Data Streaming Algorithms and Online Learning

Final Project - Dimensionality Reduction

By Millis Sahar

General

As mentioned in class, most importantly, the ideal project should be interesting.
Generally, it should focus on some “small” research topic (see suggested topics below, or your own topic), and include the following parts:

1. Your summary of what’s known about the topic, based on relevant literature. You should follow strict academic writing guidelines, including proper citation and clarity.
2. Experimental part, which consists of implementation and validation on data (either real or simulated). This part can serve as an empirical confirmation (or refutation) of a known theoretical result or conjecture.

Projects do not must to have both parts, but they should require (and will be evaluated accordingly) a proper amount of effort.

For example:

- if you write a comprehensive survey of a topic that may become the ”standard text” that people read for that topic, it’s great.
- If you make interesting progress in an open question, even “small one”, it’s great.
- If you write a short summary but implement a huge, large scale project withinteresting empirical findings, it’s great.

You don’t have to decide in advance whether you’re going to spend more time on the theoretical or on the experimental side.
I suggest you start reading about a topic you find interesting, think about it, and see where it takes you.

Sublinear Algorithms

Sublinear algorithms include any algorithm that runs in time o(n) for input of size n.
Often, you don't even see the entire input. There is a nice list of open problems in sublinear algorithms (and some of these problems involve streaming and dimensionality reduction):
https://sublinear.info/index.php?title=Open_Problems:By_Number (https://sublinear.info/index.php?title=Open_Problems:By_Number) It's a great place to get ideas, together with relevant literature.

Clustering

Clustering means to partition all data points such that each partition consists of "similar" or "close" points, while all those between two clusters are "different" or "far".
In MachineLearning, clustering is considered the most important examples of unsupervised learning.
There are extensive previous work on clustering, for example [6,34] are some interesting results related k-means.
Example of literature on streaming algorithms for clustering data can be found in [1,4,16].

Nearest neighbor search

Nearest neighbor searching is aiming at preprocess a dataset so that, given a new data point, the closest datapoint can be found (with high probability). It is widely used in MachineLearning as a non-parametric learning algorithm (given a new instance point, assign to it the label of the closest instance you had in training set).
For theoretical papers, you can find some upper bounds in [23,33,39], and some lower bounds in [35, 36].
There is also much practical work in nearest neighbor searching, see [32].

Sparse Recovery

Sparse recovery is the idea that, if you have an (approximate) sparse prior on your data, then you need only few (random) measurements in order to faithfully represent it.
The theory of applications of sparse recovery (as known as "compressed sensing") is deep also from a practical point of view. [13] is the classis intro to this field, [44] discusses the connection with random projections.
Some more recent stuff are [3,24].

MapReduce

This large scale distributed computational architecture is widely used. See [21, 27, 30, 29,43].
Matrix Sampling and Sketching, Large Scale Numerical Linear Algebra How to "summarize" large matrices, and what operations can we estimate using this summarization? See [17,18,19,20,31,42].
Matrix Completion (The "Netflix" Problem) The idea of matrix completion is how to approximately complete a big matrix when only small part of it is known. This subject has much work in both the practical and theoretical extremes. In [9,10,28] Yehuda Koren's team explain how they won the Netflix challenge.
There is also vast literature in practical Machine-Learning [14, 15, 22, 40].

Random Projections

Random projections is about reducing dimensionality of data that is oblivious to the data.
For Euclidean metrics, a name that is almost synonymous with "random projections" is "JohnsonLindenstrauss" [25]. See also [2,5,26].
Random projections have been also studied as preprocessing steps for other MachineLearning algorithms that rely on input dimensionality (SVM, linear regression, SVD, etc). See [7,37,38].

Conferences:

You can also search the prestigious Big-data conferences:

- Theoretical focus:
    - FOCS. Foundations of Computer Science.
    - ICALP (track A). International Colloquium on Automata, Languages and Programming.
    - PODS. Symposium on Principles of Database Systems.
    - RANDOM. International Workshop on Randomization and Computation.
    - SODA. Symposium on Discrete Algorithms.
    - STOC. Symposium on the Theory of Computing.

- Experimental focus:
    - KDD. Knowledge Discovery and Data Mining. (data mining)
    - ICML. International Conference on Machine Learning. (machine learning)
    - NIPS. Neural Information Processing Systems. (machine learning)
    - SIGMOD. ACM SIGMOD International Conference on Management of Data. (databases)
    - VLDB. Very Large Data Bases. (databases)

References

[1] Streaming-data algorithms for high-quality clustering. In Proceedings of the 18th International Conference on Data Engineering, ICDE ’02, pages 685–, 2002.

[2] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnsonlindenstrauss transform. In Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pages 557–563. ACM, 2006.

[3] Nir Ailon, Yudong Chen, and Xu Huan. Breaking the small cluster barrier of graph clustering. arXiv preprint arXiv:1302.4549, 2013.

[4] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. In Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada., pages 10–18, 2009.

[5] Nir Ailon and Edo Liberty. An almost optimal unrestricted fast johnson-lindenstrauss transform. ACM Transactions on Algorithms (TALG), 9(3):21, 2013.

[6] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[7] Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: Supercharging lapack’s least-squares solver. SIAM Journal on Scientific Computing, 32(3):1217–1236, 2010.

[8] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. Machine Learning, 56:89–113, 2004.

[9] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. ACM SIGKDD Explorations Newsletter, 9(2):75–79, 2007.

[10] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize, 2007.

[11] Francesco Bonchi, David Garc´ıa-Soriano, and Edo Liberty. Correlation clustering: from theory to practice. In The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, New York, NY, USA - August 24 - 27, 2014, page 1972, 2014.

[12] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near optimal columnbased matrix reconstruction. In FOCS, pages 305–314, 2011. 6

[13] E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. IEEE Trans. Inf. Theor., 52(2):489–509, 2006.

[14] Emmanuel J Candes and Benjamin Recht. Exact matrix completion via convex optimization. Foundations of Computational mathematics, 9(6):717–772, 2009.

[15] Emmanuel J Candes and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. Information Theory, IEEE Transactions on, 56(5):2053–2080, 2010.

[16] Moses Charikar, Liadan O’Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA, pages 30–39, 2003.

[17] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In APPROX-RANDOM, pages 292–303, 2006.

[18] Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. Fast approximation of matrix coherence and statistical leverage. 2011.

[19] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Sampling algorithms for l2 regression and applications. In SODA, 2006.

[20] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. J. ACM, 51(6):1025–1041, November 2004.

[21] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Algorithms and Computation, pages 374–383. Springer, 2011.

[22] D. Gross. Recovering low-rank matrices from few coefficients in any basis. IEEE Trans. Inf. Theor., 57(3):1548–1566, March 2011.

[23] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory of computing, 8(1):321–350, 2012.

[24] Jarvis Haupt, Rui M Castro, and Robert Nowak. Distilled sensing: Adaptive sampling for sparse detection and estimation. Information Theory, IEEE Transactions on, 57(9):6222– 6235, 2011.

[25] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. Contemporary Mathematics, 26:189–206, 1984.

[26] Daniel M Kane and Jelani Nelson. A derandomized sparse johnson-lindenstrauss transform. arXiv preprint arXiv:1006.3585, 2010. 7

[27] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

[29] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In Proceedings of the 25th ACM symposium on Parallelism in algorithms and architectures, pages 1–10. ACM, 2013.

[30] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in map-reduce. In Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures, pages 85–94. ACM, 2011.

[31] Edo Liberty. Simple and deterministic matrix sketching. In The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013, pages 581–588, 2013.

[32] Ting Liu, Charles Rosenberg, and Henry A Rowley. Clustering billions of images with large scale nearest neighbor search. In Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on, pages 28–28. IEEE, 2007.

[33] Huy L Nguyen. Approximate nearest neighbor search. arXiv preprint arXiv:1306.3601, 2013.

[34] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. Journal of the ACM (JACM), 59(6):28, 2012.

[35] Ryan ODonnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). ACM Transactions on Computation Theory (TOCT), 6(1):5, 2014.

[36] Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on, pages 805–814. IEEE, 2010.

[37] Saurabh Paul, Christos Boutsidis, Malik Magdon-Ismail, and Petros Drineas. Random projections for linear support vector machines. TKDD, 8(4):22, 2014.

[38] Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In In Neural Infomration Processing Systems, 2007. 8

[39] Ilya Razenshteyn. Beyond Locality-Sensitive Hashing. PhD thesis, Massachusetts Institute of Technology, 2014.

[40] Benjamin Recht. A simpler approach to matrix completion. The Journal of Machine Learning Research, 12:3413–3430, 2011.

[41] Benjamin Recht and Christopher Re. Parallel stochastic gradient algorithms for largescale matrix completion. Mathematical Programming Computation, 5(2):201–226, 2013.

[42] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In FOCS, pages 143–152, 2006.

[43] Leslie G Valiant. A bridging model for multi-core computing. In Algorithms-ESA 2008, pages 13–28. Springer, 2008.

[44] Rachel Ward and Felix Krahmer. New and improved Johnson-Lindenstrauss embeddings via the restricted isometry property. SIAM Jorunal on Mathematical Analysis, 43:1269– 1281, 2011.

Project - Dimensionality Reduction

In accordance with the guidelines, I started the project when I wanted to focus on the dimensionality reduction concept.
This was while I had a dataset with hundreds of dimensions with a goal of displaying different forms of dimensionality reduction on the existing data, while relying on implementations by Scikit Learn.
But after my professor's everlasting repetition of the word "boring", I chose to apply dimensional reduction methods to 3D images.

While looking for articles and references on the subject, I came across this (https://ieeexplore.ieee.org/document/6780074) article:
Dimensionality reduction for fast and accurate video search and retrieval in a large scale database,2013 by Utkarsha S Pacharaney ; Pritam S. Salankar ; Saradadevi Mandalapu

The above article gave me a new direction - `reducing dimensions by nonlinear methoeds!`
An area that I did not deal with in my current work as a data scientist. Also, it does not indicate that we will study the field as part of our master's degree program.

