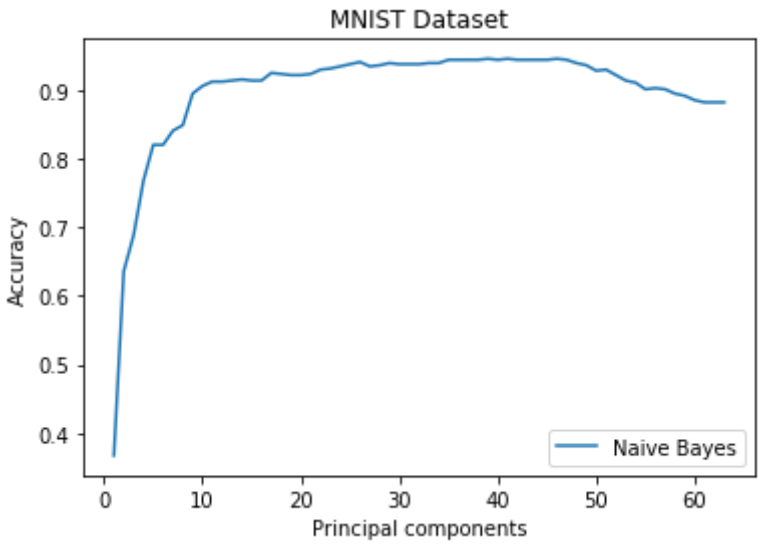
In [1]:
```
# 1. Image Classification with Dimensionality Reduction
# (a) Prepare two image classification datasets
from sklearn.datasets import load_digits
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split

mnist = load_digits()
X_mnist = mnist.data
y_mnist = mnist.target
X_mnist_train, X_mnist_test, y_mnist_train, y_mnist_test = train_test_sp
lit(X_mnist, y_mnist, test_size=0.35, random_state=42)

lfw = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X_lfw = lfw.data
y_lfw = lfw.target
X_lfw_train, X_lfw_test, y_lfw_train, y_lfw_test = train_test_split(X_lf
w, y_lfw, test_size=0.35, random_state=42)
```

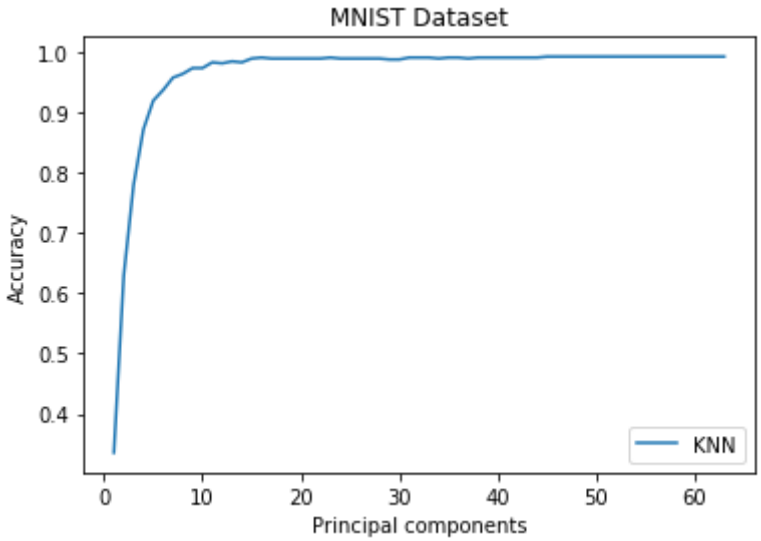
In [2]:
```
# (b) Train and evaluate classification algorithms
# Train and evaluate MNIST using Naive Bayes
from sklearn.decomposition import PCA

from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline

mnist_features = len(X_mnist[0])

accuracy = []
principal = []

for i in range(1, 64):

    pca = PCA(n_components=i)
    pca.fit(X_mnist_train)
    X_mnist_train_reduced = pca.transform(X_mnist_train)
    X_mnist_test_reduced = pca.transform(X_mnist_test)
    gaussian_model = GaussianNB()
    gaussian_model.fit(X_mnist_train_reduced, y_mnist_train)
    y_predictedNB = gaussian_model.predict(X_mnist_test_reduced)
    accuracy.append(accuracy_score(y_predictedNB, y_mnist_test))
    principal.append(i)

plt.plot(principal, accuracy, label='Naive Bayes')
plt.ylabel('Accuracy')
plt.xlabel('Principal components')
plt.title('MNIST Dataset')
plt.legend()
```

