February 27, 2018

Lab Assignment 3

Sanchit Singhal

INF 385T – Introduction to Machine Learning with Danna Gurari

Spring 2018

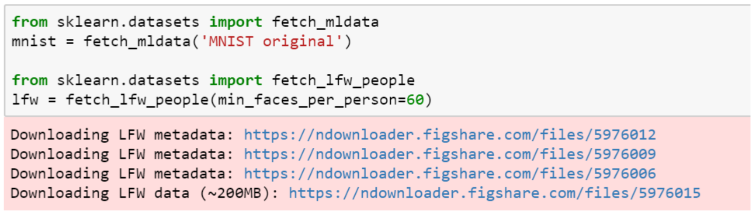
School of Information

Hyperlink to code: https://introtoml-sanchit1276.notebooks.azure.com/nb/notebooks/IntroToML/LabAssignment3.ipynb

1. Image Classification with Dimensionality Reduction

a) Prepare two classification datasets

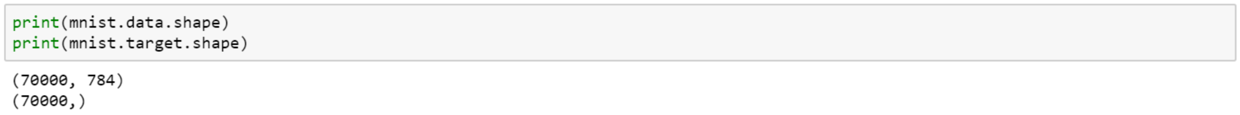
Load data from external sources:



```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')

from sklearn.datasets import fetch_lfw_people
lfw = fetch_lfw_people(min_faces_per_person=60)
```

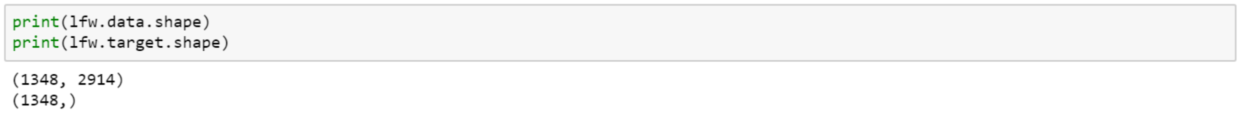
```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
```

Split data using 70/30 split:



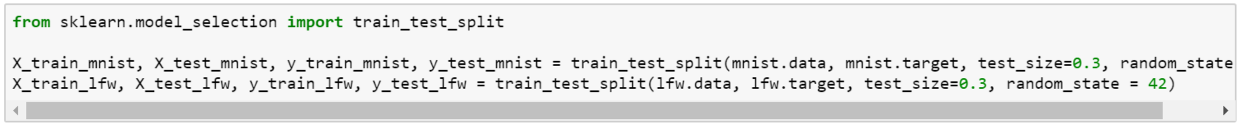
```
print(mnist.data.shape)
print(mnist.target.shape)
```

```
(70000, 784)
(70000,)
```

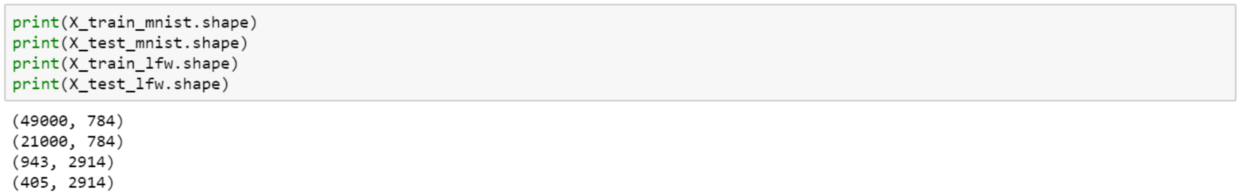


```
print(lfw.data.shape)
print(lfw.target.shape)
```

```
(1348, 2914)
(1348,)
```



```
from sklearn.model_selection import train_test_split

X_train_mnist, X_test_mnist, y_train_mnist, y_test_mnist = train_test_split(mnist.data, mnist.target, test_size=0.3, random_state
X_train_lfw, X_test_lfw, y_train_lfw, y_test_lfw = train_test_split(lfw.data, lfw.target, test_size=0.3, random_state = 42)
```

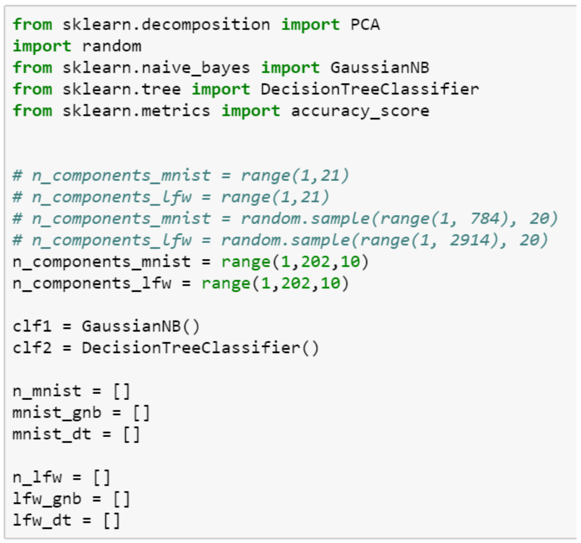


```
print(X_train_mnist.shape)
print(X_test_mnist.shape)
print(X_train_lfw.shape)
print(X_test_lfw.shape)
```

```
(49000, 784)
(21000, 784)
(943, 2914)
(405, 2914)
```