Therefore, in the current project, I saw a great opportunity to explore, study and experiment in this field - **Dimension Reduction by Nonlinear Methods.**

All Imports

In [2]:

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.linear_model import LogisticRegression as LR
from sklearn.decomposition import  PCA
from sklearn.manifold import Isomap

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report as CR
from sklearn.metrics import accuracy_score as AS


# datasets
from sklearn.datasets import make_swiss_roll
from sklearn.datasets import make_moons
from sklearn.datasets import make_s_curve
from sklearn.datasets import load_iris
from sklearn.datasets import load_diabetes
from sklearn.datasets import load_boston
from sklearn.datasets import fetch_olivetti_faces
from sklearn.datasets import fetch_california_housing
from sklearn.datasets import fetch_lfw_people

# methods - Isomap
from sklearn.manifold import Isomap
from scipy.spatial.distance import pdist
from scipy.spatial.distance import squareform

# methods - Levina-Bickel
from sklearn.neighbors import NearestNeighbors

# methods - Granata-Carnevale
import sys,argparse
from scipy.optimize import curve_fit
from scipy.sparse import csr_matrix
from scipy.spatial import distance
from sklearn.neighbors import kneighbors_graph, radius_neighbors_graph
from sklearn.utils.graph import graph_shortest_path
```

Hide warnings

In [49]:

```
import warnings
warnings.filterwarnings('ignore')
```

Disabling autoscrolling for long output

In [50]:

```
%%javascript
require(
        ["notebook/js/outputarea"],
        function (oa) {
            oa.OutputArea.auto_scroll_threshold = -1;
            console.log("Setting auto_scroll_threshold to -1");
        });
```

What is Dimension reduction

Dimension reduction are **powerful techniques** useful for exploratory data analysis, among other things.
A typical machine learning challenge will include a large number of predictors, which makes visualization somewhat challenging.

The general idea is to `reduce the dimension` of the dataset while `preserving important characteristics` , such as the distance between features or observations. Also, an important advantage is when having fewer dimensions, `visualization becomes more feasible` .

A well writen explanation and a simple example can be found [here (https://rafalab.github.io/dsbook/large-datasets.html#dimension-reduction)](https://rafalab.github.io/dsbook/large-datasets.html#dimension-reduction).

First Step

I tried to get fmiliar with a couple of Nonlinear Methods of Dimension Reduction. 2 methos were standing out:

- Isomap method
- Levina-Bickel method

So my first instinct is to comare a nonlinear vs a linear method.

PCA vs Isomap

Principal component analysis (PCA) is a `statistical procedure` that uses an `orthogonal transformation` to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

Isomap is a `nonlinear dimensionality reduction method` . It is one of several widely used low-dimensional embedding methods. Isomap is used for computing a quasi-isometric, low-dimensional embedding of a set of high-dimensional data points. The algorithm provides a simple method for estimating the intrinsic geometry of a data manifold based on a rough estimate of each data point’s neighbors on the manifold. Isomap is highly efficient and generally applicable to a broad range of data sources and dimensionalities.

Reference of the implementation insperation:

- Python examples of Principal Component Analysis (https://gist.github.com/Mashimo/69f0972d51358d65f088a7147dfc5ff1)
- Python examples of isomap algorithm (https://gist.github.com/Mashimo/b8a8d4dc18bf6875c8547134b543898f)

Digits detection Comparing PCA and isomap

Applying both PCA and Isomap to the raw images to derive 2D principal components and a 2D embedding of the data's intrinsic geometric structure.

Whatever your high-dimensional samples are, be they images, sound files, or thoughtfully collected attributes, they can all be considered single points in a high dimensional feature-space. Each one of your observations is just a single point.
Even with a high dimensionality, it's possible that most or all your samples actually lie on a lower dimension surface.

Isomap aims to capture that embedding, which is essentially the motion in the underlying, non-linear degrees of freedom.
By testing isomap on a carefully constructed dataset, you will be able to visually confirm its effectiveness, and gain a deeper understanding of how and why each parameter acts the way it does.

Manifold extraction and Isomap(specifically) are really good with vision recognition problems, speech problems, and many other real-world tasks, such as identifying similar objects, or objects that have undergone some change.

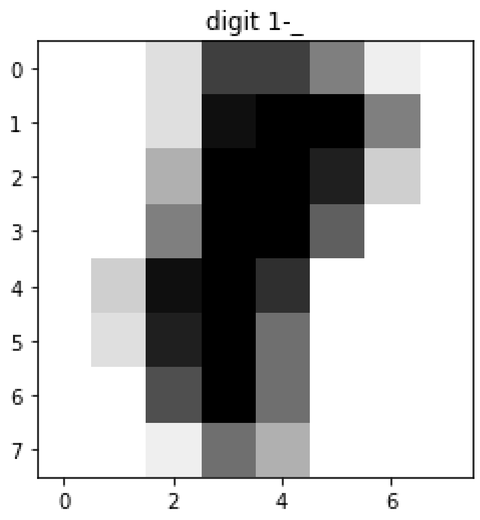
In [128]:

```python
digits = load_digits()
print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..1
6.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten
+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute o
f
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological Universit
y.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [79]:

```
def show_digit(image,target,prediction='_'):
    plt.imshow(image,cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('digit ' + str(target) + '-' +  str(prediction))
    plt.show()

temp = np.random.randint(len(digits.images))
show_digit(digits.images[temp], digits.target[temp])
```



In [159]:

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

Without dimention reduction (original data)

In [346]:

```
np.random.seed(42)

time0 = time.time()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# clf = RFC(max_depth=2)
clf = LR()
clf.fit(X_train,y_train)
time_original = time.time()-time0

pred_original = clf.predict(X_test)

cr_original = CR(y_true=y_test,y_pred=pred_original,output_dict=True)
df_cr_original = pd.DataFrame(cr_original).transpose()

accuracy_original = AS(y_true=y_test,y_pred=pred_original)
```

With linear dimention reduction

In [355]:

```
number_of_components = 10
```

In [356]:

```
np.random.seed(42)

time0 = time.time()
technique = PCA(n_components=number_of_components)
X_reduced = technique.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, shuffl
e=False)

# clf_pca = RFC(max_depth=2)
clf = LR()
clf.fit(X_train,y_train)
time_linear = time.time()-time0

pred_linear = clf.predict(X_test)

cr = CR(y_true=y_test,y_pred=pred_linear,output_dict=True)
df_cr_linear = pd.DataFrame(cr).transpose()

accuracy_linear = AS(y_true=y_test,y_pred=pred_linear)
```

With linear dimention reduction

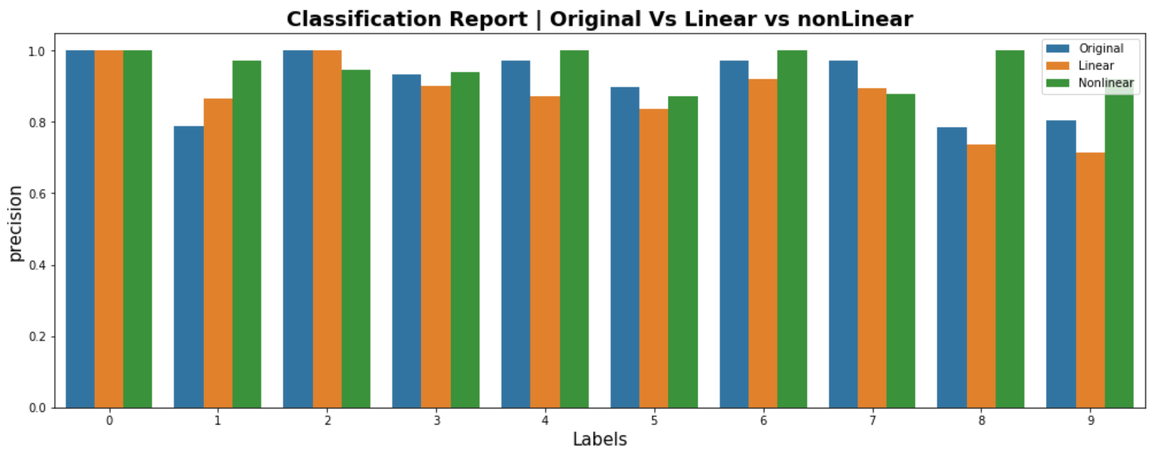
In [357]:

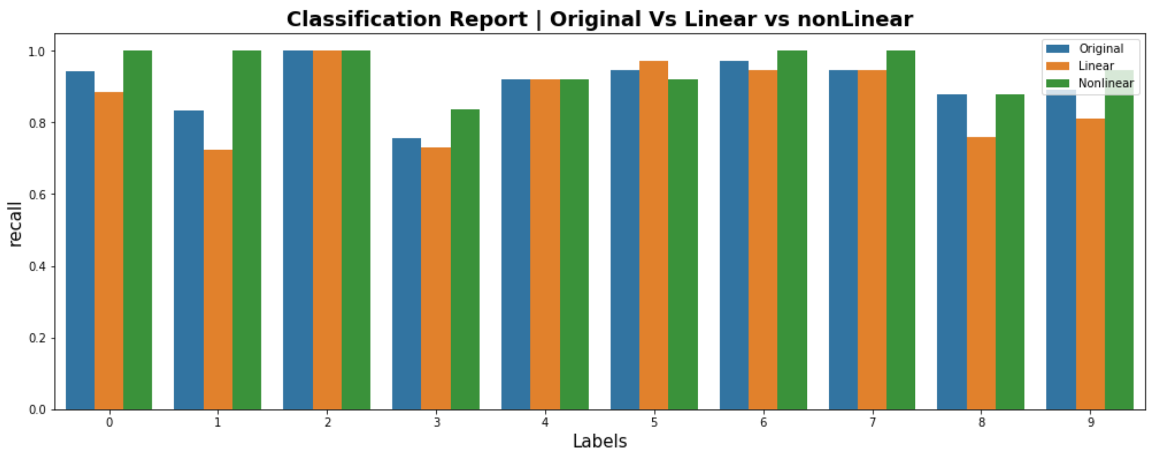
```
np.random.seed(42)

time0 = time.time()
technique = Isomap(n_components=number_of_components)
X_reduced = technique.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, shuffl
e=False)

# clf_pca = RFC(max_depth=2)
clf = LR()
clf.fit(X_train,y_train)
time_nonlinear = time.time()-time0

pred_nonlinear = clf.predict(X_test)

cr = CR(y_true=y_test,y_pred=pred_nonlinear,output_dict=True)
df_cr_nonlinear = pd.DataFrame(cr).transpose()

accuracy_nonlinear = AS(y_true=y_test,y_pred=pred_nonlinear)
```

