

# Feature selection and extraction

We decided to write a work in English. So our theme is "Feature selection and extraction". What does that mean? Raw data can be complex and difficult to process without extracting or selecting appropriate features

beforehand. We review common methods of feature selection and extraction and test them on real data. Both feature selection and extraction reduce the dimensionality of data, but do it differently. While selection only finds best features(for a given task) out of existing ones, extraction methods try to find a representation in a new, smaller feature space with the least amount of lost data.

A lot of people nowadays think about feature selection as a way to reduce dimensionality of data and about feature extraction only as a visualization tool for high dimensional data. But those can be used for many other reasons and applications. We explore some of those methods and try to show other advantages of them. Also we try some close-to-SOTA methods for feature

Our work consists of 3 main parts:

1. Feature selection methods
2. Feature extraction methods
3. DL applications

These parts were made by Kirill, Timothy and Dmitriy respectively.

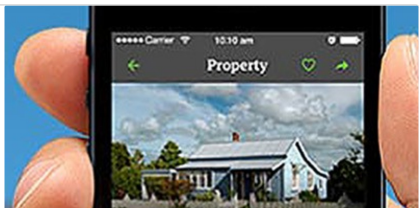
## 0. Dataset and baseline

Choosing proper dataset was a hard task. Some datasets are just too boring like ones from UCI, other ones are too big (especially for some complex methods). At the same time we wanted somehow known dataset, so we have chosen Mobile Price Classification.

Mobile Price Classification

classify mobile price range

<https://www.kaggle.com/iabhishekofficial/mobile-price-classification>



It's a classification task of finding a right price sector for a phone by its features. The dataset has almost a 1000 Kernels on kaggle even though its 2 years old. It's not so big, but at the same time big enough: 2000 samples with 20 columns.

Baseline solution is done by using LogisticRegressionCV and scored  $f1\_macro$  of **0.9498**.

## 1. Feature selection


The idea here is simple - to reduce the number of features. It is not an easy task, because the total number of possible subsets of features is exponentially big. Therefore, we need to come up with

some method that works in a reasonable time.

There are 2 main ideas - *filter* and *wrapper* methods of feature selection. I'll start with the first one.

Consider a ranking mechanism used to grade the features (variables) and the features are then removed by setting a threshold. These ranking methods are categorized as filter methods because they filter the features before feeding to a learning model. Filter methods are based on two concepts “**relevance**” and “**redundancy**”.

The main idea of most *filter* methods is the same - to find a metric to compare any feature with the target-feature and leave features based on that. There is a well-known tool for using *filter* methods in sklearn:

sklearn.feature\_selection.GenericUnivariateSelect - scikit-learn 0.23.1 documentation  
scikit-learn: machine learning in Python  
 [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.GenericUnivariateSelect.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.GenericUnivariateSelect.html)



It lets using any function as scoring function and choose either a number of features to leave by that metric, or leave any percentage of features.

- Correlation criteria (Pearson Correlation Criteria, PCC) [**Relevance**]

Here a Pearson correlation coefficient is used. It is found by the formula:

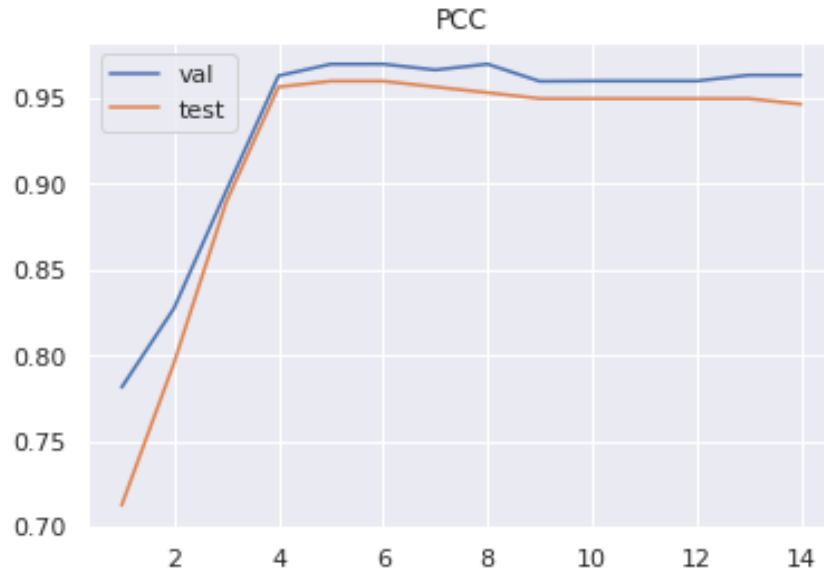
$$\rho_{x^j, t} := \frac{\text{cov}(x^j, t)}{\sqrt{\text{var}(x^j) \text{var}(t)}},$$

Where  $\text{cov}()$  and  $\text{var}()$  denote the covariance and variance, respectively.

It is counted between all the features and the target leaving highest scores.

