Dressing Recommendation by different Bayes Classifier
--Pingye Sun

1. Problem Description

- We learned Bayes classifier in our class, and we have implemented the full Bayes and Naïve Bayes in our code by Python. Meanwhile, there are different Bayes implements in Python unit and different Bayes models in Naïve Bayes, such as 'Multinomial Naive Bayes', 'Bernoulli Naive Bayes' etc. My project is to better understand different implementations of Bayes, and give the comparison of them.
- To do the comparison, I use a dressing dataset to do training and testing, the dataset is downloaded from UCI website

2. Learn more about Bayes classifier

2.1 Models of Bayes

- Full Bayes
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bernoulli Naive Bayes

In our class, we studied the Bayes classifier and implemented the Full and Gaussian Naive Bayes by our code.

The main difference between these models is the definition for the 'likelihood of the features'. Different Naïve Bayes models has different likelihood function definition.

2.2 Gaussian Naive Bayes

`GaussianNB` implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Source: http://scikit-learn.org/stable/modules/naive_bayes.html

2.3 Multinomial Naive Bayes

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Source: http://scikit-learn.org/stable/modules/naive_bayes.html

2.4 Bernoulli Naive Bayes

`BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a `BernoulliNB` instance may binarize its input (depending on the `binarize` parameter).

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature.

Source: http://scikit-learn.org/stable/modules/naive_bayes.html

3. Dressing Recommendation

We need to apply the algorithms to real data and questions of interest, so I choose the 'Dressing Recommendation' problem.

3.1 Data

There is a database of dressing information, we use different classifiers to do the recommendation, and test the accurate rate, running time and performance of different classifiers.

- Data source: https://archive.ics.uci.edu/ml/datasets/Dresses_Attribute_Sales
- Data set including 500 dressing information
- Each piece of information includes the value of attributes for a dress,such as
  Style,Price,Rating,Size,Season,NeckLine,SleeveLength……
- Two types of classes:
  1: recommended

0: not recommended

3.1.1 Raw data format

The raw data form(Attribute DataSet.xlsx) contains the information of different features for the dresses

Dress_ID	Style	Price	Rating	Size	Season	NeckLine	SleeveLength	waiseline	Material	FabricType	Decoration	Pattern Type	Recommendation
1006032852	Sexy	Low	4.6	M	Summer	o-neck	sleeveless	empire	null	chiffon	ruffles	animal	1
1212192089	Casual	Low	0	L	Summer	o-neck	Petal	natural	microfiber	null	ruffles	animal	0
1190380701	vintage	High	0	L	Autumn	o-neck	full	natural	polyster	null	null	print	0
966005983	Brief	Average	4.6	L	Spring	o-neck	full	natural	silk	chiffon	embroidary	print	1
876339541	cute	Low	4.5	M	Summer	o-neck	butterfly	natural	chiffonfabric	chiffon	bow	dot	0
1068332458	bohemian	Low	0	M	Summer	v-neck	sleeveless	empire	null	null	null	print	0
1220707172	Casual	Average	0	XL	Summer	o-neck	full	null	cotton	null	null	solid	0
1219677488	Novelty	Average	0	free	Autumn	o-neck	short	natural	polyster	broadcloth	lace	null	0
1113094204	Flare	Average	0	free	Spring	v-neck	short	empire	cotton	broadcloth	beading	solid	1
985292672	bohemian	Low	0	free	Summer	v-neck	sleeveless	natural	nylon	chiffon	null	null	1
1117293701	party	Average	5	free	Summer	o-neck	full	natural	polyster	broadcloth	lace	solid	0
898481530	Flare	Average	0	free	Spring	v-neck	short	null	nylon	null	null	animal	0
957723897	sexy	Low	4.7	M	Winter	o-neck	threequarter	null	null	chiffon	lace	print	1
749031896	vintage	Average	4.8	M	Summer	o-neck	short	empire	cotton	jersey	null	animal	1
1055411544	Casual	Low	5	M	Summer	boat-neck	short	null	cotton	null	sashes	solid	0
1162628131	Casual	Low	0	free	Winter	boat-neck	full	null	other	other	lace	null	0
624314841	cute	Average	4.7	L	spring	o-neck	short	null	cotton	null	sashes	solid	1