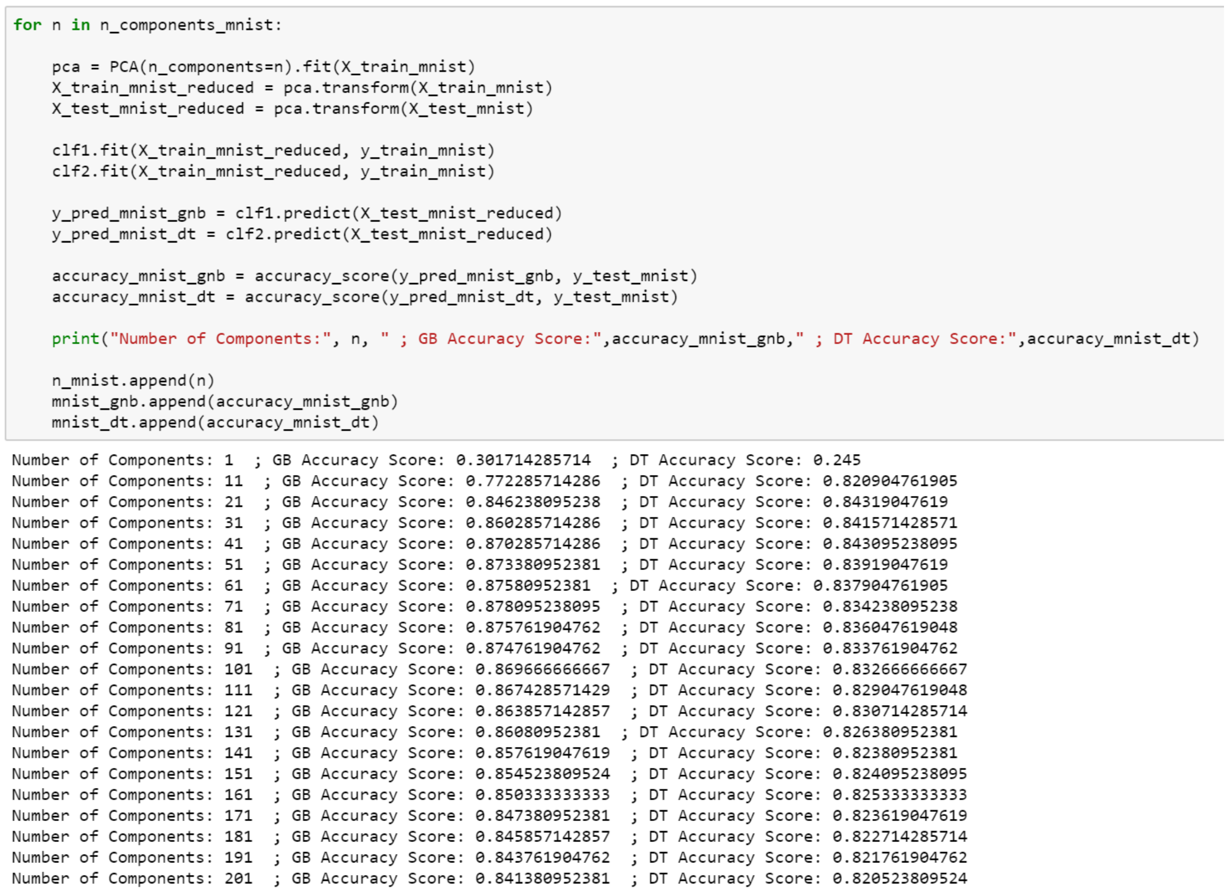
b) Training and Evaluation of each dataset with 20 different feature dimension sizes on 2 classifiers

Preparing Naïve Bayes and Decision Tree classifiers, building array of component sizes:



```
from sklearn.decomposition import PCA
import random
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


# n_components_mnist = range(1,21)
# n_components_lfw = range(1,21)
# n_components_mnist = random.sample(range(1, 784), 20)
# n_components_lfw = random.sample(range(1, 2914), 20)
n_components_mnist = range(1,202,10)
n_components_lfw = range(1,202,10)

clf1 = GaussianNB()
clf2 = DecisionTreeClassifier()

n_mnist = []
mnist_gnb = []
mnist_dt = []

n_lfw = []
lfw_gnb = []
lfw_dt = []
```