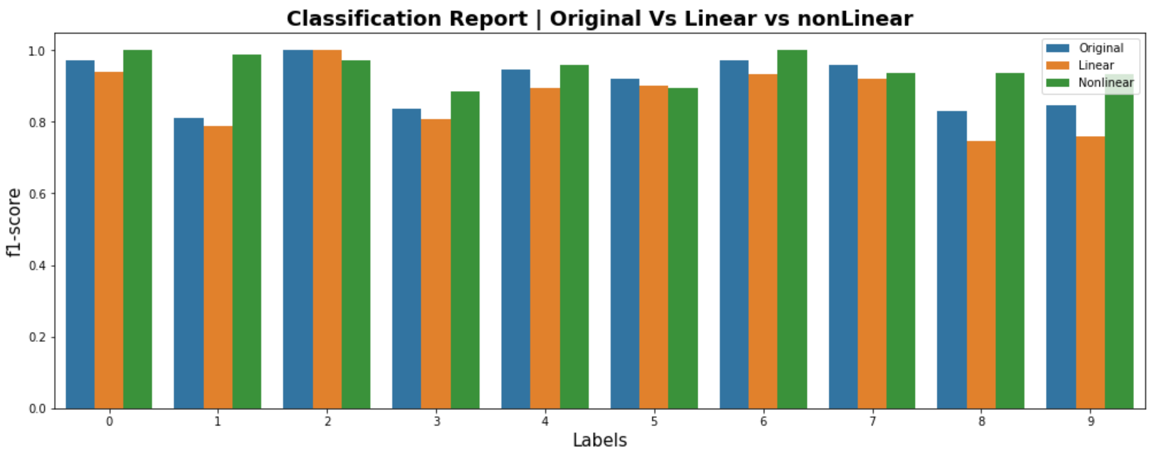
Visualize findings

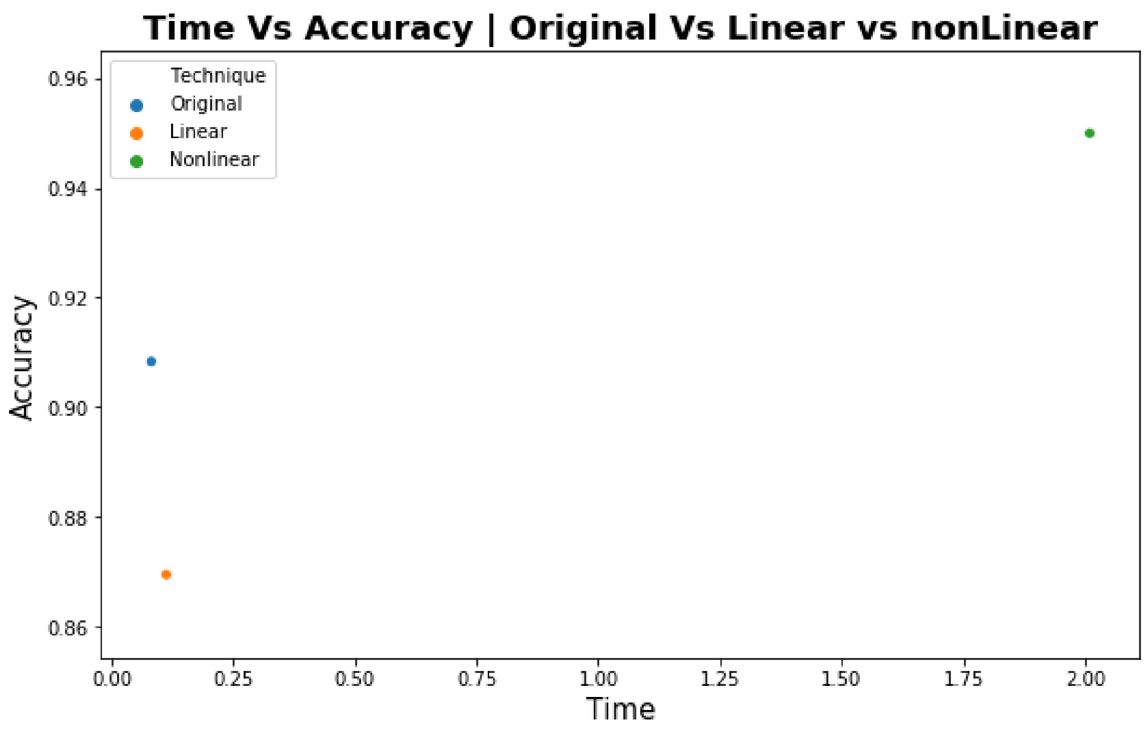
In [358]:

```
df_cr_original['Technique'] = 'Original'
df_cr_linear['Technique'] = 'Linear'
df_cr_nonlinear['Technique']= 'Nonlinear'
df_cr_labels = pd.concat([df_cr_original[0:10],df_cr_linear[0:10],df_cr_nonlinear[0:10
]])

for col in ['precision','recall','f1-score']:
    print('\n')
    plt.figure(figsize=(17,6))
    sns.barplot(x='index',y=col,hue='Technique',data=df_cr_labels.reset_index())
    plt.title('Classification Report | Original Vs Linear vs nonLinear',fontweight="bol
d", size=18)
    plt.ylabel(col,size=15)
    plt.xlabel('Labels',size=15)
    plt.legend()
    plt.show()
    print('\n')

print('\n')
plt.figure(figsize=(10,6))
sns.scatterplot(x='Time',y='Accuracy',hue='Technique',data=pd.DataFrame({'Technique': [
'Original','Linear','Nonlinear'], 'Time':[time_original,time_linear,time_nonlinear],'Ac
curacy':[accuracy_original,accuracy_linear,accuracy_nonlinear]}))
plt.title('Time Vs Accuracy | Original Vs Linear vs nonLinear',fontweight="bold", size=
18)
plt.ylabel('Accuracy',size=15)
plt.xlabel('Time',size=15)
plt.legend()
plt.show()
```


Classification Report | Original Vs Linear vs nonLinear
precision
Labels
Original
Linear
Nonlinear


Classification Report | Original Vs Linear vs nonLinear
recall
Labels
Original
Linear
Nonlinear


Classification Report | Original Vs Linear vs nonLinear
f1-score
Labels
Original
Linear
Nonlinear



I can identify a few trends, but before I assming stuff, let's check another thing.

Improvement

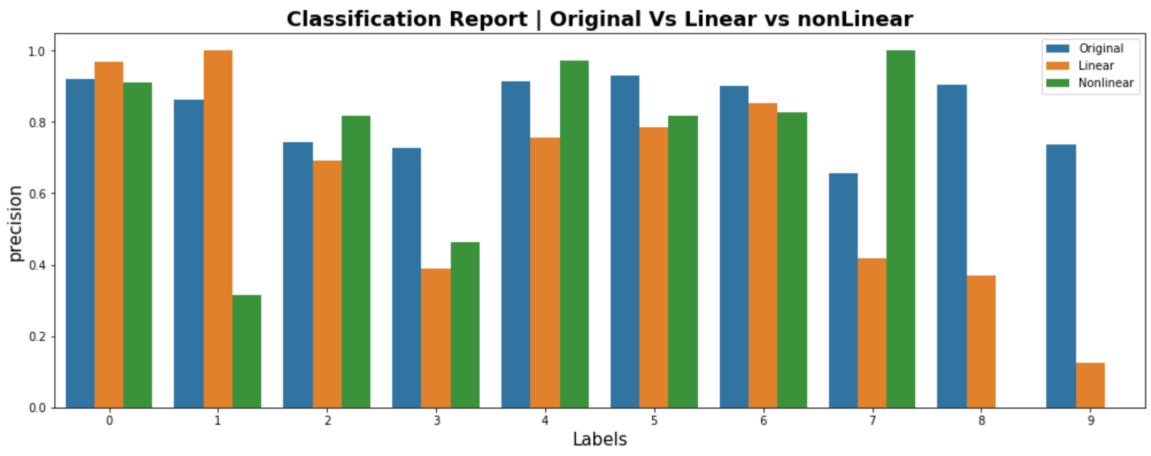
- Will make the model more complicated - I'll use an ensemble classifier.
- Will make reduce the dataset further - I'll decrease the number of compenents.

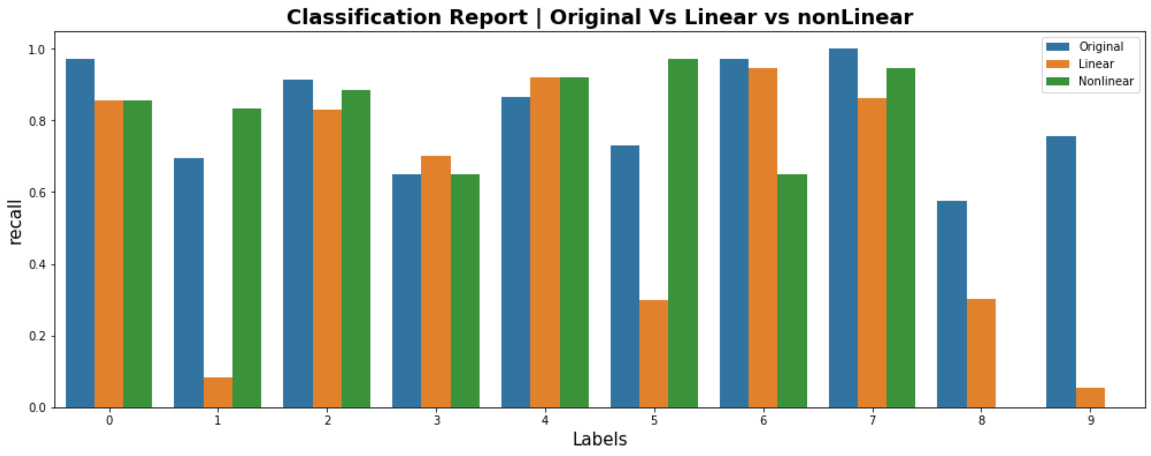
Obiusly there is an optimal number of components, but this is specific to the data, and I'm trying to understand trends.

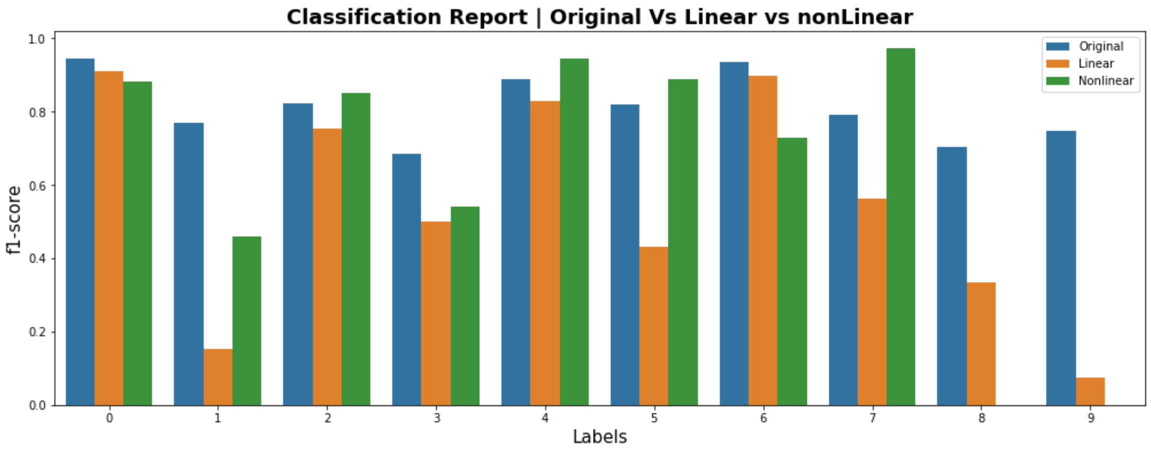
In [365]:

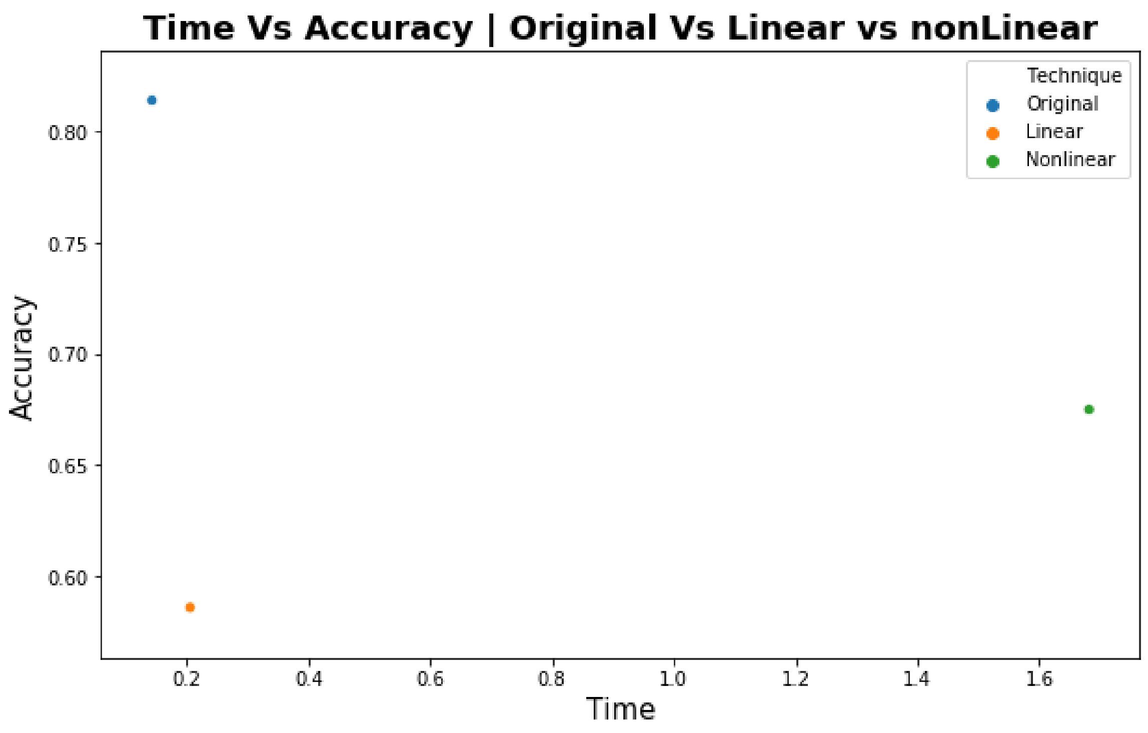
```
np.random.seed(42)
number_of_components = 4


# Original
time0 = time.time()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

clf = RFC(max_depth=2)
clf.fit(X_train,y_train)
time_original = time.time()-time0

pred_original = clf.predict(X_test)

cr_original = CR(y_true=y_test,y_pred=pred_original,output_dict=True)
df_cr_original = pd.DataFrame(cr_original).transpose()

accuracy_original = AS(y_true=y_test,y_pred=pred_original)


# Linear
time0 = time.time()
technique = PCA(n_components=number_of_components)
X_reduced = technique.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, shuffl
e=False)

clf_pca = RFC(max_depth=2)
clf.fit(X_train,y_train)
time_linear = time.time()-time0

pred_linear = clf.predict(X_test)

cr = CR(y_true=y_test,y_pred=pred_linear,output_dict=True)
df_cr_linear = pd.DataFrame(cr).transpose()

accuracy_linear = AS(y_true=y_test,y_pred=pred_linear)


# Nonlinear
time0 = time.time()
technique = Isomap(n_components=number_of_components)
X_reduced = technique.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, shuffl
e=False)

clf_pca = RFC(max_depth=2)
clf.fit(X_train,y_train)
time_nonlinear = time.time()-time0

pred_nonlinear = clf.predict(X_test)

cr = CR(y_true=y_test,y_pred=pred_nonlinear,output_dict=True)
df_cr_nonlinear = pd.DataFrame(cr).transpose()

accuracy_nonlinear = AS(y_true=y_test,y_pred=pred_nonlinear)
```

```
# Visualize
df_cr_original['Technique'] = 'Original'
df_cr_linear['Technique'] = 'Linear'
df_cr_nonlinear['Technique']= 'Nonlinear'
df_cr_labels = pd.concat([df_cr_original[0:10],df_cr_linear[0:10],df_cr_nonlinear[0:10
]])

for col in ['precision','recall','f1-score']:
    print('\n')
    plt.figure(figsize=(17,6))
    sns.barplot(x='index',y=col,hue='Technique',data=df_cr_labels.reset_index())
    plt.title('Classification Report | Original Vs Linear vs nonLinear',fontweight="bol
d", size=18)
    plt.ylabel(col,size=15)
    plt.xlabel('Labels',size=15)
    plt.legend()
    plt.show()
    print('\n')

print('\n')
plt.figure(figsize=(10,6))
sns.scatterplot(x='Time',y='Accuracy',hue='Technique',data=pd.DataFrame({'Technique': [
'Original','Linear','Nonlinear'], 'Time':[time_original,time_linear,time_nonlinear],'Ac
curacy':[accuracy_original,accuracy_linear,accuracy_nonlinear]}))
plt.title('Time Vs Accuracy | Original Vs Linear vs nonLinear',fontweight="bold", size=
18)
plt.ylabel('Accuracy',size=15)
plt.xlabel('Time',size=15)
plt.legend()
plt.show()
```


Classification Report | Original Vs Linear vs nonLinear
precision
Labels
Original
Linear
Nonlinear


Classification Report | Original Vs Linear vs nonLinear
recall
Labels
Original
Linear
Nonlinear


Classification Report | Original Vs Linear vs nonLinear
f1-score
Labels
Original
Linear
Nonlinear



Conclutions

- The Original dataset has a higher accuracy compare to the reduced dataset. But this is due to the nature of this specific data, as seen between the two examples.
- The Nonlinear take signifecntly more time to reduce, but always(almost) testing better.
- The reduced data take less space, it's "easier" to train and visualize.

What is actualy nonliner reduction does?

All this aside, I didn't quite understand what the nonlinearity implies in this context.

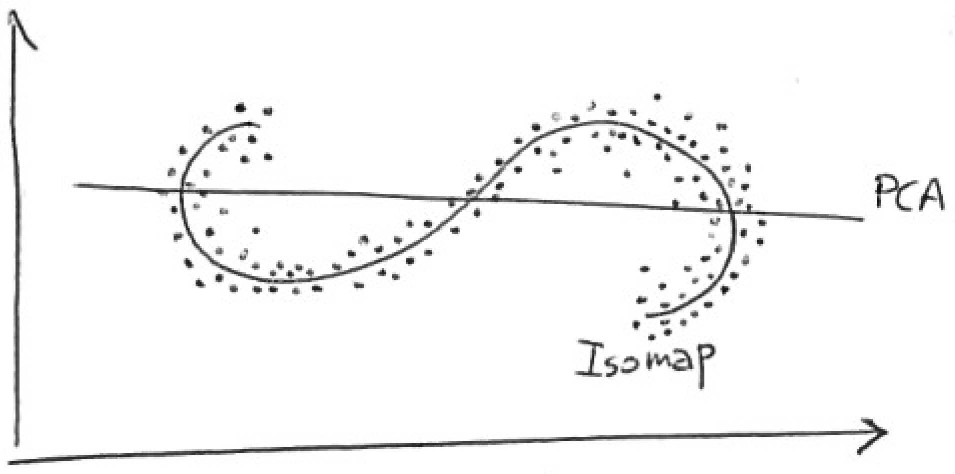
If Dimensionality Reduction means that you map each many-dimensional vector into a low-dimensional vector.
In other words, you represent (replace) each many-dimensional vector by a low-dimensional vector.

Therefore, Linear Dimensionality Reduction means that components of the low-dimensional vector are given by linear functions of the components of the corresponding high-dimensional vector.

For example in case of reduction to two dimensions we have:

```
 [x1, x2, ..., xn] ->  [f1(x1, x2, ..., xn), f2(x1, x2, ..., xn)]
```

So, **If f1 and f2 are nonlinear functions - this will be a nonlinear dimensionality reduction**. Simple as that!
And in the spirit of `a picture is worth a thousand words` :



Here you can see for 1-dimensional structure in 2D. The points lie along an S-shaped curve.

- PCA tries to describe the data with a linear 1-dimensional manifold, which is simply a line; of course a line fits these data quite bad.
- Isomap is looking for a nonlinear (i.e. curved!) 1-dimensional manifold, and should be able to discover the underlying S-shaped curve.

Exploratory Analysis Of Nonlinear Methods

In this stage I'll **analyze, implement and test** 2 non linear methods:

- Isomap
- Levina-Bickel

Datasets

Initialy I'll download a lot of known different datasets to use on: (arranged by #features)

- Swiss Roll
- Moons
- Iris
- California Housing
- Diabetes
- Boston House Prices
- Labeled Faces In The Wild (LFW)
- Olivetti Faces (AT&T)

*note: some of the data will be generated, randomized and noisey.

In [51]:

```
#### Swiss roll ####
print('Swiss Roll')
n,noise = 200,1
swiss_roll_data = make_swiss_roll(n)[0]
swiss_roll_with_noise_data = make_swiss_roll(n, noise)[0]
print('Number of objects = ', n)
print('Number of features = ',swiss_roll_data.shape[1])
print()

#### Moons ####
print('Moons')
n,noise = 200,0.1
moons_data = np.zeros((n,3))
moons_data[:,0:2] = make_moons(n)[0]
moons_data[:,2] = np.random.uniform(0,1,200)
moons_with_noise_data = np.zeros((n,3))
moons_with_noise_data[:,0:2] = make_moons(n,noise=noise)[0]
moons_with_noise_data[:,2] = np.random.uniform(0,1,200)
print('Number of objects = ', n)
print('Number of features = ',moons_data.shape[1])
print()

#### Iris ####
print('Iris')
iris_data = pd.DataFrame(load_iris().data)
print('Number of objects = ', iris_data.shape[0])
print('Number of features = ',iris_data.shape[1])
print()

#### California housing ####
print('California Housing')
california_data = pd.DataFrame(fetch_california_housing().data)
print('Number of objects = ', california_data.shape[0])
print('Number of features = ',california_data.shape[1])
print()

#### Diabetes ####
print('Diabetes')
diabetes_data = pd.DataFrame(load_diabetes().data)
print('Number of objects = ', diabetes_data.shape[0])
print('Number of features = ',diabetes_data.shape[1])
print()

#### Boston house prices ####
print('Boston House Prices')
boston_data = pd.DataFrame(load_boston().data)
print('Number of objects = ', boston_data.shape[0])
print('Number of features = ',boston_data.shape[1])
print()

#### LFW ####
print('LFW')
lfw_data = pd.DataFrame(fetch_lfw_people(resize = 0.5).data)
print('Number of objects = ', lfw_data.shape[0])
print('Number of features = ',lfw_data.shape[1])
print()

#### Olivetti faces (AT&T) ####
print('Olivetti Faces')
```

```
olivetti_data = pd.DataFrame(fetch_olivetti_faces().data)
print('Number of objects = ', olivetti_data.shape[0])
print('Number of features = ',olivetti_data.shape[1])
print()
```

```
Swiss Roll
Number of objects =  200
Number of features =  3

Moons
Number of objects =  200
Number of features =  3

Iris
Number of objects =  150
Number of features =  4

California Housing
Number of objects =  20640
Number of features =  8

Diabetes
Number of objects =  442
Number of features =  10

Boston House Prices
Number of objects =  506
Number of features =  13

LFW
Number of objects =  13233
Number of features =  2914

Olivetti Faces
Number of objects =  400
Number of features =  4096
```

Isomap Method

Following this (http://wearables.cc.gatech.edu/paper_of_week/isomap.pdf) paper by Joshua B. Tenenbaum,Vin de Silva, John C. Langford : A Global Geometric Framework for Nonlinear Dimensionality Reduction, and using this (https://github.com/scikit-learn/scikit-learn/blob/95d4f0841/sklearn/manifold/_isomap.py) implementation by Scikit-Learn.

Abstract

Scientists working with large volumes of high-dimensional data, such as global climate patterns, stellar spectra, or human gene distributions, regularly confront the problem of dimensionality reduction: finding meaningful low-dimensional structures hidden in their high-dimensional observations. The human brain confronts the same problem in everyday perception, extracting from its high-dimensional sensory inputs-30,000 auditory nerve fibers or 106 optic nerve fibers-a manageably small number of perceptually relevant features. Here we describe an approach to solving dimensionality reduction problems that uses easily measured local metric information to learn the underlying global geometry of a data set. Unlike classical techniques such as principal component analysis (PCA) and multidimensional scaling (MDS), our approach is capable of discovering the nonlinear degrees of freedom that underlie complex natural observations, such as human handwriting or images of a face under different viewing conditions. In contrast to previous algorithms for nonlinear dimensionality reduction, ours efficiently computes a globally optimal solution, and, for an important class of data manifolds, is guaranteed to converge asymptotically to the true structure.

Understanding Isomap

Isomap is similar to PCA. Although Isomap does linear estimations in the data point neighborhoods, the synergies extracted are nonlinear because these small neighborhoods are stitched together without trying to maintain linearity.

The following were the steps involved in estimating nonlinear synergies using Isomap:

1. Define neighbors for each data point
2. Find D, a matrix of inter-point distances
3. Find eigenvectors of τ(D), where τ(D) = −HSH/2, Sij = (Dij)/2 and Hij = dij − 1/N, where N is number of data points and d is Kronecker delta function.

In PCA we estimated the eigen values and eigen vectors of covariance of the data.
Similarly, this nonlinear approach to preserve inter-point distances on the manifold. The matrix D is similar to covariance matrix in PCA. D can actually be thought of as the covariance matrix in higher dimensions.
Since in an N-dimensional space, the dimensions are the data points, the covariance for a particular pair of dimensions is the distance between the data points that define those dimensions.

