Lab Assignment3

1.(a)

```
from sklearn.datasets import fetch_lfw_people
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

mnist = fetch_openml('mnist_784')
lfw = fetch_lfw_people(min_faces_per_person=60)

x_lfw=lfw.data
y_lfw=lfw.target
x_lfw_train, x_lfw_test, y_lfw_train, y_lfw_test = train_test_split(x_lfw,y_lfw,
test_size = 0.35, random_state=2)

x_mnist=mnist.data
y_mnist=mnist.target
x_mnist_train, x_mnist_test, y_mnist_train, y_mnist_test =
train_test_split(x_mnist,y_mnist, test_size = 0.35, random_state=2)
```

(b)

#MNIST KNN

```
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline

accuracy=[]

components = range(1,50)
for n in components:
    pca = PCA(n_components = n).fit(x_mnist_train)
    x_train_reduced = pca.transform(x_mnist_train)
```

```
    x_test_reduced = pca.transform(x_mnist_test)
    model = KNeighborsClassifier()
    model.fit(x_train_reduced, y_mnist_train)
    y_pred = model.predict(x_test_reduced)
    accuracy.append(accuracy_score(y_pred,y_mnist_test))

plt.plot(components, accuracy, label = "KNNAccuracy")
plt.xlabel("Number of principle components")
plt.ylabel("Accuracy")
plt.legend()
```

#MNIST Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

accuracy=[]

components = range(1,100)
for n in components:
    pca = PCA(n_components = n)
    pca.fit(x_mnist_train)
    x_train_reduced = pca.transform(x_mnist_train)
    x_test_reduced = pca.transform(x_mnist_test)
    model = GaussianNB()
    model.fit(x_train_reduced, y_mnist_train)
    y_pred = model.predict(x_test_reduced)
    accuracy.append(accuracy_score(y_pred,y_mnist_test))

plt.plot(components, accuracy, label = "BayesAccuracy")
plt.xlabel("Number of principle components")
plt.ylabel("Accuracy")
plt.legend()
```

#LFW KNN

```
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline

accuracy=[]

components = range(1,100)
for n in components:
    pca = PCA(n_components = n).fit(x_lfw_train)
    x_train_reduced = pca.transform(x_lfw_train)
    x_test_reduced = pca.transform(x_lfw_test)
    model = KNeighborsClassifier()
    model.fit(x_train_reduced, y_lfw_train)
    y_pred = model.predict(x_test_reduced)
    accuracy.append(accuracy_score(y_pred,y_lfw_test))

plt.plot(components, accuracy, label = "KNNAccuracy")
plt.xlabel("Number of principle components")
plt.ylabel("Accuracy")
plt.legend()
```

#LFW Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

accuracy=[]

components = range(1,100)
for n in components:
    pca = PCA(n_components = n)
    pca.fit(x_lfw_train)
    x_train_reduced = pca.transform(x_lfw_train)
```

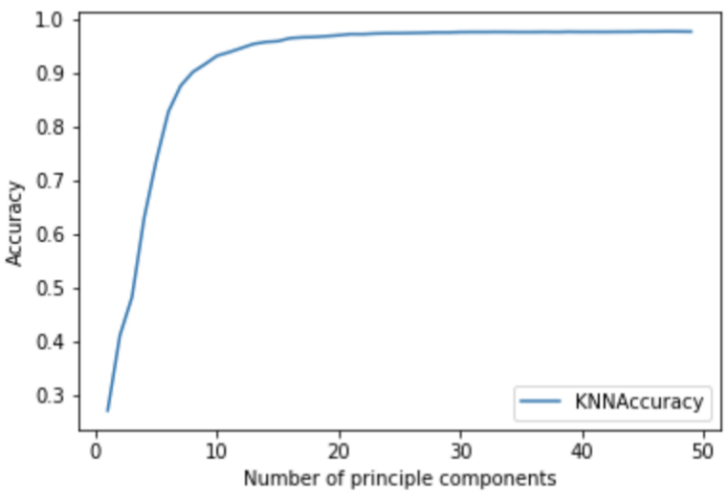
```
    x_test_reduced = pca.transform(x_lfw_test)
    model = GaussianNB()
    model.fit(x_train_reduced, y_lfw_train)
    y_pred = model.predict(x_test_reduced)
    accuracy.append(accuracy_score(y_pred,y_lfw_test))

plt.plot(components, accuracy, label = "BayesAccuracy")
plt.xlabel("Number of principle components")
plt.ylabel("Accuracy")
plt.legend()
```