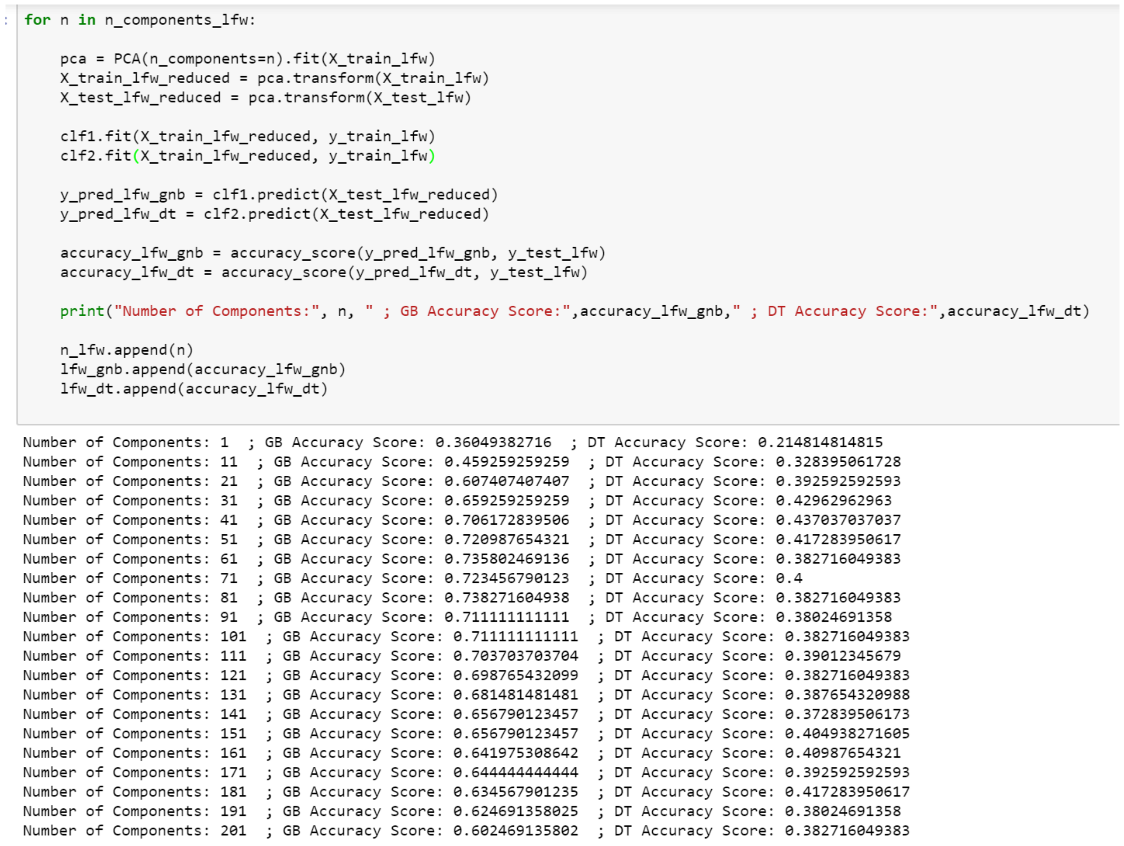
MNIST:



```python
for n in n_components_mnist:

    pca = PCA(n_components=n).fit(X_train_mnist)
    X_train_mnist_reduced = pca.transform(X_train_mnist)
    X_test_mnist_reduced = pca.transform(X_test_mnist)

    clf1.fit(X_train_mnist_reduced, y_train_mnist)
    clf2.fit(X_train_mnist_reduced, y_train_mnist)

    y_pred_mnist_gnb = clf1.predict(X_test_mnist_reduced)
    y_pred_mnist_dt = clf2.predict(X_test_mnist_reduced)

    accuracy_mnist_gnb = accuracy_score(y_pred_mnist_gnb, y_test_mnist)
    accuracy_mnist_dt = accuracy_score(y_pred_mnist_dt, y_test_mnist)

    print("Number of Components:", n, " ; GB Accuracy Score:",accuracy_mnist_gnb," ; DT Accuracy Score:",accuracy_mnist_dt)

    n_mnist.append(n)
    mnist_gnb.append(accuracy_mnist_gnb)
    mnist_dt.append(accuracy_mnist_dt)
```

```
Number of Components: 1  ; GB Accuracy Score: 0.301714285714  ; DT Accuracy Score: 0.245
Number of Components: 11  ; GB Accuracy Score: 0.772285714286  ; DT Accuracy Score: 0.820904761905
Number of Components: 21  ; GB Accuracy Score: 0.846238095238  ; DT Accuracy Score: 0.84319047619
Number of Components: 31  ; GB Accuracy Score: 0.860285714286  ; DT Accuracy Score: 0.841571428571
Number of Components: 41  ; GB Accuracy Score: 0.870285714286  ; DT Accuracy Score: 0.843095238095
Number of Components: 51  ; GB Accuracy Score: 0.873380952381  ; DT Accuracy Score: 0.83919047619
Number of Components: 61  ; GB Accuracy Score: 0.87580952381  ; DT Accuracy Score: 0.837904761905
Number of Components: 71  ; GB Accuracy Score: 0.878095238095  ; DT Accuracy Score: 0.834238095238
Number of Components: 81  ; GB Accuracy Score: 0.875761904762  ; DT Accuracy Score: 0.836047619048
Number of Components: 91  ; GB Accuracy Score: 0.874761904762  ; DT Accuracy Score: 0.833761904762
Number of Components: 101  ; GB Accuracy Score: 0.869666666667  ; DT Accuracy Score: 0.832666666667
Number of Components: 111  ; GB Accuracy Score: 0.867428571429  ; DT Accuracy Score: 0.829047619048
Number of Components: 121  ; GB Accuracy Score: 0.863857142857  ; DT Accuracy Score: 0.830714285714
Number of Components: 131  ; GB Accuracy Score: 0.86080952381  ; DT Accuracy Score: 0.826380952381
Number of Components: 141  ; GB Accuracy Score: 0.857619047619  ; DT Accuracy Score: 0.82380952381
Number of Components: 151  ; GB Accuracy Score: 0.854523809524  ; DT Accuracy Score: 0.824095238095
Number of Components: 161  ; GB Accuracy Score: 0.850333333333  ; DT Accuracy Score: 0.825333333333
Number of Components: 171  ; GB Accuracy Score: 0.847380952381  ; DT Accuracy Score: 0.823619047619
Number of Components: 181  ; GB Accuracy Score: 0.845857142857  ; DT Accuracy Score: 0.822714285714
Number of Components: 191  ; GB Accuracy Score: 0.843761904762  ; DT Accuracy Score: 0.821761904762
Number of Components: 201  ; GB Accuracy Score: 0.841380952381  ; DT Accuracy Score: 0.820523809524
```