Although this method looks linear like PCA, the source of nonlinearity is the method in which inter-point distances are calculated.
Isomap do not use the Euclidean distances between the points. Rather, uses those distances only for points considered neighbors.
The rest of inter-point distances are calculated by finding the shortest path through the graph on the manifold using Floyd's algorithm.
The goal of the Isomap is to preserve the geodesic distances rather than the euclidian distances.
Geodesic distances are calculated by moving along the approximate nonlinear manifold with given data point and interpolation between them.

Implementation

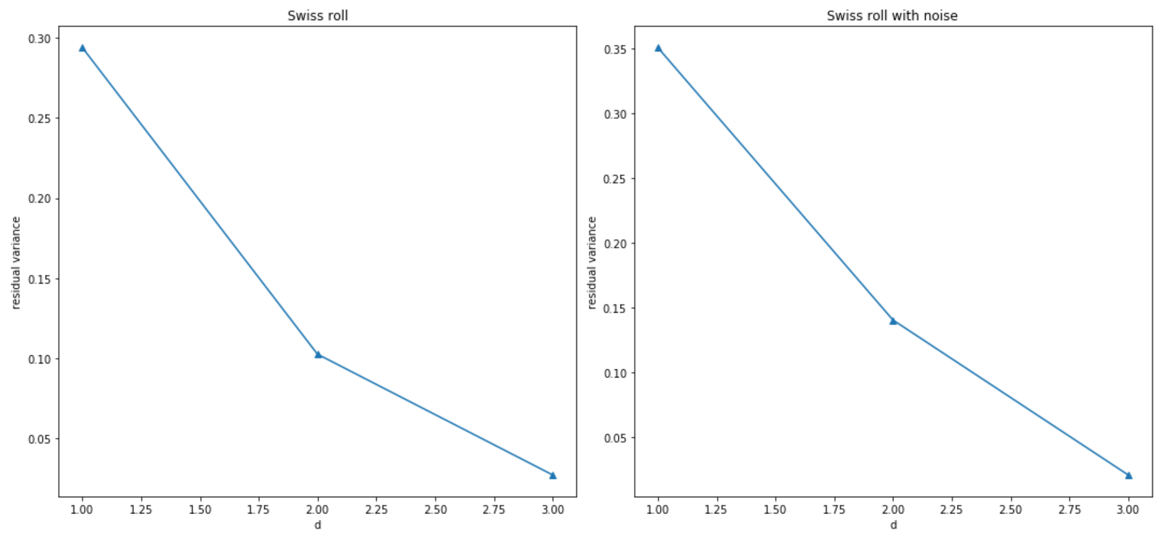
In [6]:

```
def compute_residual_variances_of_isomap(X,dims,n_neighbors = 5, plot_dependence = Fals
e, fig_name = "", title_add = ""):
    variances = []

    for m in dims:
        transformator = Isomap(n_components=m, n_neighbors=n_neighbors)
        transformator.fit(X)
        X_low = transformator.transform(X)
        D = squareform(pdist(X_low)).reshape(-1)
        D_fit = transformator.dist_matrix_.reshape(-1)
        variances.append(1 - np.corrcoef(D,D_fit)[0,1])

    if plot_dependence:
        plt.plot(dims, variances,'-^')
        plt.xlabel('dimensions')
        plt.ylabel('residual variance')
        if title_add != "":
            plt.title(title_add)
        if fig_name != "":
            plt.savefig(fig_name)

    return variances
```

Applying on Swiss Roll

In [7]:

```
f, axarr = plt.subplots(1, 2, figsize=(15,7))
n_neighbors = 10

# dimensionality estimation
dims = list(range(1, 4))
var_of_d = compute_residual_variances_of_isomap(swiss_roll_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
axarr[0].plot(dims,var_of_d,'-^')
axarr[0].set_title("Swiss roll")

var_of_d = compute_residual_variances_of_isomap(swiss_roll_with_noise_data, dims, n_nei
ghbors)

# plot dependence of residual variance from dimension
axarr[1].plot(dims,var_of_d,'-^')
axarr[1].set_title("Swiss roll with noise")

for index in range(2):
    axarr[index].set_xlabel('d')
    axarr[index].set_ylabel('residual variance')

f.tight_layout()
```

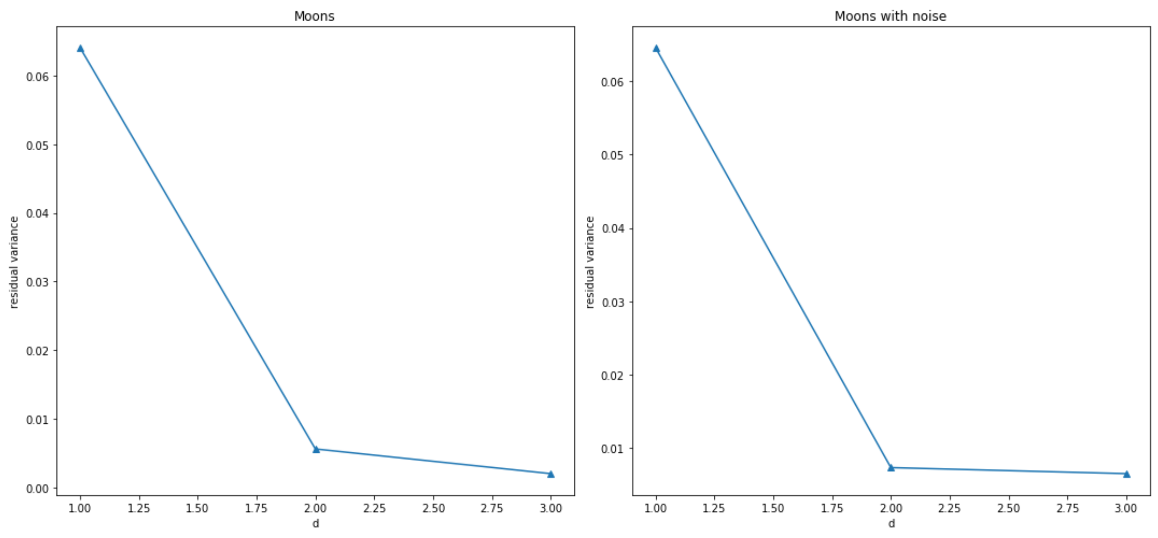


The Isomap method correctly determined the Swiss roll data dimension, even on noisy data.

Applying on Moons

In [8]:

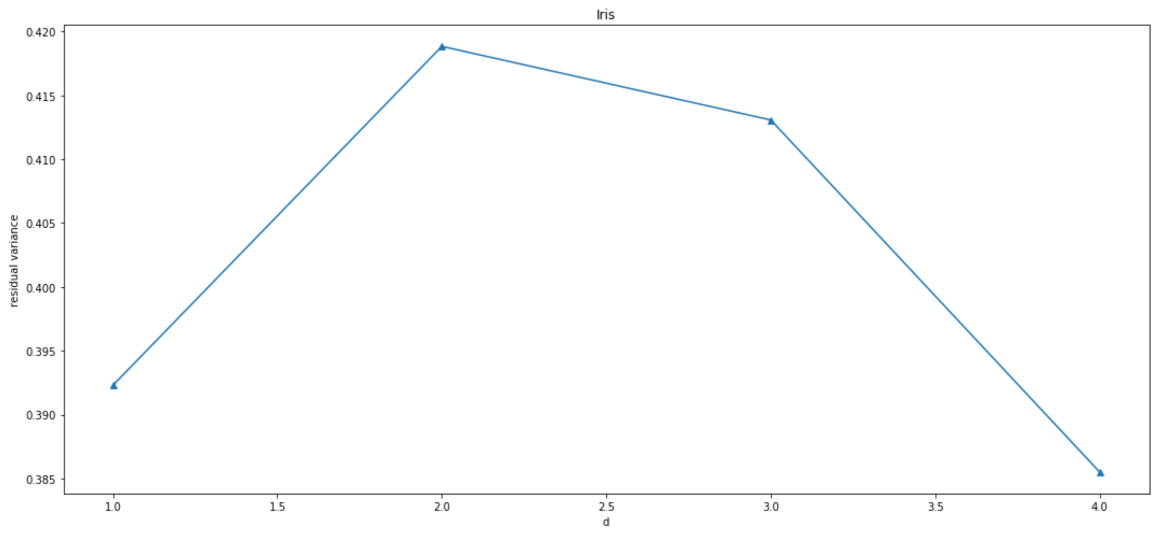
```
f, axarr = plt.subplots(1, 2, figsize=(15,7))
n_neighbors = 10

# dimensionality estimation
dims = list(range(1, 4))
var_of_d = compute_residual_variances_of_isomap(moons_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
axarr[0].plot(dims,var_of_d,'-^')
axarr[0].set_title("Moons")

var_of_d = compute_residual_variances_of_isomap(moons_with_noise_data, dims, n_neighbor
s)

# plot dependence of residual variance from dimension
axarr[1].plot(dims,var_of_d,'-^')
axarr[1].set_title("Moons with noise")

for index in range(2):
    axarr[index].set_xlabel('d')
    axarr[index].set_ylabel('residual variance')

f.tight_layout()
```



Applying on Iris

In [9]:

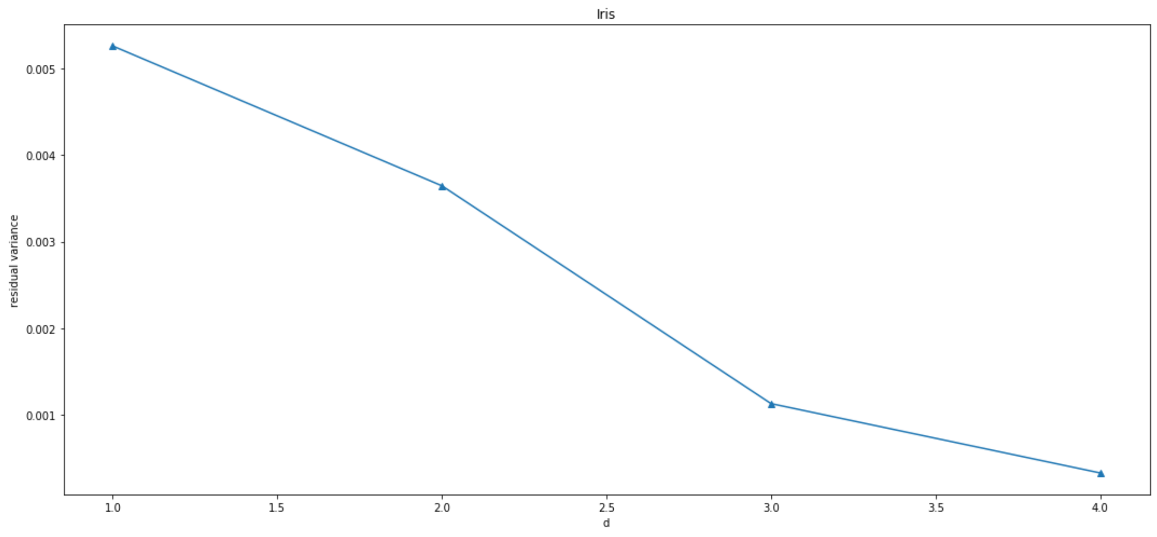
```
f = plt.figure(figsize=(15,7))
n_neighbors = 10

# dimensionality estimation
dims = list(range(1, iris_data.shape[1] + 1))
var_of_d = compute_residual_variances_of_isomap(iris_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
plt.plot(dims,var_of_d,'-^')
plt.title("Iris")

plt.xlabel('d')
plt.ylabel('residual variance')

f.tight_layout()
```



With the hyperparameter n = 10, the algorithm produced an unintelligible graph of the dependence of the residual variance on the dimension d.
I'll try on different n value!

