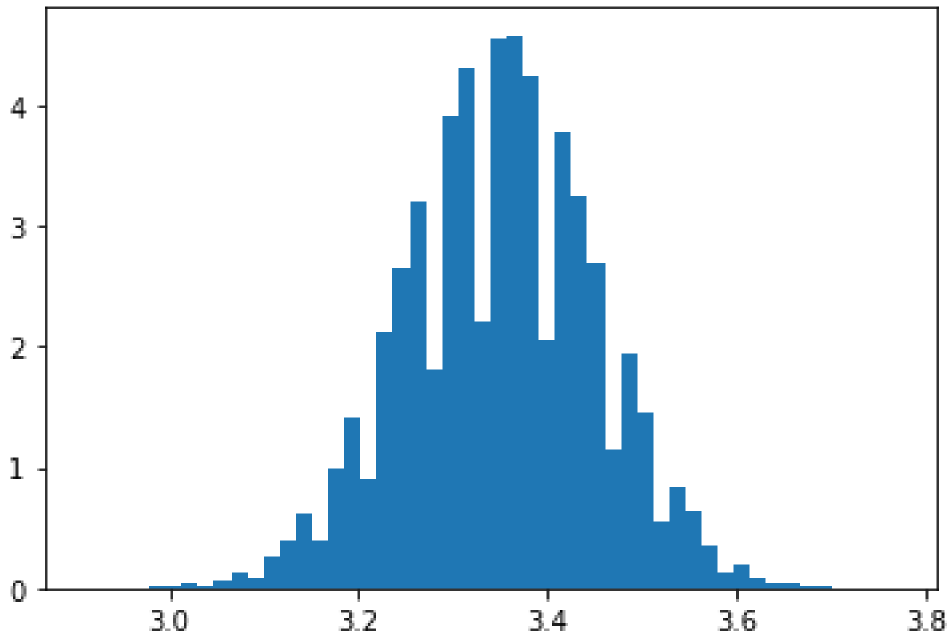
April 27, 2021

[1]:
```
from random import random
from numpy import array, mean, linspace
from scipy.stats import norm
from seaborn import distplot
import matplotlib.pyplot as plt
from math import sqrt
```

Part A: Implement the PDF Given and Plot a Histogram of the Mean Values

[2]:
```
def get_from_pmf():

    random_num = random()

    if random_num >= 0 and random_num <= .05:
        return 1

    elif random_num > .05 and random_num <= .2:
        return 2

    elif random_num > .2 and random_num <= .5:
        return 3

    elif random_num > .5 and random_num <= .9:
        return 4

    else:
        return 5
```

[3]:
```
mean_values = []

for _ in range(10**5):

    # generate a list of 100 values from the given PMF
    random_sample = [get_from_pmf() for _ in range(100)]

    mean_values.append(mean(array(random_sample)))

hist = plt.hist(mean_values, density=True, bins=50)
```



Part B: Determine Expected Value and Stdev of The Given PDF

Analytically, we get that

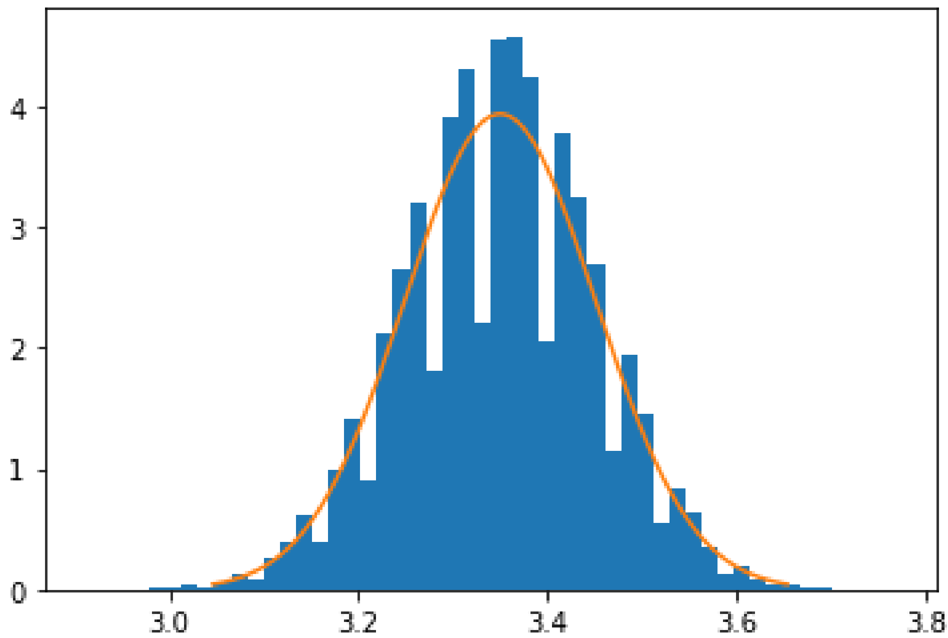
$$E[X] = 1(.05) + 2(.15) + 3(.3) + 4(.4) + 5(.1) = 3.35$$

and

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - E[X])^2} = 1.0137$$

[4]:
```
# expected value of 100 IID
# expected value does not change
exp_100 = 3.35
std_100 = 1.0137 / sqrt(100)

print(std_100)

plt.hist(mean_values, density=True, bins=50)
x_range = linspace(3.35 - std_100*3, 3.35 + std_100*3, 1000)
plt.plot(x_range, norm.pdf(x_range, loc=exp_100, scale=std_100))

print(std_100)
```

```
0.10137
0.10137
```