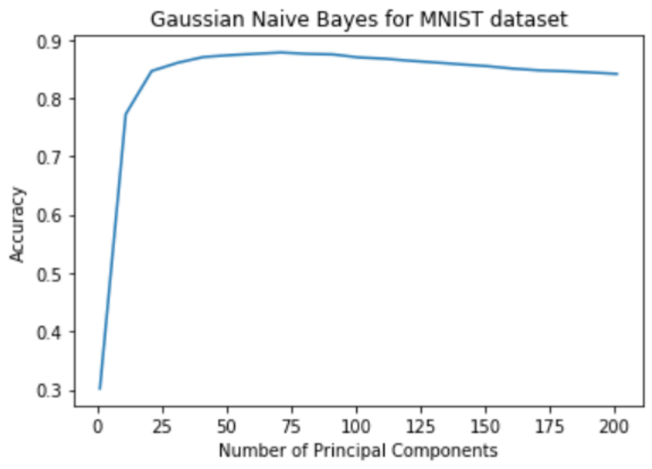
LFW:

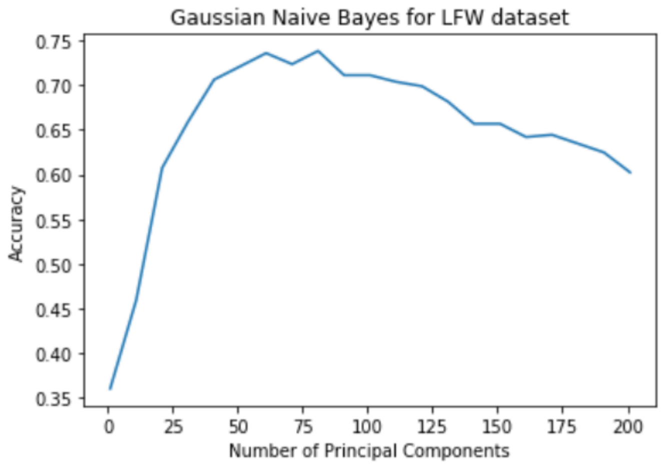


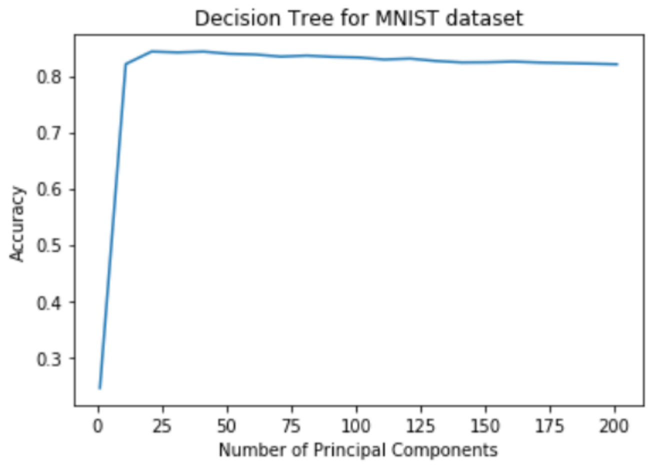
```python
for n in n_components_lfw:

    pca = PCA(n_components=n).fit(X_train_lfw)
    X_train_lfw_reduced = pca.transform(X_train_lfw)
    X_test_lfw_reduced = pca.transform(X_test_lfw)

    clf1.fit(X_train_lfw_reduced, y_train_lfw)
    clf2.fit(X_train_lfw_reduced, y_train_lfw)

    y_pred_lfw_gnb = clf1.predict(X_test_lfw_reduced)
    y_pred_lfw_dt = clf2.predict(X_test_lfw_reduced)

    accuracy_lfw_gnb = accuracy_score(y_pred_lfw_gnb, y_test_lfw)
    accuracy_lfw_dt = accuracy_score(y_pred_lfw_dt, y_test_lfw)

    print("Number of Components:", n, " ; GB Accuracy Score:",accuracy_lfw_gnb," ; DT Accuracy Score:",accuracy_lfw_dt)

    n_lfw.append(n)
    lfw_gnb.append(accuracy_lfw_gnb)
    lfw_dt.append(accuracy_lfw_dt)
```