In [10]:

```
f = plt.figure(figsize=(15,7))
n_neighbors = 50

# dimensionality estimation
dims = list(range(1, iris_data.shape[1] + 1))
var_of_d = compute_residual_variances_of_isomap(iris_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
plt.plot(dims,var_of_d,'-^')
plt.title("Iris")

plt.xlabel('d')
plt.ylabel('residual variance')

f.tight_layout()
```

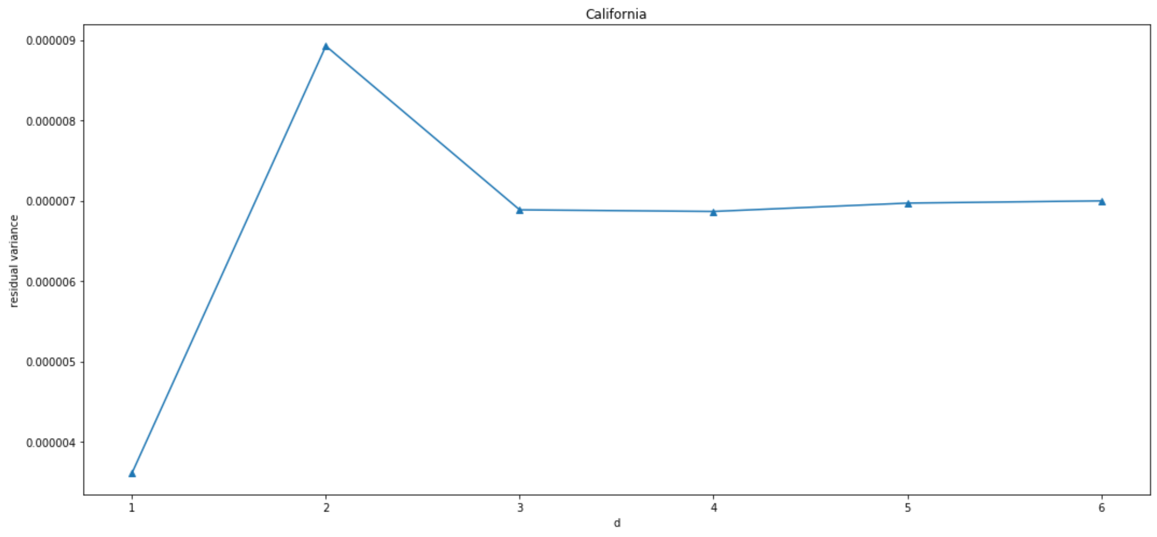


Now the Isomap algorithm has yielded better results.

Applying on California Housing

In [11]:

```
f = plt.figure(figsize=(15,7))
n_neighbors = 300

# dimensionality estimation
california_data_reduced = california_data[:4000]
dims = [1, 2, 3, 4, 5, 6]
var_of_d = compute_residual_variances_of_isomap(california_data_reduced, dims, n_neighb
ors)
# plot dependence of residual variance from dimension
plt.plot(dims,var_of_d,'-^')
plt.title("California")

plt.xlabel('d')
plt.ylabel('residual variance')

f.tight_layout()
```

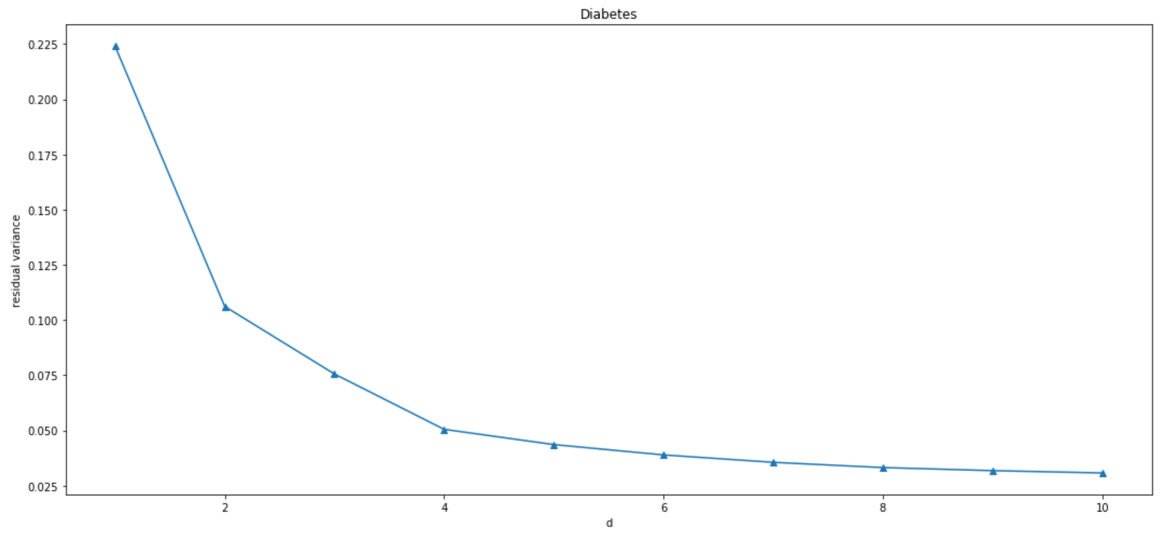


*note: There was not enough memory for 20,000 objects, so I cut the sample to 4,000 and ran the algorithm.

Applying on Diabetes

In [12]:

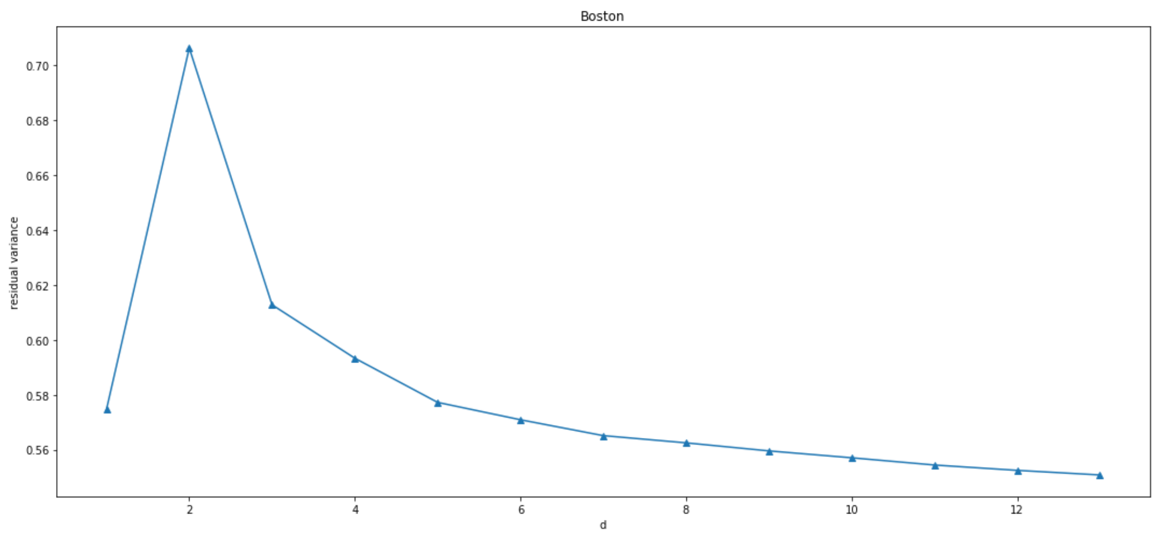
```
f = plt.figure(figsize=(15,7))
n_neighbors = 10

# dimensionality estimation
dims = list(range(1, diabetes_data.shape[1] + 1))
var_of_d = compute_residual_variances_of_isomap(diabetes_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
plt.plot(dims,var_of_d,'-^')
plt.title("Diabetes")

plt.xlabel('d')
plt.ylabel('residual variance')

f.tight_layout()
```



Applying on Boston House Prices

In [13]:

```
f = plt.figure(figsize=(15,7))
n_neighbors = 10

# dimensionality estimation
dims = list(range(1, boston_data.shape[1] + 1))
var_of_d = compute_residual_variances_of_isomap(boston_data, dims, n_neighbors)

# plot dependence of residual variance from dimension
plt.plot(dims,var_of_d,'-^')
plt.title("Boston")

plt.xlabel('d')
plt.ylabel('residual variance')

f.tight_layout()
```



Applying on LFW and on Olivetti Faces (AT&T)

Very high dimensional data.
This will take way too much time and calculation to run.
For the Isomap algorithm to work on LFW data or on Olivetti Faces data - a lot of time and memory are required, and my local machine can't handle that.

In [18]:

```
#### LFW ####
# f = plt.figure(figsize=(15,7))

# # dimensionality estimation
# dims = [1, 2, 4, 8, 20, 28, 36, 50, 70]
# var_of_d = compute_residual_variances_of_isomap(lfw_data, dims, n_neighbors)

# # plot dependence of residual variance from dimension
# plt.plot(dims,var_of_d,'-^')
# plt.title("Labeled faces wild. n = {}".format(n_neighbors))

# plt.xlabel('d')
# plt.ylabel('residual variance')

# f.tight_layout()





#### Olivetti Faces ####
# f = plt.figure(figsize=(15,7))
# n_neighbors = 10

# # dimensionality estimation
# dims = [1, 4, 10, 15, 25, 50, 70, 100, 150, 300, 500, 800, 1400, 2000, 3000, 4000]
# var_of_d = compute_residual_variances_of_isomap(olivetti_data, dims, n_neighbors)

# # plot dependence of residual variance from dimension
# plt.plot(dims,var_of_d,'-^')
# plt.title("Olivetti")

# plt.xlabel('d')
# plt.ylabel('residual variance')

# f.tight_layout()
```

Levina-Bickel Method

Following this (https://www.stat.berkeley.edu/~bickel/mldim.pdf) paper by Levina E., Bickel P.J. : Maximum Likelihood Estimation of Intrinsic Dimension, and this (https://codegists.com/snippet/python/intdim_mlepy_mehdidc_python) implementation.

Abstract:
We propose a new method for estimating intrinsic dimension of a dataset derived by applying the principle of maximum likelihood to the distances between close neighbors. We derive the estimator by a Poisson process approximation, assess its bias and variance theoretically and by simulations, and apply it to a number of simulated and real datasets. We also show it has the best overall performance compared with two other intrinsic dimension estimators.