3.1.2 Pretreated Data

We can see that the raw data is consisted with string words, which is difficult to calculate and serialize by Bayes classifier. So we need to digitize these information: for each feature, define different number of different value and save it as a csv file. So we get the pretreated data below(form: dressing.csv):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	1	4.6	1	1	1	1	1	0	1	1	1	1
2	2	1	0	2	1	1	2	2	1	0	1	1	0
3	3	2	0	2	2	1	3	2	2	0	0	2	0
4	4	3	4.6	2	3	1	3	2	3	1	2	2	1
5	5	1	4.5	1	1	1	4	2	4	1	3	3	0
6	6	1	0	1	1	2	1	1	0	0	0	2	0
7	2	3	0	3	1	1	3	0	5	0	0	4	0
8	7	3	0	4	2	1	5	2	2	2	4	0	0
9	8	3	0	4	3	2	5	1	5	2	5	4	1
10	6	1	0	4	1	2	1	2	6	1	0	0	1
11	9	3	5	4	1	1	3	2	2	2	4	4	0
12	8	3	0	4	3	2	5	0	6	0	0	1	0
13	1	1	4.7	1	4	1	6	0	0	1	4	2	1
14	10	3	4.8	1	1	1	5	1	5	3	0	1	1
15	2	1	5	1	1	3	5	0	5	0	6	4	0

3.2 My implementation

3.2.1 Gaussian Naïve Bayes

This part we have done in the homework2 in our class. The probability density function is like below:

```
def calculateProbability(x, mean, cov, isNaive = False):
    """calculate the probability density function of Fx.
    input:
    x: test case
    mean:array of means for the x's type
    cov: array of covariance matrix of x's type
    isNaive: if true, is naive bayes's operation, otherwise, full bayes

    Output: the pdf
    """
    d = len(x)
    stdev = math.fabs(np.linalg.det(cov))
    n = 1 / ((math.sqrt(2*math.pi)**d * stdev))
    temp = np.dot(x - mean[0], np.linalg.inv(cov))
    exponentnum = math.fabs(np.dot(temp, np.transpose(x - mean[0])))
    exponent = float(math.exp(1)**(-exponentnum / 2))

    return n * exponent
```

The whole source code : Gaussian_bayes.py

3.2.2 Multinomial Naïve Bayes:

The main different part is the pdf function.

```
def multi_var_normal_pdf(x, classes, resultsets, Pc_count, wordtype_num):
    """
    calculate the pdf valuea for the input case x
    """
    pbs = []
    # set alpha = 1, because there may be some attribute value which is not included in the resultsets
    alpha = 1
    for cla in classes:
        count = Pc_count[cla]
        rset = resultsets[cla]
        pb = 1
        for attr in x:
            if attr not in rset:
                pb *= alpha/(count + wordtype_num)
            else:
                pb *= rset[attr]
        pbs.append(pb)
    return pbs
```

The whole source code: Multinomial_bayes.py

3.2.3 Bernoulli Naïve Bayes

The main different part is the pdf function.

```
def multi_var_normal_pdf(x, classes, resultsets):
    """
    calculate the pdf valuea for the input case x
    """
    pbs = []
    alpha = 0.1
    for cla in classes:
        rset = resultsets[cla]
        pb = 1
        for attr in rset:
            if attr in x:
                pb *= rset[attr]
            else:
                pb *= 1 - rset[attr]
        pbs.append(pb)
    return pbs
```

Whole source code: Bernoulli_bayes.py

3.3 30-Folder Test for my implementation

In this part, I will test my implementation for different Bayes classifier by 30-Foder test. I will run the test repeatedly and count the statistic information of accept and reject times. By analyzing the different result, I will give a conclusion and the possible reasons.