**Implementation** <https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.pearsonr.html>

I present a graph with the dependence of *f1\_macro* score on the number of features left.



best score 0.96 with 6 features  
+0.73% from baseline

For testing I use a validation and test sets. I find best number of features to leave by validation and get the resulting score on test set.

It is clearly seen that by using the easiest method we get an increase in score even though almost 75% of features are removed!!

- Mutual Information (MI, Information Gain, IG) **[Redundancy]**

The MI can be described using the concept given by Shannon's definition of entropy:

$$H(\mathbf{X}) := - \sum_{\mathbf{x}} p(\mathbf{x}) \log(p(\mathbf{x}))$$

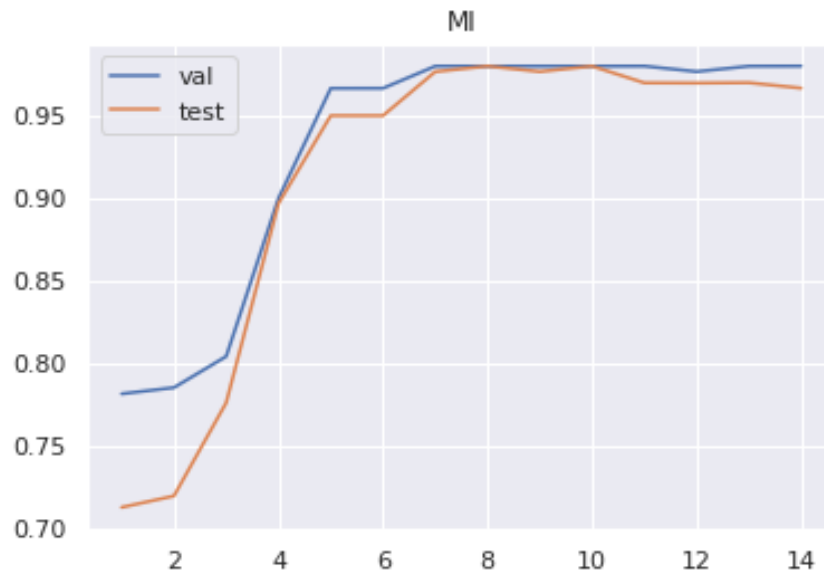
It shows the uncertainty in the random variable  $\mathbf{X}$ .

In feature selection, we need to maximize the mutual information (relevance) between the feature and the target variable. So we find the MI, which is the relative entropy between the joint distribution and product distribution:

$$\text{MI}(\mathbf{X}; \mathbf{T}) := \sum_{\mathbf{x}} \sum_{\mathbf{t}} p(\mathbf{x}^j, \mathbf{t}) \log \frac{p(\mathbf{x}^j, \mathbf{t})}{p(\mathbf{x}^j)p(\mathbf{t})}$$

For maximizing this MI, a greedy step-wise selection algorithm is adopted. On each step the value of MI is counted for (all the features and a new candidate) and (target). Taking the argmax of that we get the best feature to add to a set.

Implementation [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.mutual\\_info\\_classif.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html)

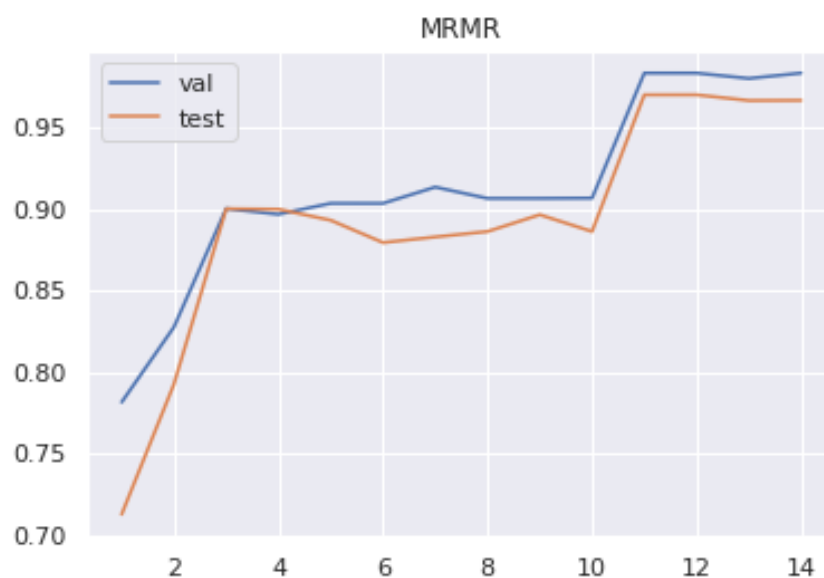


best score 0.9666 with 14 features  
+1.77% from baseline

Even though it decides to leave 14/20 features, it increases the score even more than PCC!

The next method is pretty hard in math part so I'm not going to explain it in details and only going to show the implementation and the results of the tests. Information about all of them can be found across the internet easily.

- Minimal-Redundancy-Maximal-Relevance (MRMR)



```
best score 0.9699 with 11 features
+2.11% from baseline
```

- Worth mentioning methods if you are willing to explore more:

Chi squared, Markov Blanket, Consistency-based and Fast Consistency-based Filters (didn't work for me, tho implementations can be found), Interact method.