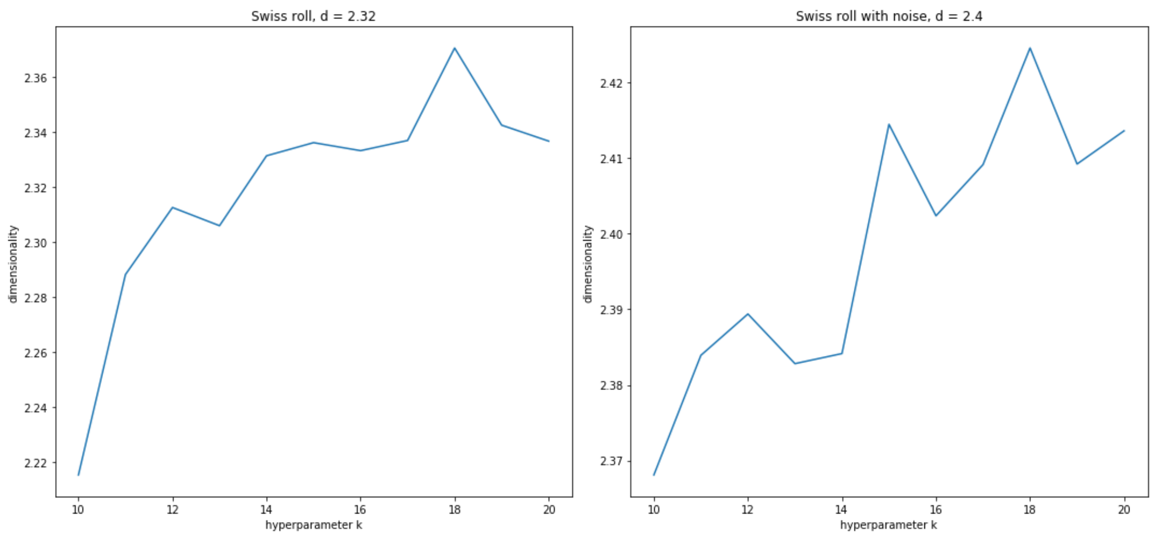
In [19]:

```
def intrinsic_dim_sample_wise(X, k=5, dist = None):
    if dist is None:
        neighb = NearestNeighbors(n_neighbors=k+1,algorithm='ball_tree').fit(X)
        dist, ind = neighb.kneighbors(X)
    dist = dist[:, 1:(k+1)]
    assert dist.shape == (X.shape[0], k)
    assert np.all(dist > 0)
    d = np.log(dist[:, k - 1: k] / dist[:, 0:k - 1])
    d = d.sum(axis=1) / (k - 2)
    d = 1. / d
    intdim_sample = d
    return intdim_sample


def intrinsic_dim_scale_interval(X, k1=10, k2=20, dist = None):
    intdim_k = []
    if dist is None:
        neighb = NearestNeighbors(n_neighbors=k+1,algorithm='ball_tree').fit(X)
        dist, ind = neighb.kneighbors(X)

    for k in range(k1, k2 + 1):
        m = intrinsic_dim_sample_wise(X, k,dist).mean()
        intdim_k.append(m)
    return intdim_k


def bootstrap_intrinsic_dim_scale_interval(X, nb_iter=100, random_state=None, k1 = 10,
k2 = 20,average = True, plot_dependence = False, fig_name = "", title_add = ""):
    if random_state is None:
        rng = np.random
    else:
        rng = np.random.RandomState(random_state)



    X = pd.DataFrame(X)
    X = X.drop_duplicates()

    nb_examples = X.shape[0]
    results = []

    neighb = NearestNeighbors(n_neighbors=k2+1,algorithm='ball_tree').fit(X)
    dist, ind = neighb.kneighbors(X)

    for i in range(nb_iter):
        idx = np.unique(rng.randint(0, nb_examples - 1, size=nb_examples))
        results.append(intrinsic_dim_scale_interval(X.iloc[idx], k1, k2, dist[idx,:]))
    results = np.array(results)
    if plot_dependence:
        dim_of_k = results.mean(axis = 0)
        levina_dimension = dim_of_k.mean()
        print('Dimension averaged over (k=',k1,'..',k2,') = ',levina_dimension)
        plt.plot(np.arange(k1,k2+1),dim_of_k)
        plt.xlabel('k - nearest neighbours')
        plt.ylabel('Dimension')
        if title_add == "":
            plt.title('Original dimension = '+str(X.shape[1])+', L-B dimension = '+str(
levina_dimension))
        else:
            plt.title(title_add + ', original dimension = '+str(X.shape[1])+', L-B dime
nsion = '+str(levina_dimension))
        if fig_name != "":
            plt.savefig(fig_name)
    if average:
        return results.mean(axis = 0)
    else:
        return results
```

Applying on Swiss Roll

In [92]:

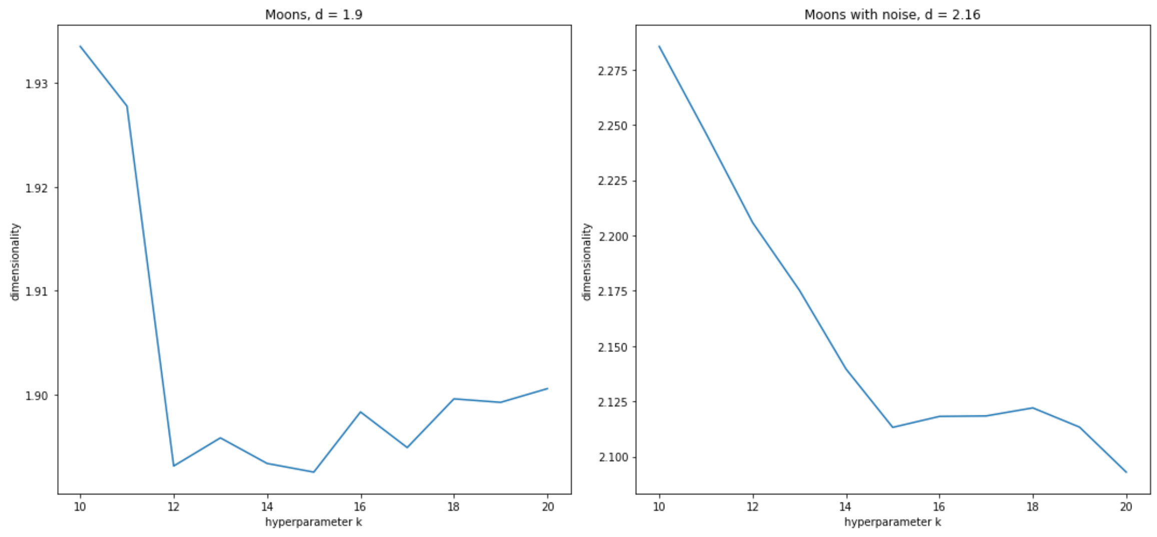
```
k1,k2 = 10,20
fig, axarr = plt.subplots(1, 2,figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(swiss_roll_data, nb_iter=50, k1=k1, k
2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
axarr[0].plot(np.arange(k1,k2+1),dim_of_k)
axarr[0].set_title("Swiss roll, d = " + str(levina_dimension))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(swiss_roll_with_noise_data, nb_iter=5
0, k1=k1, k2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
axarr[1].plot(np.arange(k1,k2+1),dim_of_k)
axarr[1].set_title("Swiss roll with noise, d = " + str(levina_dimension))

for index in range(2):
    axarr[index].set_xlabel("hyperparameter k")
    axarr[index].set_ylabel("dimensionality")

fig.tight_layout()
```



Applying to Moons

In [94]:

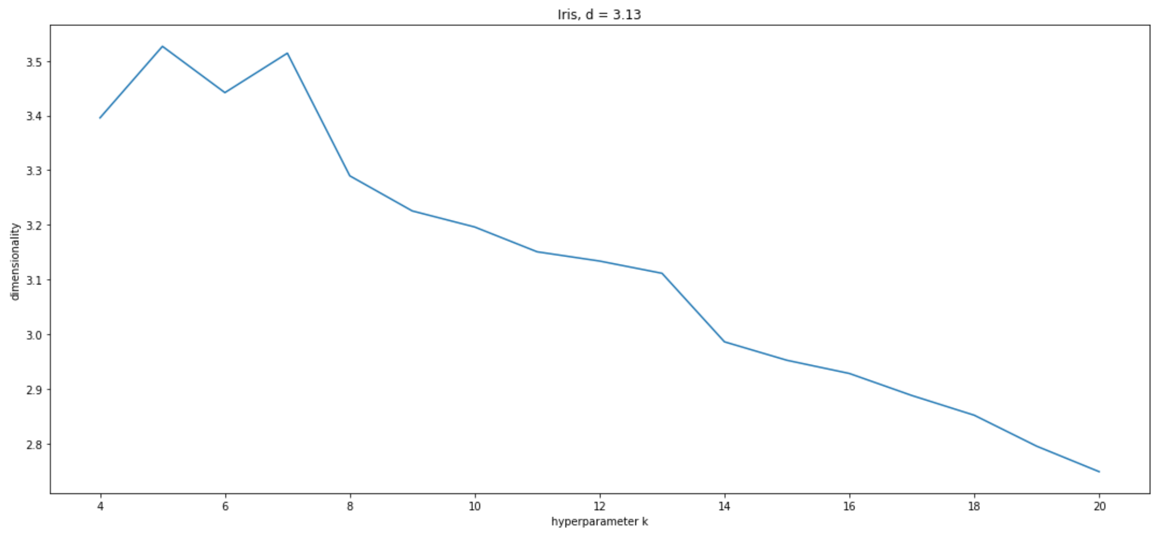
```
k1,k2 = 10,20
fig, axarr = plt.subplots(1, 2,figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(moons_data, nb_iter=50, k1=k1, k2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
axarr[0].plot(np.arange(k1,k2+1),dim_of_k)
axarr[0].set_title("Moons, d = " + str(levina_dimension))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(moons_with_noise_data, nb_iter=50, k1
=k1, k2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
axarr[1].plot(np.arange(k1,k2+1),dim_of_k)
axarr[1].set_title("Moons with noise, d = " + str(levina_dimension))

for index in range(2):
    axarr[index].set_xlabel("hyperparameter k")
    axarr[index].set_ylabel("dimensionality")

fig.tight_layout()
```



Applying on Iris

In [22]:

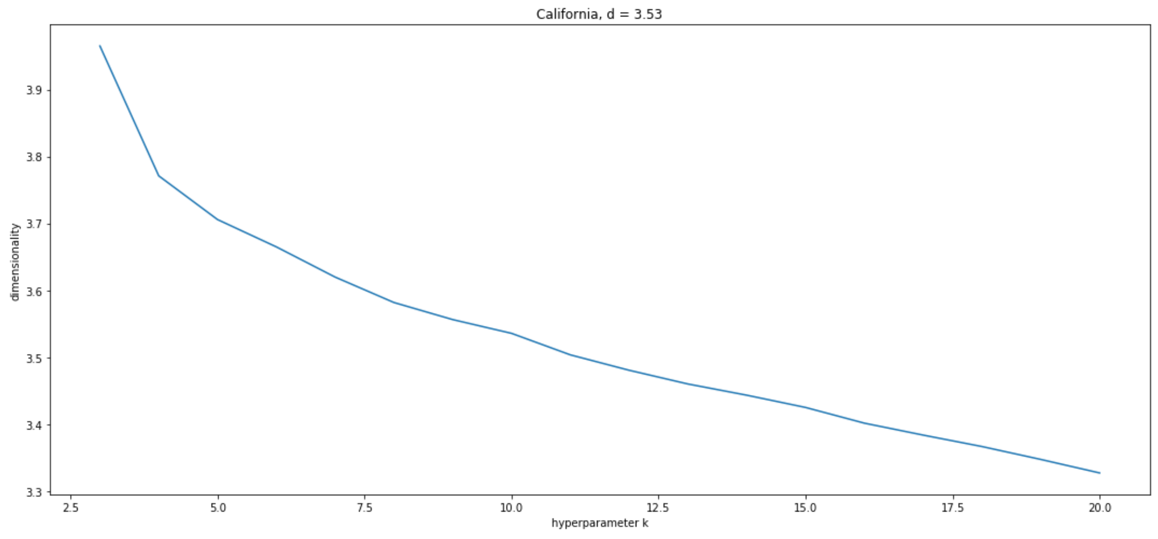
```
k1,k2 = 4,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(iris_data, nb_iter=50, k1=k1, k2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("Iris, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```



Applying on California Housing

In [23]:

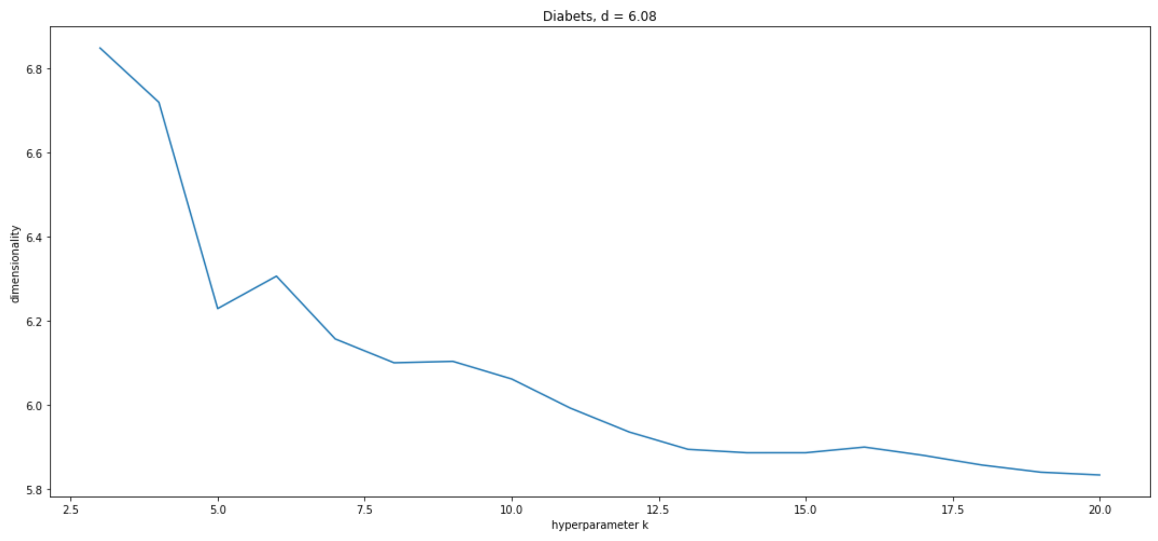
```
k1,k2 = 3,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(california_data, nb_iter=50, k1=k1, k
2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("California, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```