3.3.1 Gaussian Naïve Bayes

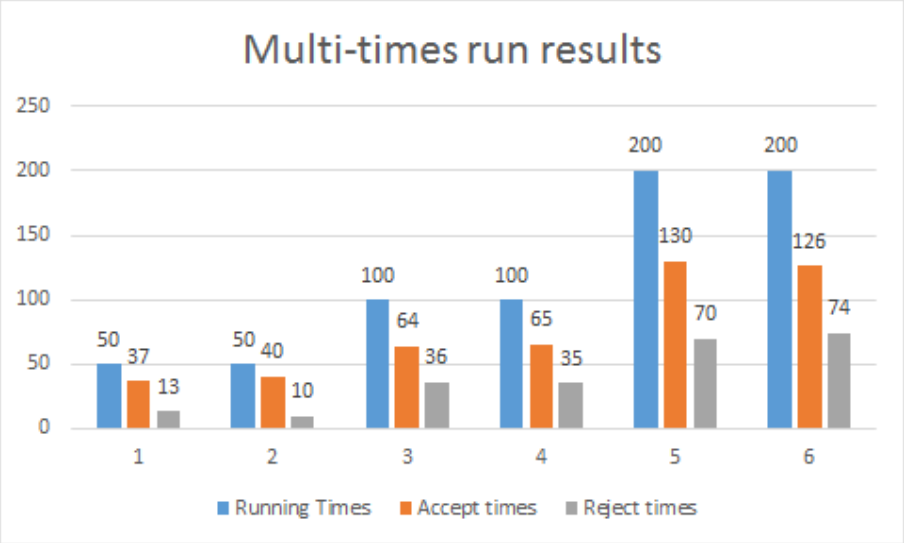
One time run result:

```
sample, full, naive, samplenumber: 23 21 31 175
sample, full, naive, samplenumber: 24 18 15 183
sample, full, naive, samplenumber: 25 13 16 198
sample, full, naive, samplenumber: 26 6 5 179
sample, full, naive, samplenumber: 27 24 20 191
sample, full, naive, samplenumber: 28 24 21 187
sample, full, naive, samplenumber: 29 8 8 183
all differences:
[ -3.  -6.  -7.  -2.  -5.  -5.   3.  -1.   0.  -1.  -7.  -7.   8.   4.  -6.
   0.   1.   4.  -1. -14. -13.  -9.  -9. -10.   3.  -3.   1.   4.   3.   0.]
z-score: -0.497234165333
bound: 2.04522964213
accept: classifiers have similar performance
[0]: bound:2.045230, z_score:-0.497234, accept
Total:
accept: 1 times
reject: 0 times
Press any key to continue . . .
```

We can see that the Gaussian Naïve Bayes' accurate rate is similar to Full Bayes, so the result is accept.

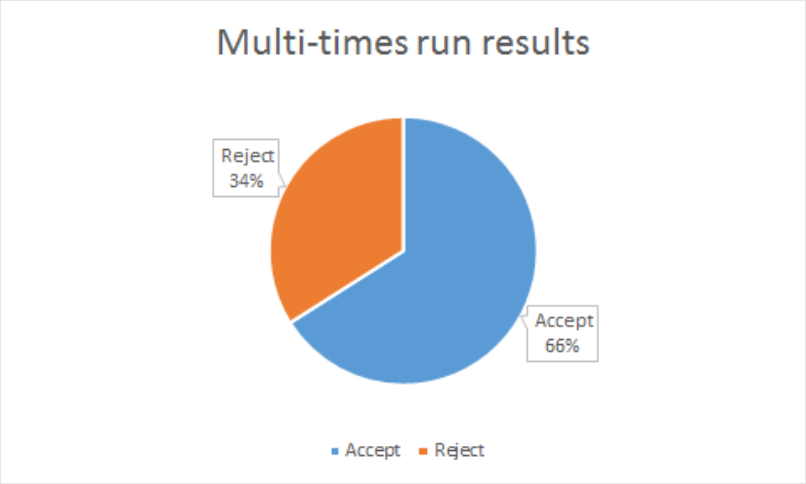
Feature more, I run the 30-Foder multi times and count the accept and reject result:

Run 50+50+100+100+200+200 = 700 times



Accept 462
Reject 238

Plot the pie chart:



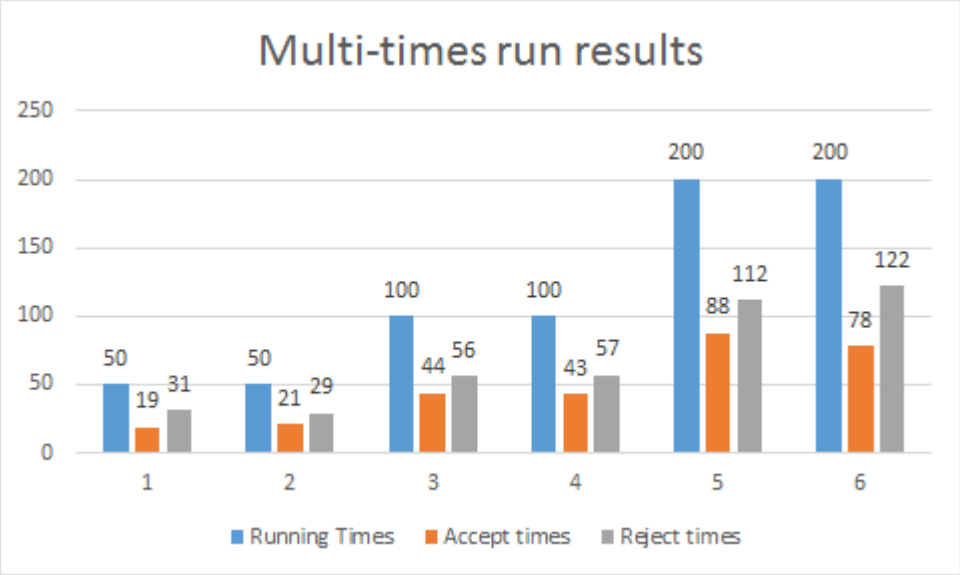
We can see that with almost 66% possibility that the Gaussian Naïve Bayes has the similar performance with the Full Bayes, it's a good performance.

3.3.2 Multinomial Naïve Bayes

One time run result:

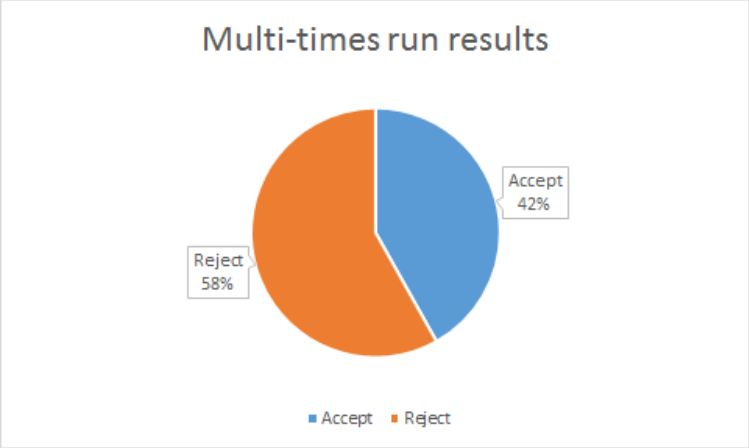
```
sample, full, naive, samplenumber: 18 5 47 181
sample, full, naive, samplenumber: 19 19 51 191
sample, full, naive, samplenumber: 20 37 70 178
sample, full, naive, samplenumber: 21 16 43 178
sample, full, naive, samplenumber: 22 23 55 171
sample, full, naive, samplenumber: 23 27 63 197
sample, full, naive, samplenumber: 24 11 50 171
sample, full, naive, samplenumber: 25 35 60 192
sample, full, naive, samplenumber: 26 20 53 185
sample, full, naive, samplenumber: 27 5 39 180
sample, full, naive, samplenumber: 28 20 45 182
sample, full, naive, samplenumber: 29 18 53 183
all differences:
[-31. -24. -29. -23. -34. -31. -37. -43. -12. -26. -31. -20. -25. -29. -24.
 -29. -28. -40. -42. -32. -33. -27. -32. -36. -39. -25. -33. -34. -25. -35.]
z-score: -3.78817473006
bound: 2.04522964213
reject: classifiers have significantly different performance
[0]: bound:2.045230, z_score:-3.788175, reject
Total:
accept: 0 times
reject: 1 times
Press any key to continue . . .
```

We can see there more error rate in Multinomial Bayes than Gaussian Bayes. Feature more, I run the 30-Foder multi times and count the accept and reject result: Run 50+50+100+100+200+200 = 700 times



Accept 293
Reject 407

Plot the pie chart:



We can see that with 42% possibility that the error rate can be acceptable for multinomial Bayes. Although its performance is not as good as the Gaussian Bayes, it's acceptable.