Out[2]: <matplotlib.legend.Legend at 0x1a24261518>



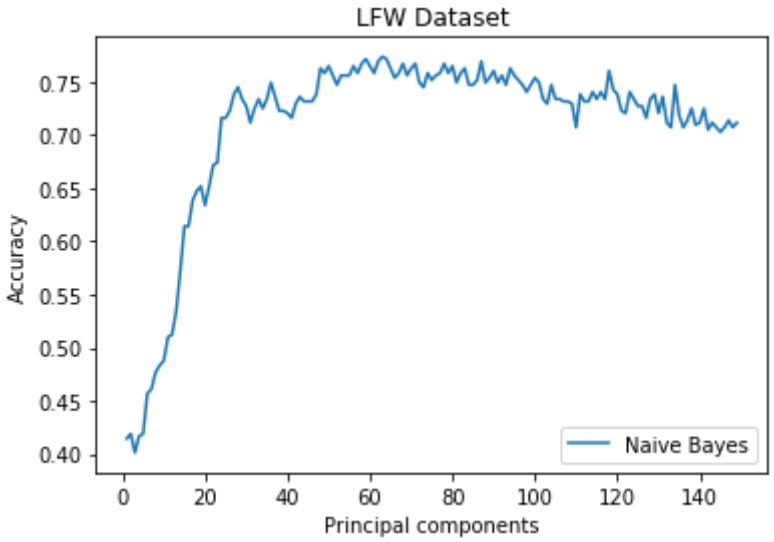
In [3]:
```
# Train and evaluate MNIST using KNN

accuracy = []
principal = []

for i in range(1, 64):

    pca = PCA(n_components=i)
    pca.fit(X_mnist_train)
    X_mnist_train_reduced = pca.transform(X_mnist_train)
    X_mnist_test_reduced = pca.transform(X_mnist_test)
    knn_model = KNeighborsClassifier()
    knn_model.fit(X_mnist_train_reduced, y_mnist_train)
    y_predictedKNN = knn_model.predict(X_mnist_test_reduced)
    accuracy.append(accuracy_score(y_predictedKNN, y_mnist_test))
    principal.append(i)

plt.plot(principal, accuracy, label='KNN')
plt.ylabel('Accuracy')
plt.xlabel('Principal components')
plt.title('MNIST Dataset')
plt.legend()
```

Out[3]:
```
<matplotlib.legend.Legend at 0x11d546c50>
```



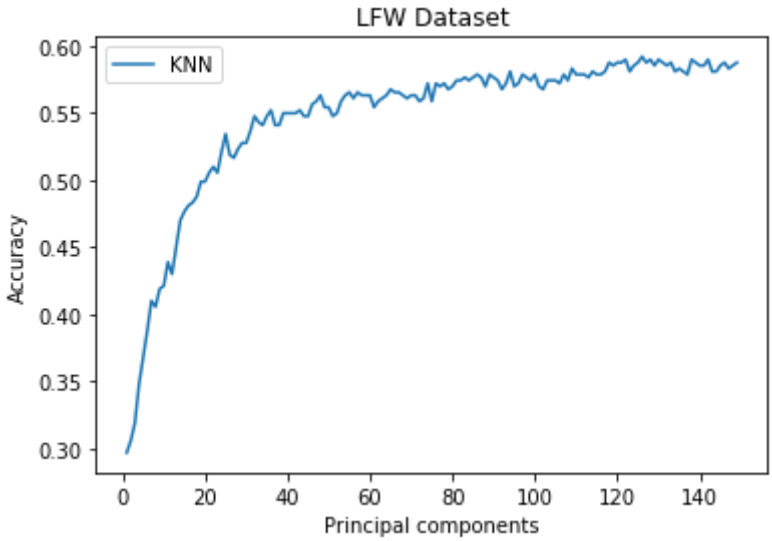
In [4]:
```
# Train and evaluate LFW using Naive Bayes

accuracy = []
principal = []

for i in range(1, 150):

    pca = PCA(n_components=i)
    pca.fit(X_lfw_train)
    X_lfw_train_reduced = pca.transform(X_lfw_train)
    X_lfw_test_reduced = pca.transform(X_lfw_test)
    gaussian_model = GaussianNB()
    gaussian_model.fit(X_lfw_train_reduced, y_lfw_train)
    y_predicted = gaussian_model.predict(X_lfw_test_reduced)
    accuracy.append(accuracy_score(y_predicted, y_lfw_test))
    principal.append(i)

plt.plot(principal, accuracy, label='Naive Bayes')
plt.ylabel('Accuracy')
plt.xlabel('Principal components')
plt.title('LFW Dataset')
plt.legend()
```