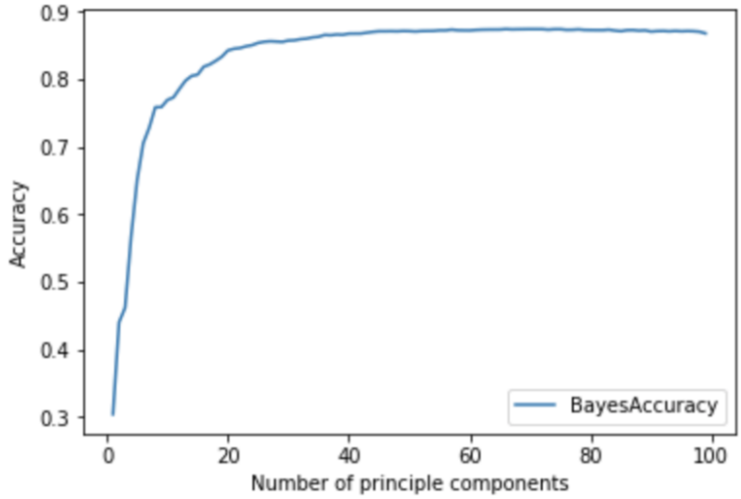
(c)

For MNIST dataset, according to the plot, we can find KNN is better, because the accuracy of KNN is higher. And when the number of principle component is almost 20, the accuracy reaches highest.

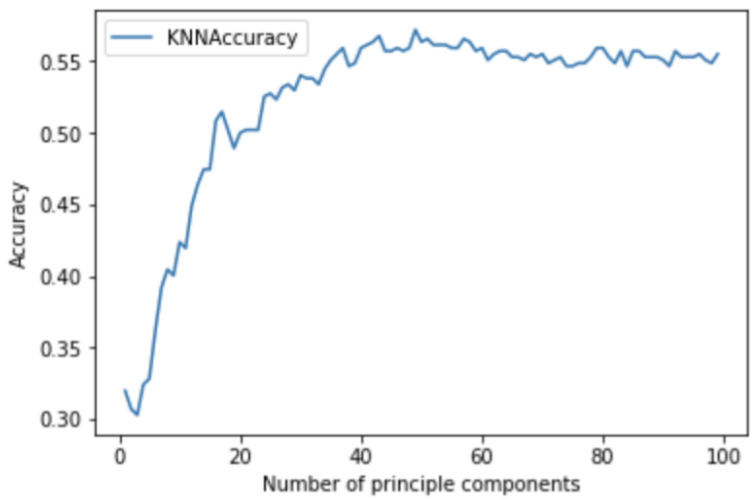
For LFW dataset, according to the plot, we can find Naïve Bayes is better, because the accuracy of Naïve Bayes is higher, which is close to 0.75

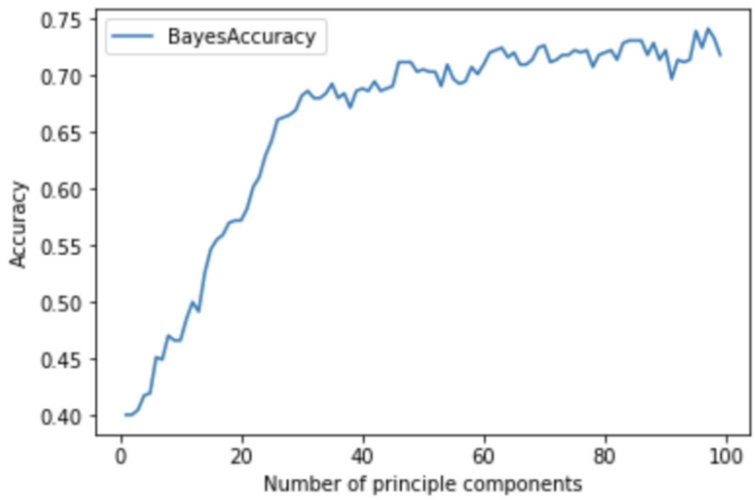
#MNIST





#LFW





(d) About MNIST dataset, in the KNN method, when the feature dimension size is 20, the method performs better. When the feature dimension size is lower than 20, the method performs worse. In the Naïve Bayes method, when the feature dimension size is 40, the method performs better. When the feature dimension size is lower than 40, the method performs worse. About LFW dataset, in the KNN method, when the feature dimension size is 50, the method performs better. When the feature dimension size is lower than 50, the method performs worse. In the Naïve Bayes method, when the feature dimension size is 100, the method performs better. When the feature dimension size is lower than 40, the performance of the method decreases gradually and between 40 and 100, the performance of the method does not change greatly.

The choice of feature dimension size is decided by many factors, including number of samples, number of features, classification methods and so on. When feature dimension size reaches the appropriate size, the performance of method is better.

By observing the classification performance across the different datasets and different classification algorithms, different datasets have different parameters, thus, they are suitable to different classification methods. In terms of PCA, the performance of different methods and datasets reaches better when there are different feature dimension sizes. Thus, focusing on different datasets, it is important to try different methods and different feature dimension sizes, compare the results and choose the best parameters and methods.