3.3.3 Bernoulli Naïve Bayes

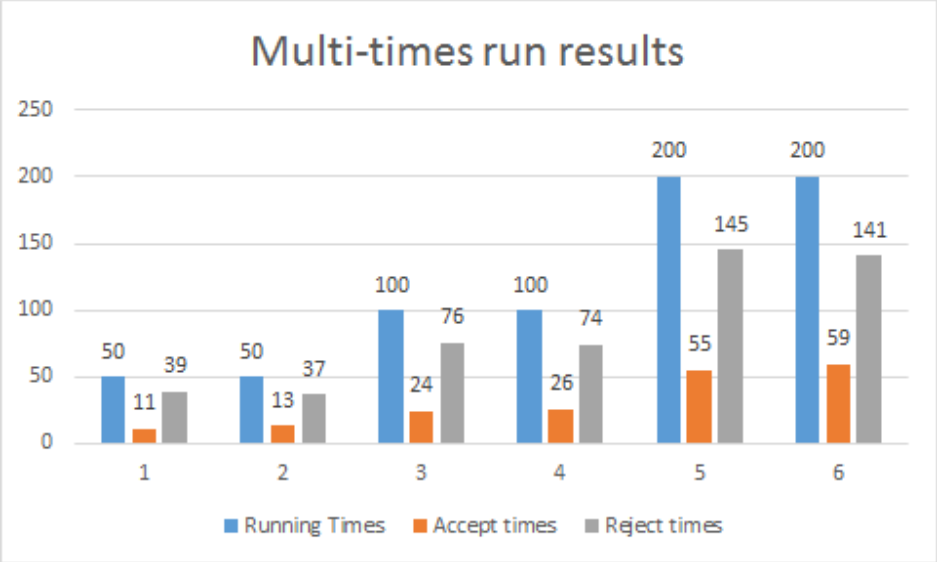
One time result:

```
sample, full, naive, samplenumber: 18 39 65 202
sample, full, naive, samplenumber: 19 27 61 182
sample, full, naive, samplenumber: 20 15 59 186
sample, full, naive, samplenumber: 21 29 54 181
sample, full, naive, samplenumber: 22 25 66 178
sample, full, naive, samplenumber: 23 19 67 182
sample, full, naive, samplenumber: 24 20 58 183
sample, full, naive, samplenumber: 25 7 57 175
sample, full, naive, samplenumber: 26 8 63 179
sample, full, naive, samplenumber: 27 36 79 188
sample, full, naive, samplenumber: 28 21 64 186
sample, full, naive, samplenumber: 29 35 82 191
all differences:
[-44. -33. -32. -33. -40. -47. -32. -37. -40. -45. -44. -46. -33. -39. -47.
 -38. -42. -31. -26. -34. -44. -25. -41. -48. -38. -50. -55. -43. -43. -47.]
z-score: -4.37461734392
bound: 2.04522964213
reject: classifiers have significantly different performance
[0]: bound:2.045230, z_score:-4.374617, reject
Total:
accept: 0 times
reject: 1 times
Press any key to continue . . .
```

We can see that Bernoulli Bayes has the largest error rate in the three models, which will be rejected.