Out[4]:
```
<matplotlib.legend.Legend at 0x11d2d3160>
```



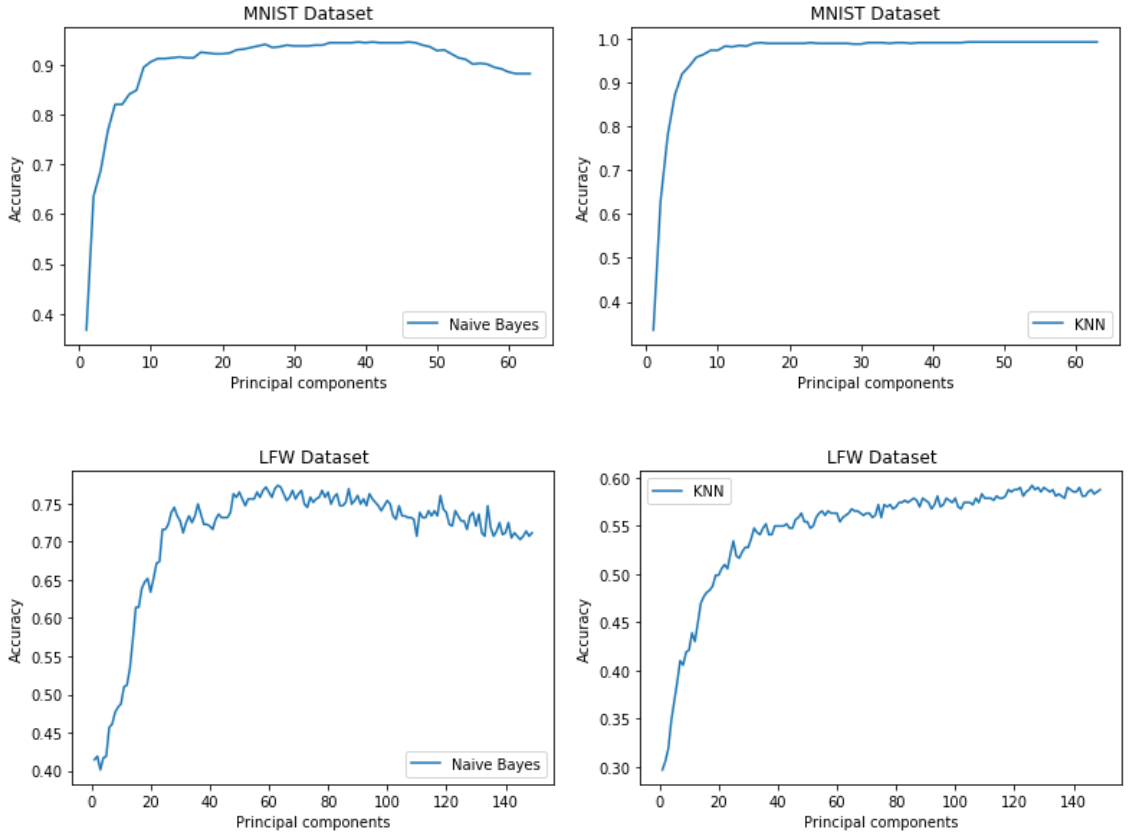
In [5]:
```
# Train and evaluate LFW using KNN

accuracy = []
principal = []

for i in range(1, 150):
    pca = PCA(i)
    pca.fit(X_lfw_train)
    X_lfw_train_reduced = pca.transform(X_lfw_train)
    X_lfw_test_reduced = pca.transform(X_lfw_test)
    knn_model = KNeighborsClassifier()
    knn_model.fit(X_lfw_train_reduced, y_lfw_train)
    y_predicted = knn_model.predict(X_lfw_test_reduced)
    accuracy.append(accuracy_score(y_predicted, y_lfw_test))
    principal.append(i)

plt.plot(principal, accuracy, label='KNN')
plt.ylabel('Accuracy')
plt.xlabel('Principal components')
plt.title('LFW Dataset')
plt.legend()
```

Out[5]:
```
<matplotlib.legend.Legend at 0x1a2419de80>
```



In [ ]:

(c)



The predictive performance is shown in pictures above. The top two plots use the hand writing digits dataset with the left one of Naïve Bayes and right one of KNN. I use 63 different sizes of feature dimension. The bottom two use LFW face dataset with the left one of Naïve Bayes and right one of KNN. I try 149 different sizes of feature dimension here. From the graph, we can see the curve grows quickly in the beginning and after one point, it begins to grow slowly.

(d) Applying PCA on the classification model, we can reduce the dimension so that we can save store and running time. It can improve machine learning algorithm performance because of the removal of multi-collinearity.