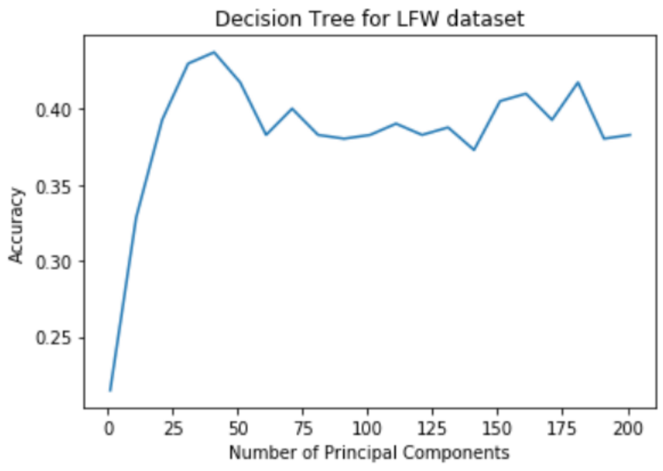
```
Number of Components: 1  ; GB Accuracy Score: 0.36049382716  ; DT Accuracy Score: 0.214814814815
Number of Components: 11  ; GB Accuracy Score: 0.459259259259  ; DT Accuracy Score: 0.328395061728
Number of Components: 21  ; GB Accuracy Score: 0.607407407407  ; DT Accuracy Score: 0.392592592593
Number of Components: 31  ; GB Accuracy Score: 0.659259259259  ; DT Accuracy Score: 0.42962962963
Number of Components: 41  ; GB Accuracy Score: 0.706172839506  ; DT Accuracy Score: 0.437037037037
Number of Components: 51  ; GB Accuracy Score: 0.720987654321  ; DT Accuracy Score: 0.417283950617
Number of Components: 61  ; GB Accuracy Score: 0.735802469136  ; DT Accuracy Score: 0.382716049383
Number of Components: 71  ; GB Accuracy Score: 0.723456790123  ; DT Accuracy Score: 0.4
Number of Components: 81  ; GB Accuracy Score: 0.738271604938  ; DT Accuracy Score: 0.382716049383
Number of Components: 91  ; GB Accuracy Score: 0.711111111111  ; DT Accuracy Score: 0.38024691358
Number of Components: 101  ; GB Accuracy Score: 0.711111111111  ; DT Accuracy Score: 0.382716049383
Number of Components: 111  ; GB Accuracy Score: 0.703703703704  ; DT Accuracy Score: 0.39012345679
Number of Components: 121  ; GB Accuracy Score: 0.698765432099  ; DT Accuracy Score: 0.382716049383
Number of Components: 131  ; GB Accuracy Score: 0.681481481481  ; DT Accuracy Score: 0.387654320988
Number of Components: 141  ; GB Accuracy Score: 0.656790123457  ; DT Accuracy Score: 0.372839506173
Number of Components: 151  ; GB Accuracy Score: 0.656790123457  ; DT Accuracy Score: 0.404938271605
Number of Components: 161  ; GB Accuracy Score: 0.641975308642  ; DT Accuracy Score: 0.40987654321
Number of Components: 171  ; GB Accuracy Score: 0.644444444444  ; DT Accuracy Score: 0.392592592593
Number of Components: 181  ; GB Accuracy Score: 0.634567901235  ; DT Accuracy Score: 0.417283950617
Number of Components: 191  ; GB Accuracy Score: 0.624691358025  ; DT Accuracy Score: 0.38024691358
Number of Components: 201  ; GB Accuracy Score: 0.602469135802  ; DT Accuracy Score: 0.382716049383
```

c) Predictive Performance









The predictive performance for the MNIST dataset (around 70%-85%) was better for both classifiers as compared to the LFW dataset (around 35%-70%). This makes sense to me as facial recognition should be a harder problem to predict than number recognition. The LFW models were also trained with a smaller training set (due to computing resource issues) which perhaps led to lesser accuracy as well.

When comparing the predictive performance of the two classifiers, Gaussian Naïve Bayes models generally performed better than the Decision Trees for both datasets. They had a higher maximum accuracy in both cases. It must be noted that the Decision Tree for the LFW dataset did especially poorly – not even predicting with a 50% accuracy on any of the number of principal components.

d) Influence of PCA on classification performance

The trend of the accuracy among the four models when applying PCA with different dimension sizes is very similar. All four models have a sharp increase in classification performance initially as the number of principal dimensions increases from 1 to 25. After which point, accuracy continues to improve but the rate of improvement tapers off. For the MNIST dataset, the highest accuracy is around 25 components whereas for the LFW dataset, it is at around n=50. After hitting the peak accuracy value, all four models begin to lose accuracy as the number of principal components continues to increase. In the MNIST dataset, the rate of fall in accuracy is relatively slow, but in the LFW dataset, the classification performance falls significantly faster.

These results clearly indicate that reducing dimensionality (in this case through the unsupervised algorithm of PCA), is very important in obtaining accurate predictions. Intuitively it may seem that the more the features the better the model would be but as it can be seen, this is not always the case. In all four models, the accuracy fell after hitting a peak. This is most likely because of overfitting – the models begin learning the noise in the training data – which offer no assistance when predicting classes for new data. That being said, it is also fairly obvious that we do not want to reduce the dimensions too much such that we are not able to learn enough about the underlying structure – this is why the accuracy is sharply increases initially.

There must be an optimum number of components to reduce each problem and the challenge lies in finding it. In our case, the MNIST dataset was most accurately modelled when around 25 principal dimensions were used and the LFW dataset when approximately 50 were used. Since both classifiers, the Naïve Bayes and Decision Tree, had the highest accuracy at similar number of components for both the datasets it can be inferred that the optimum number of components is largely based on the problem and the dataset as opposed to the model being used to estimate. This makes sense to me – the number of predictors that defined a given class must be fixed in the real-world and therefore should not depend too much on which model we pick to describe that relationship. Again, the challenge for designing a machine learning system is figuring out the best number of components to use for that problem and an analysis similar to the one performed here is a good strategy to assist with that.