Feature more, I run the 30-Foder multi times and count the accept and reject result:

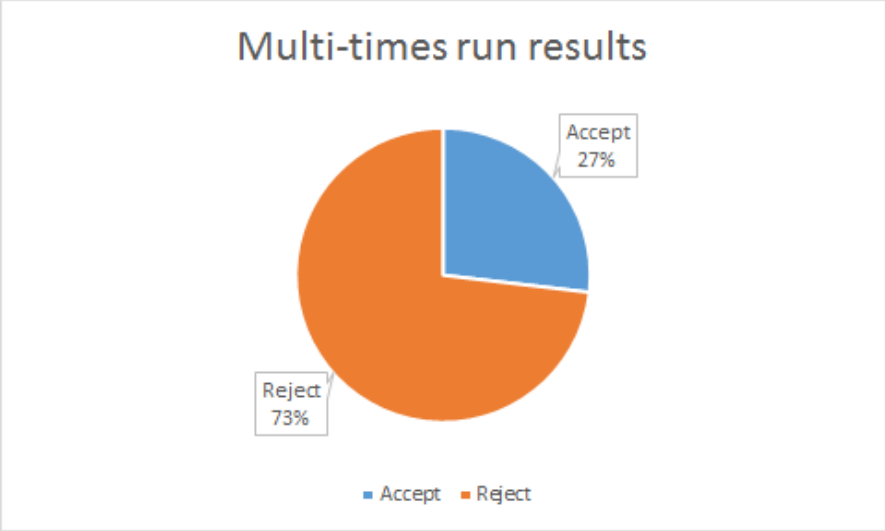
Run 50+50+100+100+200+200 = 700 times



Accept 188

Reject 512

Pie chart:



We can see that, with a high possibility(73%) , the result of Bernoulli Bayes will be rejected.

3.4 Conclusion and Analysis

By multi-time running the 30-Folder test, we can see that the performance for the Dressing Recommendation problem is:

Full Bayes > Gaussian Naïve Bayes > Multinomial Naïve Bayes > Bernoulli Naïve Bayes

The possible reason I analyze:

- The features are independent to each other, so Gaussian Naive Bayes has a similar right rate with Full Bayes

- Multinomial and Bernoulli Naive Bayes are more suitable for **test classifier**, so their right rate has a significantly different performance with Full Bayes
- Because Bernoulli has a penalization for a non-occurrence value, this can lead a huge difference when calculating the possibility for a feature.

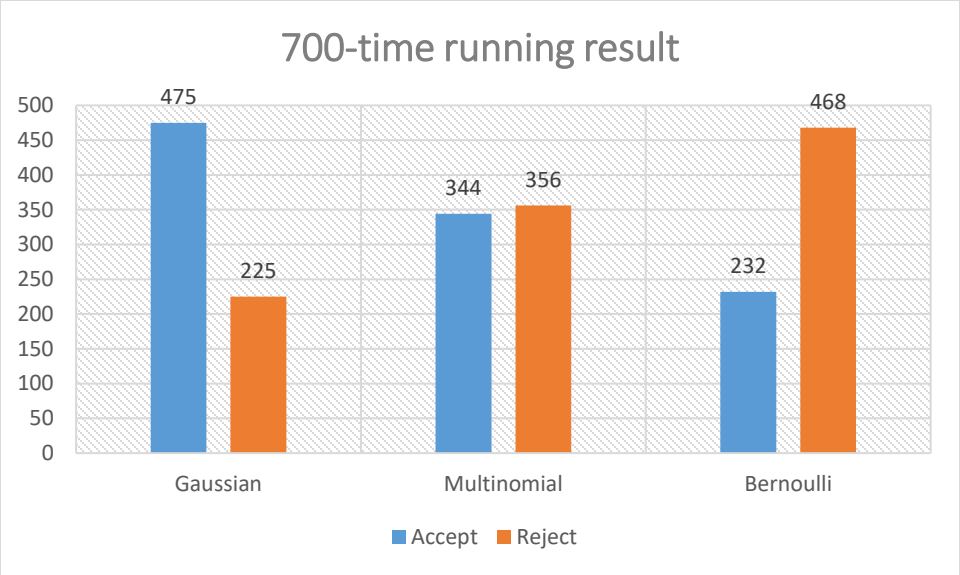
4. Sklearn.naive_bayes library implementation

There is also other third part python library has already implemented the three Naïve Bayes. I found the 'sklearn.naive_bayes' library and source code, using it to do some comparison. The source code is in the file: Sklearn_package.py

