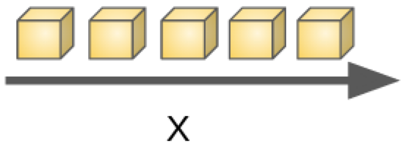
MACHINE LEARNING
MEI/1

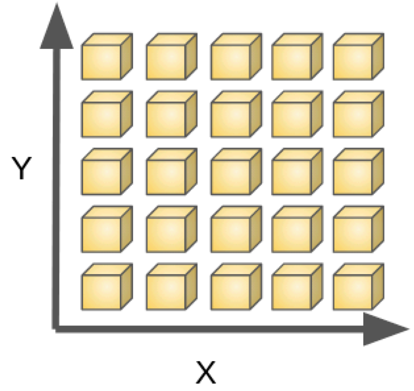
University of Beira Interior, Department of Informatics

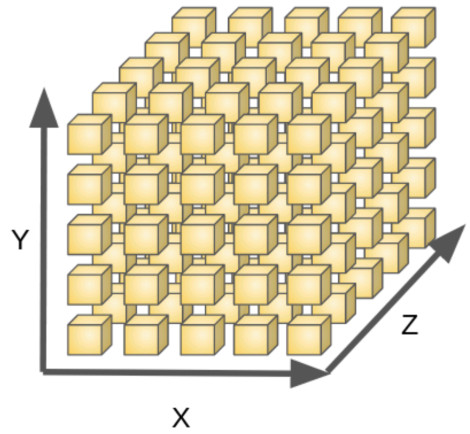
Hugo Pedro Proença, hugomcp@di.ubi.pt, 2019/2020

Dimensionality Reduction: PCA vs LDA

- Recall the **curse of dimensionality** and how it affects the performance of Machine Learning systems…
  - At one side, we want densely populated spaces (i.e., more examples should provide better results)…
  - Also, we want more features, as they correspond to bring more information to the problem, which should also be good…
  - More examples AND More Features → Imply much larger data sets, which can easily become an intractable problem
- More importantly, to keep a constant density value in high dimensional feature spaces, we need **much more** instances!
  - That is where dimensionality reduction techniques come into the story:
    - "Compact the data sets, keeping as much as possible the variance (**PCA**), or the discriminating power (**LDA**)"