Applying on Diabetes

In [24]:

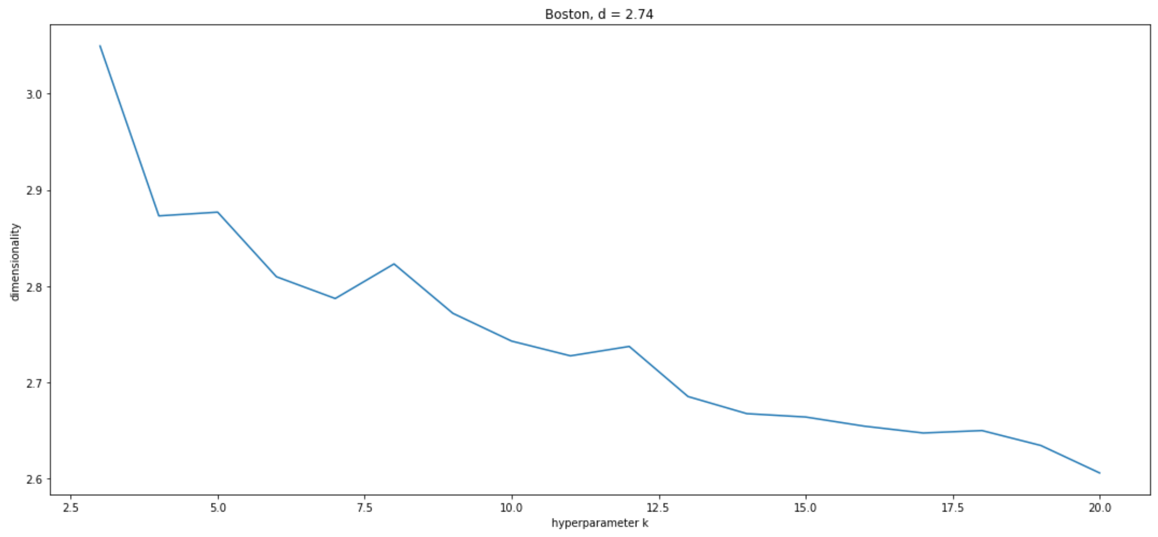
```
k1,k2 = 3,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(diabetes_data, nb_iter=50, k1=k1, k2=
k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("Diabets, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```



Applying on Boston House Prices

In [25]:

```
k1,k2 = 3,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(boston_data, nb_iter=50, k1=k1, k2=k2
)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("Boston, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```



Applying on LFW

In [ ]:

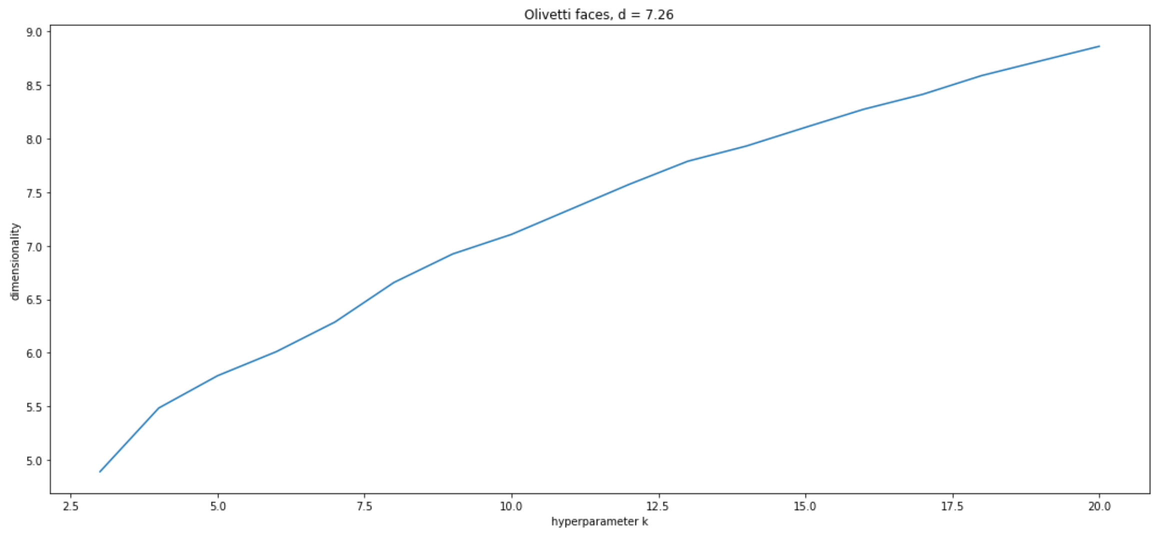
```
k1,k2 = 3,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(lfw_data, nb_iter=50, k1=k1, k2=k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("Labeled faces wild, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```

*note: this will takes a lot of computational power to comptue (or time)

Applying on Olivetti faces (AT&T)

In [27]:

```
k1,k2 = 3,20
fig = plt.figure(figsize=(15,7))

# dimensionality estimation
dim_of_k = bootstrap_intrinsic_dim_scale_interval(olivetti_data, nb_iter=50, k1=k1, k2=
k2)
levina_dimension = round(dim_of_k.mean(),2)

# plot dependence of dimension estimation from k
plt.plot(np.arange(k1,k2+1),dim_of_k)
plt.title("Olivetti faces, d = " + str(levina_dimension))

plt.xlabel("hyperparameter k")
plt.ylabel("dimensionality")

fig.tight_layout()
```



Summary & Conclusions

I experimente, implemente, test and analyze **two popular nonlinear methods** (Isomap, Levina-Bickele) on varius datasets, in order to check the magnitude of their `Dimensionality Reduction` .

To test their performance, for which the true dimension is known, both generated data and real data, were used.
From the results on the generated data, one can find the distinctive features of the methods that must be kept in mind in order to draw conclusions about the real data.

Results on generated data

According to the results, I cannot conclude that the data is in a variety of dimensions of a smaller number of features.
Both nonlinear methos failed to reduced any of the dimensions.
Which make sense due to it nature of randomize data.

Results on real data

It is almost impossible to know what is the actual dimension the data is in.
Both method done a great job reduceing the real data dimensions.
Both methos gave almost the same results.

From all these articals, codes, documentations and expirements I have learned a lot. Here are some key points I had to mentioned...

PCA

`Advantages:`

- Removes Correlated Features: In a real world scenario, this is very common that you get thousands of features in your dataset. You cannot run your algorithm on all the features as it will reduce the performance of your algorithm and it will not be easy to visualize that many features in any kind of graph. So, you MUST reduce the number of features in your dataset.
- Improves Algorithm Performance: With so many features, the performance of your algorithm will drastically degrade. PCA is a very common way to speed up your Machine Learning algorithm by getting rid of correlated variables which don't contribute in any decision making. The training time of the algorithms reduces significantly with less number of features. So, if the input dimensions are too high, then using PCA to speed up the algorithm is a reasonable choice.
- Reduces Overfitting: Overfitting mainly occurs when there are too many variables in the dataset. So, PCA helps in overcoming the overfitting issue by reducing the number of features.
- Improves Visualization: It is very hard to visualize and understand the data in high dimensions. PCA transforms a high dimensional data to low dimensional data (2 dimension) so that it can be visualized easily.

`Disadvantages:`

- Independent variables become less interpretable: After implementing PCA on the dataset, your original features will turn into Principal Components. Principal Components are the linear combination of your original features. Principal Components are not as readable and interpretable as original features.
- Data standardization: PCA is affected by scale, so You must standardize your data before implementing PCA, otherwise it will get it wrong, and won't be able to find the optimal Principal Components.
- Information Loss: Although Principal Components try to cover maximum variance among the features in a dataset, if we don't select the number of Principal Components with care, it may miss some information as compared to the original list of features.

ISOMAP

`Steps`

- Calculate (Euclidean) distances between all pairs of data points (i,j) and store them in distance matrix DX as DX(i,j)
- Construct neighborhood graph by connecting two points i and j
  - if they are closer thanε(ε-Isomap), DX(i,j)<ε
  - if i is one of the K nearest neighbors of j (K-Isomap)
  - Let the „weight“ of the edge between i and j be the distance between them, that is DX(i,j)
- Compute shortest path between each pair of points (using Dijkstra's shortest path algorithm for example), store the path lengths in the matrix DG as an approximate geodesic distance

`Advantages:`

- Works with nonlinear data
- Easy to understand and implement
- Preserves “true” relationship between data points
- Globally optimal even if input space is highly folded
- Guaranteed to lower-dimensional representation (asymptotically recover)
- Also Improves Visualization:

`Disadvantages:`

- Computationally expensive
- Stability depends heavily on parametersεand K
- Need enough data points, o.w. geodesic distance approximation is inaccurate
- Neighborhood graph creation is tricky and slightly wrong parameters can produce bad results
- Have difficulties with “holes” in the data - Isomap performs poorly when manifold is not well sampled and contains holes

More References:

Rafael Irizarry's book (https://rafalab.github.io/dsbook/large-datasets.html)

python-machine-learning-book (https://github.com/rasbt/python-machine-learning-book/tree/master/faq)

dimensionality-reduction-techniques (https://github.com/vashistak/dimensionality-reduction-techniques)

Python examples of isomap algorithm
(https://gist.github.com/Mashimo/b8a8d4dc18bf6875c8547134b543898f)

The Quest for Variance : PCA, MDS & ISOMAP (http://math.gmu.edu/~berry/Presentations/PCAMDS.pdf)

Dimensionality Reduction: A Comparative Review
(https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf)

Python examples of Principal Component Analysis
(https://gist.github.com/Mashimo/69f0972d51358d65f088a7147dfc5ff1)

Dimensionality Reduction — PCA, ICA and Manifold learning
(https://towardsdatascience.com/dimensionality-reduction-pca-ica-and-manifold-learning-65393010253e)

Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases
(http://www.cs.ucr.edu/~eamonn/kais_2000.pdf)

Dimensionality reduction for fast and accurate video search and retrieval in a large scale database
(https://ieeexplore.ieee.org/document/6780074)

Comparative Analysis of Linear and Nonlinear Dimension Reduction Techniques on Mass Cytometry Data
(https://www.biorxiv.org/content/10.1101/273862v2.full)