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB


def Gaussian_Text(X,Y,testcase):
    clf = GaussianNB()
    clf.fit(X, Y)
    GaussianNB()
    result = clf.predict(testcase)
    return result

def MultinomialNB_Text(X,Y,testcase):
    clf = MultinomialNB()
    clf.fit(X, Y)
    MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
    result = clf.predict(testcase)

def BernoulliNB_Text(X,Y,testcase):
    clf = BernoulliNB()
    clf.fit(X, Y)
    BernoulliNB(alpha=1.0, class_prior=None, fit_prior=True)
    result = clf.predict(testcase)
    return result
```

4.1 30-Folder Test

As above, I tested the three classifiers by running them 700 times, and count the accept and reject result. I got the results below:



From the result, we can see the accurate perform is like:

Gaussian > Multinomial > Bernoulli

The result is the same as my code's. So we can say that my code has the similar performance to the sklearn python library.

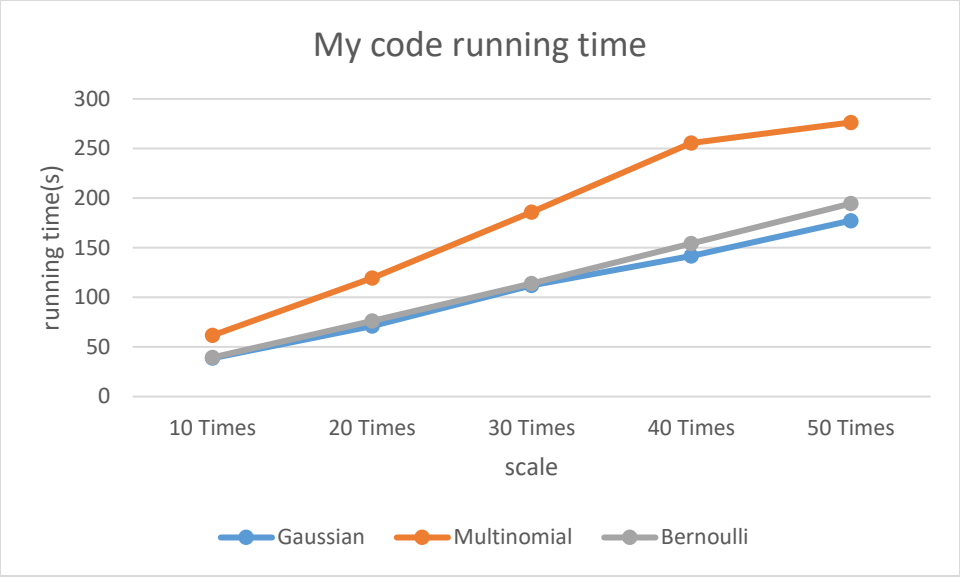
4.2 Efficiency Test

In this part, we will test the efficiency of the three different classifiers, not only the python library code, but also my code. And do some comparison.

We separately run different classifier in different 30-Folder test times, and count the running time.

- My code: time unit is second

	10 Times	20 Times	30 Times	40 Times	50 Times
Gaussian	38.6783	70.9228	111.9757	141.5693	177.3118
Multinomial	61.614	119.3233	185.9473	255.593	276.2683
Bernoulli	39.4343	76.2202	114.0329	154.4123	194.528

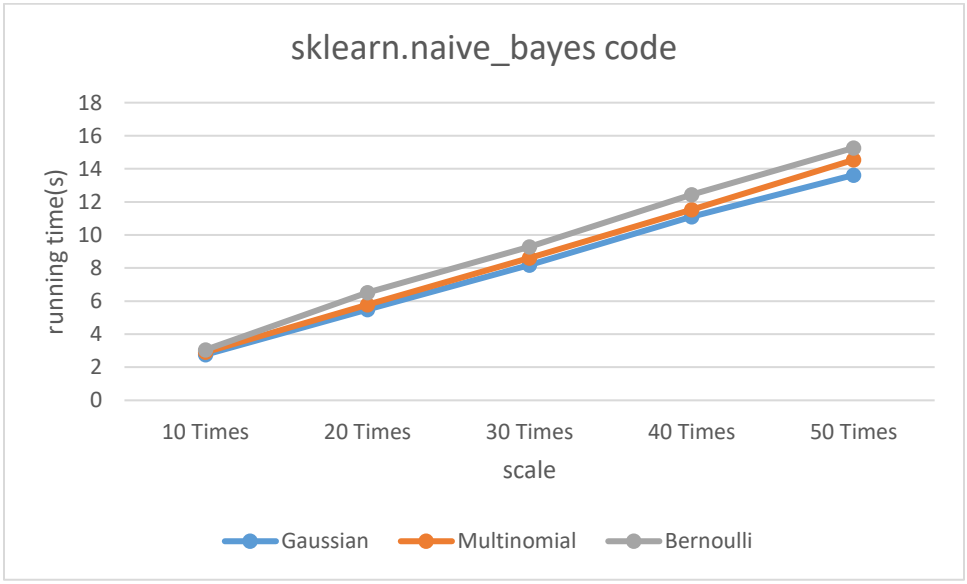


From the plot we can see, Gaussian and Bernoulli Bayes has almost the same running time under different test scales, while Multinomial Bayes has a larger running time than the other two.

- sklearn.naive_bayes code

we run multi-time test by sklearn.naive_bayes code, and count the running time, unit is second

	10 Times	20 Times	30 Times	40 Times	50 Times
Gaussian	2.7624	5.5001	8.1846	11.1086	13.623
Multinomial	2.909	5.7873	8.6066	11.5332	14.5385
Bernoulli	3.0556	6.5239	9.2793	12.4387	15.2627



From the plot we can see, the three classifier almost have the same running time by different test scales. And the running time is much smaller than my code. We will analyze the reason.

5. Analyze and Optimize

From the above test, we can see that although my code has the similar accurate performance with the sklearn.naive_bayes code, but the later has a huge advantage in the running time.

But analyze the source code of the sklearn.naive_bayes, I may find the reason.

For example, in the file 'naive_bayes.py' which is under 'Anaconda3\Lib\site-packages\sklearn'

`class MultinomialNB(BaseDiscreteNB): implemented the multinomial bayes.`

`The likelihood function is like this:`