2. Ensemble Learning

a) Load two classification datasets

Load datasets:

```
from sklearn.datasets import load_iris
from sklearn.datasets import load_wine

iris = load_iris()
wine = load_wine()
```

```
print(iris.data.shape)
print(iris.target.shape)

print(wine.data.shape)
print(wine.target.shape)
```

```
(150, 4)
(150,)
(178, 13)
(178,)
```

```
print(iris.keys())
print(wine.keys())
```

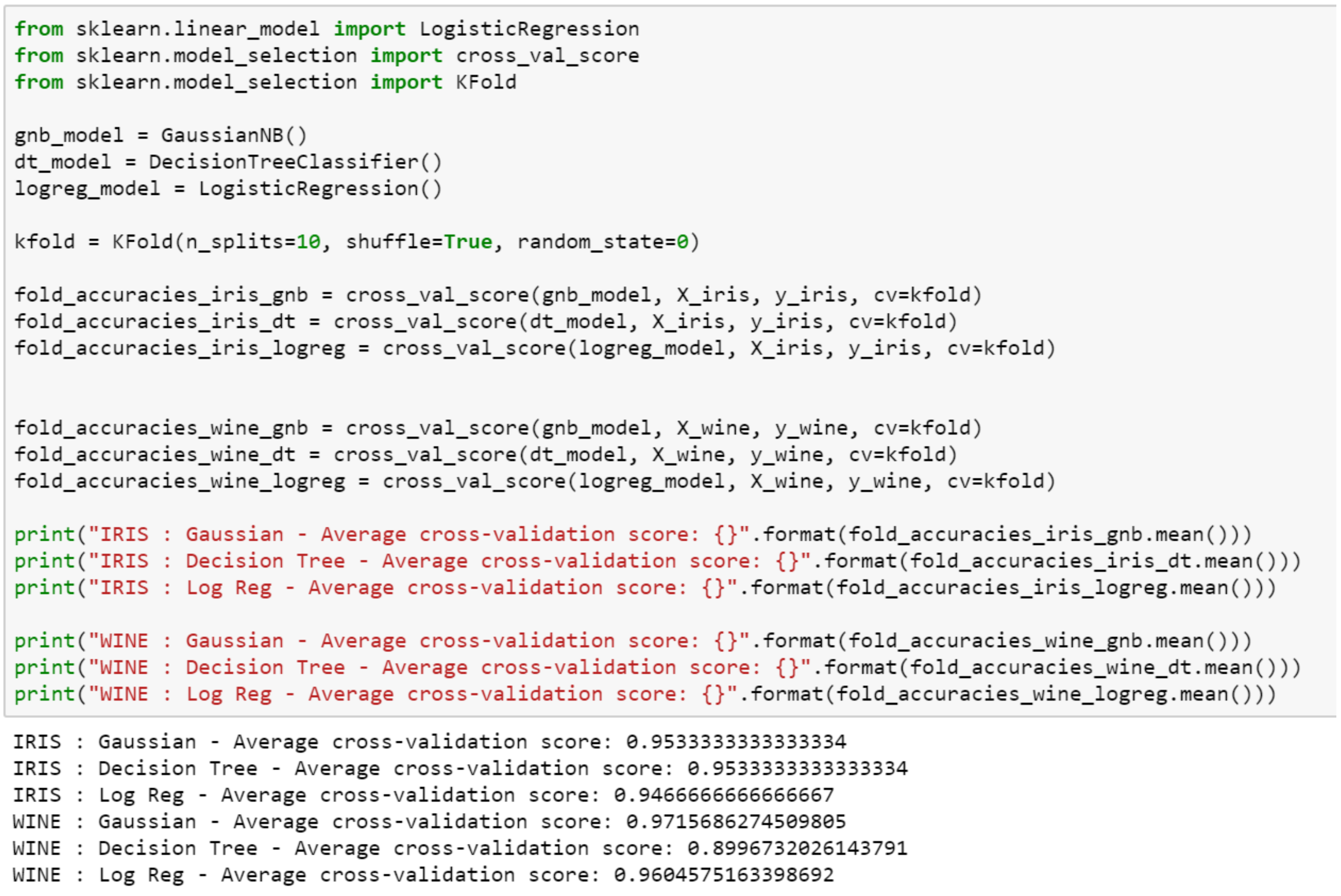
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
X_iris = iris.data
y_iris = iris.target

X_wine = wine.data
y_wine = wine.target
```

b) Evaluate the following using 10-fold cross validation

i) Three different Classifiers

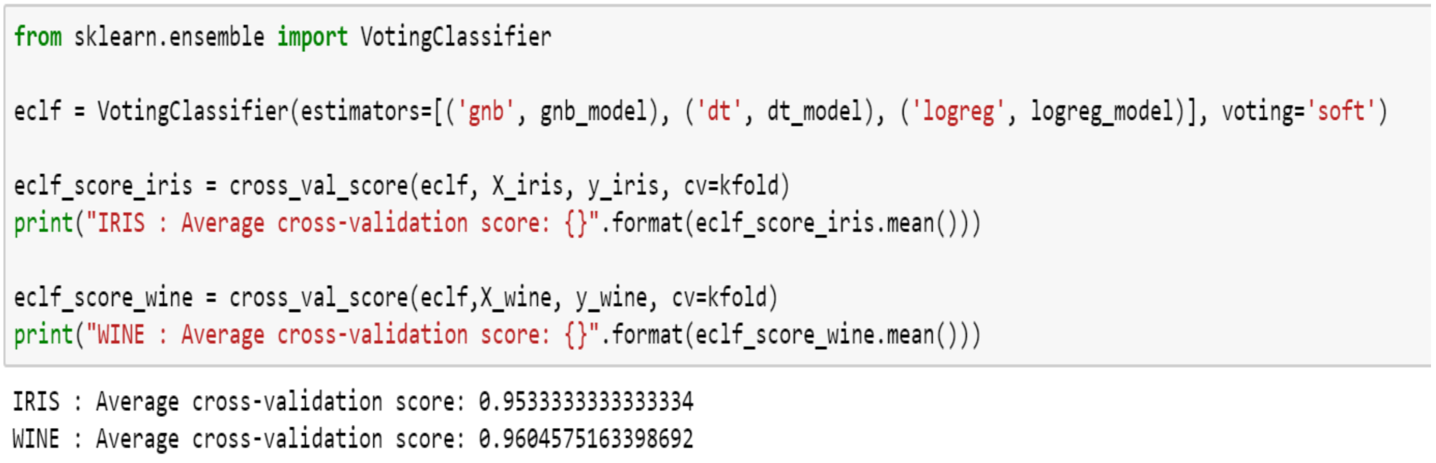
Chosen Classifiers: Naïve Bayes, Decision Tree, Logistic Regression