2.(a)
```
#movie review dataset
import random
import nltk
from nltk.corpus import movie_reviews
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing
import numpy as np

nltk.download('movie_reviews')

documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
```

```
random.shuffle(documents)

x_review = [" ".join(documents[i][0]) for i in range (len(documents))]
y_review = [documents[i][1] for i in range (len(documents))]

count = CountVectorizer()
x_review_bag = count.fit_transform(x_review).toarray()

le = preprocessing.LabelEncoder()
y_review_bag = le.fit_transform(y_review)

# brown dataset
from nltk.corpus import brown
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing
import numpy as np

nltk.download('brown')

documents = [(list(brown.words(fileid)), category)
             for category in brown.categories()
              for fileid in brown.fileids(category)]

random.shuffle(documents)

x_brown = [" ".join(documents[i][0]) for i in range (len(documents))]
y_brown = [documents[i][1] for i in range (len(documents))]

count = CountVectorizer()
x_brown_bag = count.fit_transform(x_brown).toarray()
le = preprocessing.LabelEncoder()
y_brown_bag = le.fit_transform(y_brown)
```

(b) # movie review dataset

```
# Majority vote classifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier

kfold = KFold(n_splits=5, shuffle = True, random_state=2)

clf1 = DecisionTreeClassifier()
clf2 = KNeighborsClassifier()
clf3 = GaussianNB()
eclf = VotingClassifier(estimators=[('decisiontree', clf1), ('knn', clf2), ('gnb', clf3)],
voting = 'hard')

scores = cross_val_score(eclf, x_review_bag, y_review_bag, cv=kfold)
scores.mean()
# Bagging
from sklearn.ensemble import BaggingClassifier

bagging = BaggingClassifier(clf3)
scores = cross_val_score(bagging, x_review_bag, y_review_bag, cv=kfold)
scores.mean()

# Boosting
from sklearn.ensemble import AdaBoostClassifier

adabooster = AdaBoostClassifier(n_estimators = 50)
scores = cross_val_score(adabooster, x_review_bag, y_review_bag, cv=kfold)
scores.mean()
```

```
#decision tree
decisiontree_scores = cross_val_score(clf1, x_review_bag, y_review_bag, cv=kfold)
decisiontree_scores.mean()

#KNN
knn_scores = cross_val_score(clf2, x_review_bag, y_review_bag, cv=kfold)
knn_scores.mean()

# Naïve Bayes
gnb_scores = cross_val_score(clf3, x_review_bag, y_review_bag, cv=kfold)
gnb_scores.mean()

#brown dataset
# Majority vote classifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier

kfold = KFold(n_splits=5, shuffle = True, random_state=2)

clf1 = DecisionTreeClassifier()
clf2 = KNeighborsClassifier()
clf3 = GaussianNB()
eclf = VotingClassifier(estimators=[('decisiontree', clf1), ('knn', clf2), ('gnb', clf3)],
voting = 'hard')

scores = cross_val_score(eclf, x_brown_bag, y_brown_bag, cv=kfold)
scores.mean()

#Bagging
```

```
from sklearn.ensemble import BaggingClassifier

bagging = BaggingClassifier(clf3)
scores = cross_val_score(bagging, x_brown_bag, y_brown_bag, cv=kfold)
scores.mean()

#Boosting
from sklearn.ensemble import AdaBoostClassifier

adabooster = AdaBoostClassifier(n_estimators = 50)
scores = cross_val_score(adabooster, x_brown_bag, y_brown_bag, cv=kfold)
scores.mean()

#decision tree
decisiontree_scores = cross_val_score(clf1, x_brown_bag, y_brown_bag, cv=kfold)
decisiontree_scores.mean()

#KNN
knn_scores = cross_val_score(clf2, x_brown_bag, y_brown_bag, cv=kfold)
knn_scores.mean()

#Naive Bayes
gnb_scores = cross_val_score(clf3, x_brown_bag, y_brown_bag, cv=kfold)
gnb_scores.mean()
```

(c)

	Decision tree	KNN	Naïve Bayes	Majority vote	Bagging	Boosting
movie review	0.646	0.6215	0.6525	0.695	0.6735	0.79

	Decision	KNN	Naïve	Majority	Bagging	Boosting

	tree		Bayes	vote		
brown	0.298	0.312	0.318	0.354	0.27	0.218

(d)In terms of ensemble methods, about movie review dataset, boosting method is better and bagging is worse. About brown dataset, majority voting is better and boosting method is worse.

Different datasets have different parameters, including number of samples, number of classes and so on. Thus, focusing on different datasets, different methods have different performance. The ensemble methods combine the predictions of several base estimators built with a given learning algorithm. Thus, usually, ensemble methods can improve generalizability and robustness over a single estimator.

About movie review dataset, ensemble methods (Majority Vote, Bagging, Boosting) perform better than non-ensemble methods (Decision tree, KNN, Naïve Bayes). About brown dataset, non-ensemble methods are better than Bagging and Boosting methods, but they perform worse than Majority vote.

Focusing on different datasets, because of different number of samples, dimensionalities, classes and other parameters, they are appropriate for different methods. Thus, in the process of choosing methods, it is important to compare the results and choose the best method.