```
def _joint_log_likelihood(self, X):
    """Calculate the posterior log probability of the samples X"""
    check_is_fitted(self, "classes_")

    X = check_array(X, accept_sparse='csr')
    return (safe_sparse_dot(X, self.feature_log_prob_.T)
            + self.class_log_prior_)
```

Feature more, if we go to the '`safe_sparse_dot`' `function:`

```
def safe_sparse_dot(a, b, dense_output=False):
    """Dot product that handle the sparse matrix case correctly

    Uses BLAS GEMM as replacement for numpy.dot where possible
    to avoid unnecessary copies.
    """
    if issparse(a) or issparse(b):
        ret = a * b
        if dense_output and hasattr(ret, "toarray"):
            ret = ret.toarray()
        return ret
    else:
        return fast_dot(a, b)
```

We can see : "`Uses BLAS GEMM as replacement for numpy.dot where possible to avoid unnecessary copies.`"

As the same, in all the Bayes in sklearn.naive_bayes code, it doesn't use numpy to store the data, while my code always use numpy and may use it repeatedly.

I think that's the reason why my code is much smaller than the sklearn.naive_bayes code: **mass of using numpy causes a lot of unnecessary copies, which slow down the calculation.**

6. Conclusion

By doing this final project, I have a further understanding for the Bayes classifier, and learn much more about the Naïve Bayes.

At the same time, I got some conclusion:

- If the features are independent to each other, Gaussian Naive Bayes has a similar right rate with Full Bayes
- Multinomial and Bernoulli Naive Bayes are more suitable for **test classifier**, so for the Dressing Recommendation, their right rate has a significantly different performance with Full Bayes
- Because Bernoulli has a penalization for a non-occurrence value, this can lead a huge difference when calculating the possibility for a feature.
- Mass of using numpy in calculation can lead a lot of unnecessary copies, which can slow down the efficiency.

7. Reference

[1] Wikipedia: Naive Bayes classifier

[2] Scikit-learn: 1.9. Naive Bayes

[3] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. Introduction to Information Retrieval Cambridge University Press. 2008: The Bernoulli model

[4] Sebastian Raschka, Naive Bayes and Text Classification Oct 4, 2014