For the number recognition dataset, Naïve Bayes algorithm works well when the number of principal components is around 12, KNN works well when the number of principal components is around 8. For the face recognition dataset, Naïve Bayes works well when the number of principal components is around 30, KNN works well when the number of principal components is around 30. If the number is too small, the accuracy will be low. If the number is too large, it will take much more computing. Both situations perform worse than previous one. So it's important to find the suitable number.

Comparing different classification algorithm for the same dataset, we can find they have different "best number" of principal components and they have different accuracies. For example, the average accuracy of KNN is higher than Naïve Bayes in the first dataset while the

situation is reversed in the second dataset. From the plots, we can also see that same algorithm works differently on different dataset. I think it's important to choose suitable algorithm every time we train different dataset. Also, it's important to tune the parameters.

In [1]:

```
# 2. Ensemble Learning
# (a) Load two text-based classification datasets and pre-processing
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import TfidfTransformer

airline = pd.read_csv('dataset/Tweets.csv')
spam = pd.read_csv('dataset/YouTube-Spam-Collection-v1/Youtube01-Psy.cs
v')

count = CountVectorizer()
X_airline_bag = count.fit_transform(airline['text'][:2000]).toarray()

X_spam_bag = count.fit_transform(spam['CONTENT']).toarray()

tfidf = TfidfTransformer(use_idf=True, norm = 'l2', smooth_idf=True)
np.set_printoptions(precision=2)

X_airline_tfidf = tfidf.fit_transform(count.fit_transform(airline['text'
][:2000])).toarray()
y_airline = airline['airline_sentiment'][:2000]

X_spam_tfidf = tfidf.fit_transform(count.fit_transform(spam['CONTENT']))
.toarray()
y_spam = spam['CLASS']

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

In [2]:

```
# (b)
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

clf1 = LogisticRegression()
clf2 = GaussianNB()
clf3 = KNeighborsClassifier()
eclf = VotingClassifier(estimators=[('lr', clf1), ('gnb', clf2), ('knn',
clf3)], voting='hard')
bagging = BaggingClassifier(max_samples=50)
boosting = AdaBoostClassifier(n_estimators =200)

for i in [clf1, clf2, clf3, eclf, bagging, boosting]:
    accuracies = cross_val_score(i, X_airline_tfidf, y_airline, cv=kfold
)
    print(accuracies.mean())
```

```
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/weig
ht_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an int
ernal NumPy module and should not be imported. It will be removed in a
future NumPy release.
  from numpy.core.umath_tests import inner1d
```

```
0.7144999999999999
0.657
0.705
```

```
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

```
0.7200000000000001
0.6585000000000001
0.679
```

In [3]:
```python
for i in [clf1, clf2, clf3, eclf, bagging, boosting]:
    accuracies = cross_val_score(i, X_spam_tfidf, y_spam, cv=kfold)
    print(accuracies.mean())
```

```
0.962857142857143
0.8714285714285716
0.917142857142857
```

```
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/preprocessin
g/label.py:151: DeprecationWarning: The truth value of an empty array i
s ambiguous. Returning False, but in future this will result in an erro
r. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

```
0.9571428571428573
0.8771428571428572
0.9485714285714286
```

In [ ]:

(c)

	Logistic Regression	Naïve Bayes	KNN	Majority Voting	Bagging	Boosting
Mean accuracy	0.71	0.66	0.71	0.72	0.66	0.68

Tweets.csv

	Logistic Regression	Naïve Bayes	KNN	Majority Voting	Bagging	Boosting
Mean accuracy	0.96	0.87	0.92	0.96	0.88	0.95

Youtube01-Psy.csv

Tweets.csv is a dataset that collections text sentiment of texts sent to airline. It contains more than ten thousand rows which is out of my computer's competence, so I only use two thousand rows of it. Youtube01-Psy.csv is a dataset about YouTube spam text.

(d) From the table above, we can see that majority voting has better performance. Naïve Bayes has worse performance than the others. Since majority voting returns most popular prediction from multiple prediction algorithms and reduces the likelihood of unfortunate instance, it has better performance than the others. The performance of Naïve Bayes is not that good here, I guess that's because Naïve Bayes is not that suitable for the dataset here. The amount of data is not that sufficient here, so there is much deviation here.
Compared to non-ensemble methods, ensemble methods work better than or equivalent to non-ensemble methods. Although the advantage is not that obvious here, I do notice how the accuracy improves after I tune the parameters. It's necessary to tune the parameters to make an algorithm have better performance.
By observing the classification performance across the different datasets, I notice the accuracy has close relation to the dataset itself. For example, the mean accuracy of the first dataset is lower than the second dataset. The classification performance is influenced by the content and quality of that dataset.