```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

gnb_model = GaussianNB()
dt_model = DecisionTreeClassifier()
logreg_model = LogisticRegression()

kfold = KFold(n_splits=10, shuffle=True, random_state=0)

fold_accuracies_iris_gnb = cross_val_score(gnb_model, X_iris, y_iris, cv=kfold)
fold_accuracies_iris_dt = cross_val_score(dt_model, X_iris, y_iris, cv=kfold)
fold_accuracies_iris_logreg = cross_val_score(logreg_model, X_iris, y_iris, cv=kfold)


fold_accuracies_wine_gnb = cross_val_score(gnb_model, X_wine, y_wine, cv=kfold)
fold_accuracies_wine_dt = cross_val_score(dt_model, X_wine, y_wine, cv=kfold)
fold_accuracies_wine_logreg = cross_val_score(logreg_model, X_wine, y_wine, cv=kfold)

print("IRIS : Gaussian - Average cross-validation score: {}".format(fold_accuracies_iris_gnb.mean()))
print("IRIS : Decision Tree - Average cross-validation score: {}".format(fold_accuracies_iris_dt.mean()))
print("IRIS : Log Reg - Average cross-validation score: {}".format(fold_accuracies_iris_logreg.mean()))

print("WINE : Gaussian - Average cross-validation score: {}".format(fold_accuracies_wine_gnb.mean()))
print("WINE : Decision Tree - Average cross-validation score: {}".format(fold_accuracies_wine_dt.mean()))
print("WINE : Log Reg - Average cross-validation score: {}".format(fold_accuracies_wine_logreg.mean()))
```

```
IRIS : Gaussian - Average cross-validation score: 0.9533333333333334
IRIS : Decision Tree - Average cross-validation score: 0.9533333333333334
IRIS : Log Reg - Average cross-validation score: 0.9466666666666667
WINE : Gaussian - Average cross-validation score: 0.9715686274509805
WINE : Decision Tree - Average cross-validation score: 0.8996732026143791
WINE : Log Reg - Average cross-validation score: 0.9604575163398692
```

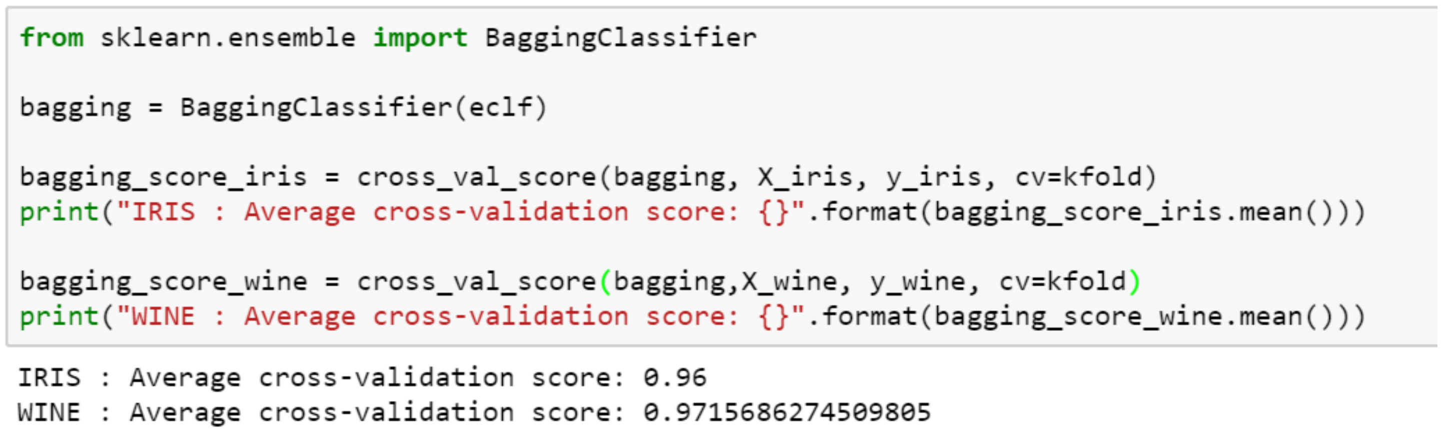
ii) Majority vote classifier



```
from sklearn.ensemble import VotingClassifier

eclf = VotingClassifier(estimators=[('gnb', gnb_model), ('dt', dt_model), ('logreg', logreg_model)], voting='soft')

eclf_score_iris = cross_val_score(eclf, X_iris, y_iris, cv=kfold)
print("IRIS : Average cross-validation score: {}".format(eclf_score_iris.mean()))

eclf_score_wine = cross_val_score(eclf,X_wine, y_wine, cv=kfold)
print("WINE : Average cross-validation score: {}".format(eclf_score_wine.mean()))
```