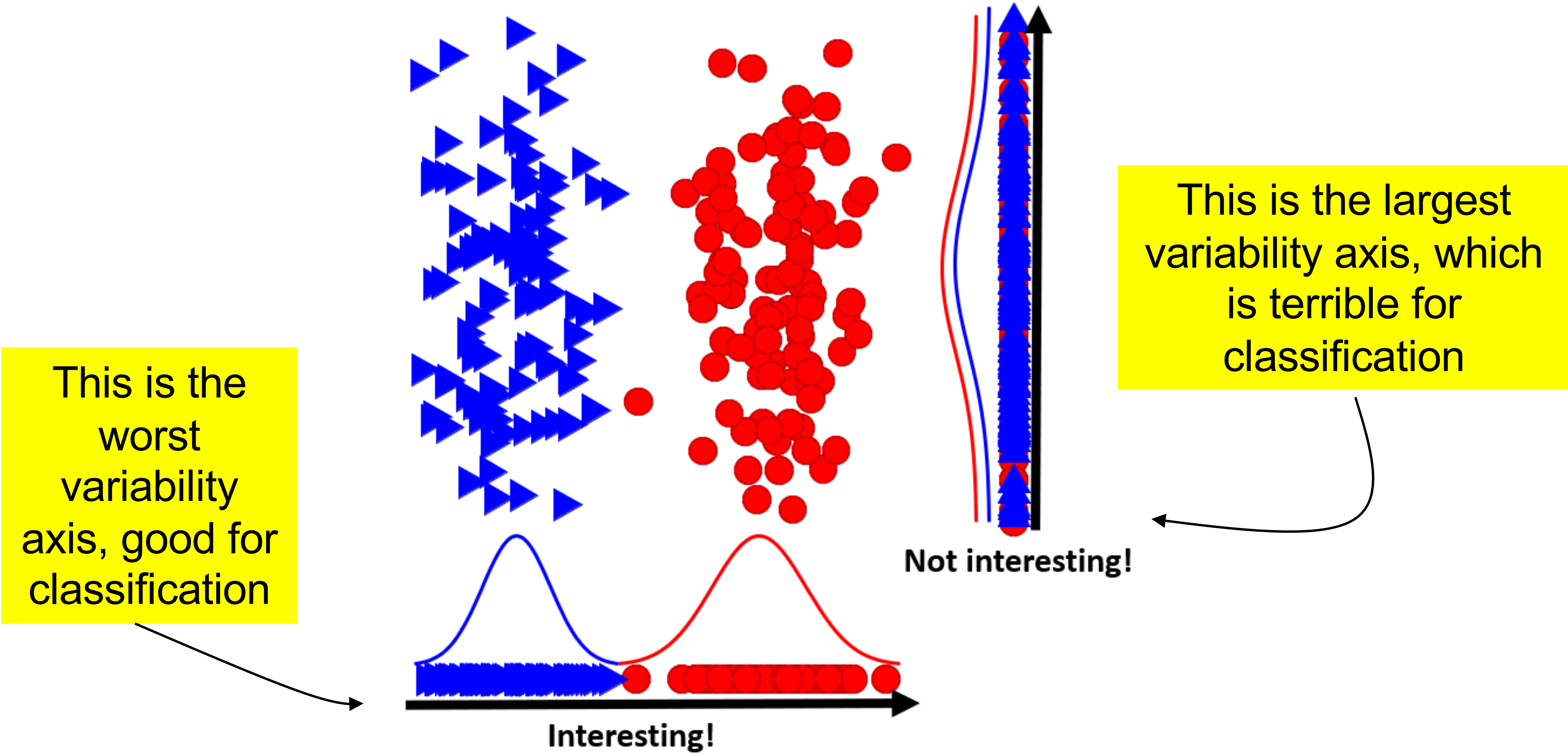






Dimensionality Reduction

- There are certain "PCA" projections that are not good for classification purposes
  - All cases where the axes corresponding to the largest variability are not those that enable to better discriminate between classes.



Machine Learning: “Digits” Example

- This week we will use the “Digits” dataset to illustrate the benefits of **PCA/LDA** projections.
- “**sklearn**” library provides an easy way to access the dataset:

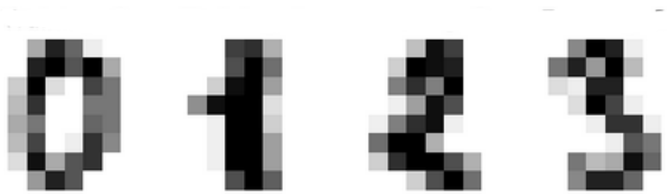
```
from sklearn.datasets import load_digits
digits = load_digits(return_X_y=True)
```

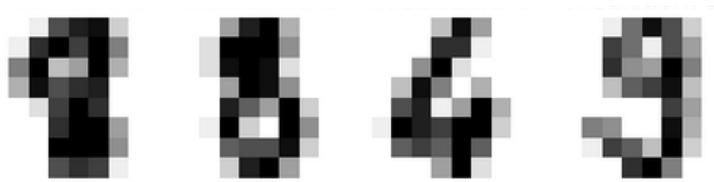
- There are a total of 1797 instances in the set, with 64 features:

```
print(digits.data.shape)
```

- Then, we can see any digit of the dataset (using its index “idx”):

```
import matplotlib.pyplot as plt
plt.gray()
plt.matshow(digits.images[idx])
plt.show()
```





Machine Learning: “Digits” Example

- We start by transforming the dataset into a (n_samples, n_features) matrix

```
digits = load_digits()
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target
```

- Next, we will divide the data into two disjoint sets: training + tests

```
# repeatability
seed = 3
np.random.seed(seed)
# learning/test split
train_indices = np.random.choice(len(X), round(len(X)*0.8), replace=False)
test_indices = np.array(list(set(range(len(X))) - set(train_indices)))
X_train = X[train_indices]
X_test = X[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]
```

Machine Learning: “Digits” Example

- Next, we can perform feature normalization

```
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

- Note that the .fit() method finds the minimum and maximum values per column (**only in the learning data!**), and then we convert all values into the unit interval, both the learning and test
    - By doing this, note that it is not assured that all elements in the test data are in the unit interval (why?)
- Next, we obtain the PCA and LDA representations:

```
pca = PCA(n_components=2)
X_train_pca = pca.fit(X_train).transform(X_train)
X_test_pca = pca.transform(X_test)

lda = LinearDiscriminantAnalysis(n_components=2)
X_train_lda = lda.fit(X_train, y_train).transform(X_train)
X_test_lda = lda.transform(X_test)
```

Machine Learning: “Digits” Example

- Finally, we create a simple Linear classifier to perceive how “good” the PCA and LDA projections are doing their jobs, for classification purposes:

```
clf = LinearDiscriminantAnalysis()
clf.fit(X_train, y_train)
y_out = clf.predict(X_test)
result = accuracy_score(y_test, y_out)    #accuracy on the original set

clf.fit(X_train_pca, y_train)
y_out_pca = clf.predict(X_test_pca)
result_pca = accuracy_score(y_test, y_out_pca)   #accuracy on PCA representation


clf.fit(X_train_lda, y_train)
y_out_lda = clf.predict(X_test_lda)
result_lda = accuracy_score(y_test, y_out_lda)    #accuracy on LDA representation
```

Machine Learning: “Digits” Exercise

- This week, you should run the “scr_PCA_vs_LDA.py” script, available at the course web site, and:
  - Compare the effectiveness of the original representation to PCA and LDA representations, depending of the number of componentes used in their projections;
    - Using a simple classifier, such as the LinearDiscriminant()
    - Using a a more sophisticated classifier, such as a SVM

```
from sklearn import svm
clf = svm.SVC()        #for multi-class classification
clf.fit(X, y)          #train the classifier
clf.predict(…)         #Predict the responses for test instances
```