[ ]:

q4

April 27, 2021

[1]:
```python
from sklearn.datasets import load_iris
from sklearn import naive_bayes
from numpy import array
```

Part A: Fit a Gaussian Naive Bayes Model to the Full Iris Dataset

[2]:
```python
iris = load_iris()

X = iris['data']
target = iris['target']

clf_NB = naive_bayes.GaussianNB()

# fit the model to the full Iris dataset
clf_NB.fit(X, target)

# print the values of theta_c and sigma^2_c
theta_c = clf_NB.theta_
variance_c = clf_NB.sigma_
print("Theta_c: \n\n{}\n\nSigma^2_c: \n\n{}".format(theta_c, variance_c))
```

```
Theta_c:

[[5.006 3.428 1.462 0.246]
 [5.936 2.77  4.26  1.326]
 [6.588 2.974 5.552 2.026]]

Sigma^2_c:

[[0.121764 0.140816 0.029556 0.010884]
 [0.261104 0.0965   0.2164   0.038324]
 [0.396256 0.101924 0.298496 0.073924]]
```

The matrices above show the values of the mean of each feature per class and variance of each each feature per class respectively.

As we see from the matrix of per-class variances, attribute 3 has the smallest variance in all classes. The individual Gaussian distributions for these attributes will be "thinner" than those of the other attributes among classes.

Attribute 2 has the smallest expected value in class 1. For the distributions of attribute 2 among all 3 classes, we can expect the plot of the distribution for this attribute to be centered farthest to the left on the x-axis for class 1.

Part B: Determine the Probability the Given Vector Belongs to Each Class

[3]:
```
x_0 = array([5, 3, 2, .8])

# use predict_proba method to determine for each class the probability that x_0␣
↪belongs to each class

predictions = clf_NB.predict_proba(x_0.reshape(1, -1))[0]

for class_num, prob in enumerate(predictions):

    print("Probability Class {}: {:f}".format(class_num + 1, prob))
```

```
Probability Class 1: 0.385446
Probability Class 2: 0.614554
Probability Class 3: 0.000000
```

The probability that the attribute vector belongs to class 2 is approximately .6146, which is higher than the rest of the probabilities, meaning the attribute vector most likely belongs to this class.

q5

April 27, 2021

[1]:
```
from scipy.stats import norm
from sklearn import naive_bayes
from numpy import array, prod, append
from sklearn.datasets import load_iris
from math import sqrt
```

[2]:
```
def predict_proba_custom(M, x):

    # get the values of theta_c and sigma^2_c
    theta_c = clf_NB.theta_
    variance_c = clf_NB.sigma_

    # get class probabilities from model
    class_probabilities = M.class_prior_
    # calculate normalizing term K before computing each class probability

    numerators = []

    # iterate through each class
    for class_num in range(len(M.class_count_)):

        # get the per-class expectations and variances
        theta_c_i = theta_c[class_num]
        variance_c_i = variance_c[class_num]

        # prior probability of class c
        prior_c = class_probabilities[class_num]

        # calculate numerator of P(c | x) for the current c

        # compute the probability of getting the attribute vector given class c␣
↪(P(x | c))
        # we need to look at theta_c, sigma^2_c per class attribute
        # the probability of x given c is the product of P(X_i | c) by␣
↪independence assumption
        prob_x_given_c = prod([norm.pdf(x_i, loc=theta_c_i[i],␣
↪scale=sqrt(variance_c_i[i])) for i, x_i in enumerate(x)])
```

```python
        numerators.append(prior_c * prob_x_given_c)

    K = sum(numerators)

    probabilities = []
    for numerator in numerators:
        probabilities.append(numerator/K)

    return array(probabilities)
```

[3]:
```python
iris = load_iris()

X = iris['data']
target = iris['target']

clf_NB = naive_bayes.GaussianNB()

# fit the model to the full Iris dataset
clf_NB.fit(X, target)

x_0 = [5, 3, 2, .8]
proba = predict_proba_custom(clf_NB, x_0)

for class_num, prob in enumerate(proba):

    print("Probability Class {}: {:f}".format(class_num + 1, prob))
```

```
Probability Class 1: 0.385446
Probability Class 2: 0.614554
Probability Class 3: 0.000000
```