```
IRIS : Average cross-validation score: 0.9533333333333334
WINE : Average cross-validation score: 0.9604575163398692
```

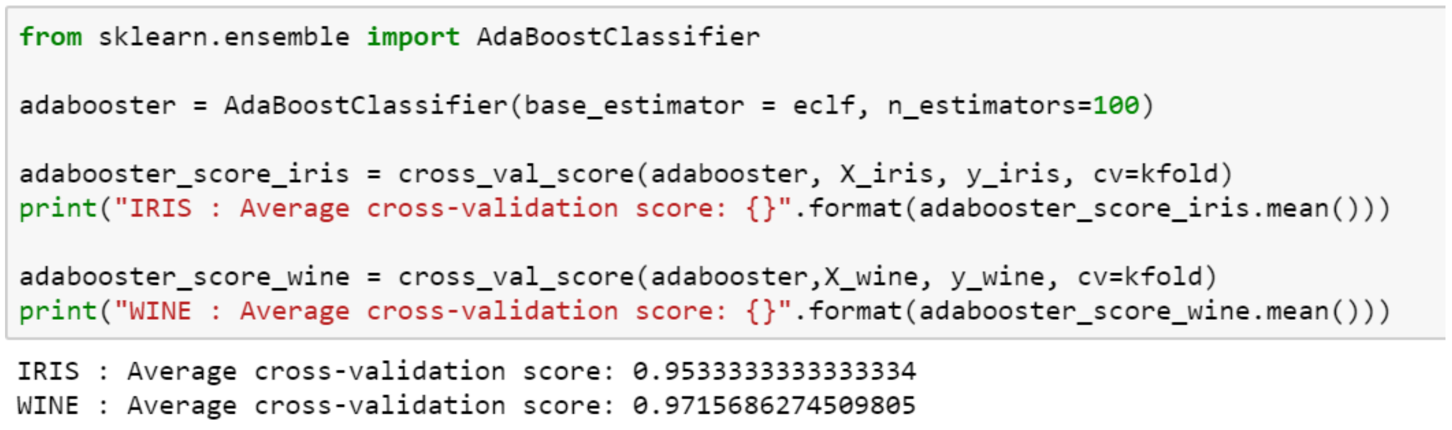
iii) Bagging Method



```python
from sklearn.ensemble import BaggingClassifier

bagging = BaggingClassifier(eclf)

bagging_score_iris = cross_val_score(bagging, X_iris, y_iris, cv=kfold)
print("IRIS : Average cross-validation score: {}".format(bagging_score_iris.mean()))

bagging_score_wine = cross_val_score(bagging,X_wine, y_wine, cv=kfold)
print("WINE : Average cross-validation score: {}".format(bagging_score_wine.mean()))
```

```
IRIS : Average cross-validation score: 0.96
WINE : Average cross-validation score: 0.9715686274509805
```

iv) Boosting Method



```python
from sklearn.ensemble import AdaBoostClassifier

adabooster = AdaBoostClassifier(base_estimator = eclf, n_estimators=100)

adabooster_score_iris = cross_val_score(adabooster, X_iris, y_iris, cv=kfold)
print("IRIS : Average cross-validation score: {}".format(adabooster_score_iris.mean()))

adabooster_score_wine = cross_val_score(adabooster,X_wine, y_wine, cv=kfold)
print("WINE : Average cross-validation score: {}".format(adabooster_score_wine.mean()))
```

```
IRIS : Average cross-validation score: 0.9533333333333334
WINE : Average cross-validation score: 0.9715686274509805
```

c) Mean Accuracy of each classifier for each dataset

IRIS dataset:

Classifier	Naïve Bayes	Decision Tree	Log Reg	Majority Vote	Bagging	Boosting
Mean Accuracy	95.34%	95.34%	94.67%	95.34%	96%	95.34%

WINE dataset:

Classifier	Naïve Bayes	Decision Tree	Log Reg	Majority Vote	Bagging	Boosting
Mean Accuracy	97.16%	89.97%	94.67%	96.05%	97.16%	97.16%

For IRIS, Bagging of the ensemble provided the highest accuracy (96%) where as Bagging and Boosting performed the best for the WINE data (97.16%).

d) Performance of the different ensemble methods

On average it can be seen that the ensemble learning methods performed slightly better than the individual classifiers on both the data sets. This makes sense to me as the ensemble learning classifiers (in orange) draw from the individual learners and therefore should at least perform as well as the best base. When comparing the ensemble to the worst individual learning, Decision Tree on the Wine dataset, it can be clearly seen that an ensemble is able to mask the effects of a bad base classifier through voting and weights.

Since I have performed the ensemble methods of bagging and boosting on the eclf, which is the ensemble classifier, it is essentially using all three individual classifiers to consistently give more accurate classifications. I have decided to do this so that I am considering separate levels of ensembles where I am using majority voting as well as bagging/boosting. Just to try it out, my code also shows bagging/boosting on the individual classifier that performed the best, Naïve Bayes. It was interesting to see the results – Iris dataset did extremely well but the wine performed good, but not as good as the ensemble of ensembles.

This indicates to me that we have to be careful when building ensembles. If we choose a single classifier to boost/bag with then it may carry forward any systematic problems that it has. More importantly, I feel that ensembles are even more susceptible to overfitting as they may learn the noise which may lead to worse results on the test data. On the other hand, a good model together with bagging and boosting can amplify the results too because bias, and variance are minimized. As a modeler, I decided to use an ensemble to bag and boost because this would provide a diverse set of individual classifiers to the ensemble and more consistently give me better results. I believe it was also an important step to cross validate our models to reduce the possibility of overfitting which as mentioned can be fairly high in ensembles.