*Wrapper methods.*


As seen previously, filter methods select the optimal features to be passed to the learning model, i.e., classifier, regression, etc. Wrapper methods, on the other hand, integrate the model within the feature subset search. In this way, different subsets of features are found or generated and evaluated through the model. The fitness of a feature subset is evaluated by training and testing it on the model. Thus in this sense, the algorithm for the search of the best suboptimal subset of the feature set is essentially “wrapped” around the model.

- Sequential Feature Selector (SFS)

This method is implemented and described very nicely in the mlxtend library:

#### Sequential Feature Selector

Implementation of sequential feature algorithms (SFAs) -- greedy search algorithms -- that have been developed as a suboptimal solution to the computationally often not feasible exhaustive search. from mlxtend.feature\_selection import SequentialFeatureSelector Sequential feature selection algorithms are a family of greedy search algorithms that are used to

 [http://rasbt.github.io/mlxtend/user\\_guide/feature\\_selection/SequentialFeatureSelector/](http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/)

```
best score 0.9866 with 6 features
+3.53% on baseline
```

It takes more time, but gives the best results with only 6/20 features! It probably can't be used for bigger datasets or models, but a cool idea and could be helpful.

# 2.Feature Extraction

Besides feature selection, another handy technique which can be used to reduce features dimension is feature extraction, which allows to produce a set of new lower dimension features using the initial dataset.

In this project following feature extraction algorithms are used and compared:

- [Multidimensional Scaling \(MDS\)](#)
- [Isomap](#)
- [Locally-linear embeddings \(LLE\)](#)
- [Laplacian Eigenmaps](#)
- [Linear Discriminant Analysis \(LDA\)](#)
- [T-SNE](#)

# MultiDimensional Scaling

Takes an input matrix giving dissimilarities between pairs of items and outputs a coordinate matrix whose configuration minimizes a loss function called *strain*.

$$\text{Strain}_D(x_1, x_2, \dots, x_N) = \left( \frac{\sum_{i,j} (b_{ij} - \langle x_i, x_j \rangle)^2}{\sum_{i,j} b_{ij}^2} \right)^{1/2}$$

Steps of a Classical MDS algorithm:

Classical MDS uses the fact that the coordinate matrix  $X$  can be derived by eigenvalue decomposition from  $B = X X'$ . And the matrix  $B$  can be computed from proximity matrix  $D$  by using double centering.

1. Set up the squared proximity matrix  $D^{(2)} = [d_{ij}^2]$
2. Apply double centering:  $B = -\frac{1}{2} J D^{(2)} J$  using the centering matrix  $J = I - \frac{1}{n} 11'$  where  $n$  is the number of objects.
3. Determine the  $m$  largest eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_m$  and corresponding eigenvectors  $e_1, e_2, \dots, e_m$  of  $B$  (where  $m$  is the number of dimensions desired for the output).
4. Now,  $X = E_m \Lambda_m^{1/2}$ , where  $E_m$  is the matrix of  $m$  eigenvectors and  $\Lambda_m$  is the diagonal matrix of  $m$  eigenvalues of  $B$ .

# Isomap

Isomap is a nonlinear dimensionality reduction method. Isomap is used for computing a quasi-isometric, low-dimensional embedding of a set of high-dimensional data points.

Isomap algorithm:

- Determine the neighbors of each point.
  - All points in some fixed radius.
  - $K$  nearest neighbors.
- Construct a neighborhood graph.
  - Each point is connected to other if it is a  $K$  nearest neighbor.
  - Edge length equal to Euclidean distance.
- Compute shortest path between two nodes.
  - Dijkstra's algorithm
  - Floyd–Warshall algorithm
- Compute lower-dimensional embedding.
  - Multidimensional scaling



# Locally-linear embedding and Laplacian Eigenmaps

**LLE** was presented at approximately the same time as Isomap. It has several advantages over Isomap, including faster optimization (when implemented to take advantage of sparse matrix algorithms).

LLE also begins by finding a set of the nearest neighbours of each point. It then computes a set of weights for each point that best describes the point as a linear combination of its neighbours. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbours. LLE tends to handle non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities.

**Laplacian Eigenmaps** uses spectral techniques to perform dimensionality reduction. This technique relies on the basic assumption that the data lies in a low-dimensional manifold in a high-dimensional space. This algorithm cannot embed out-of-sample points, but techniques based on Reproducing kernel Hilbert space regularization can be used in order to add this capability.

# PCA

PCA is one of the most commonly used algorithms for feature extraction.

Algorithm:

Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one that minimizes the average squared distance from a point to the line.

The next best-fitting line can be similarly chosen from directions perpendicular to the first. Repeating this process yields an orthogonal basis in which different individual dimensions of the data are uncorrelated. These basis vectors are called principal components, and several related procedures principal component analysis (PCA).

# T-SNE and LDA

**T-distributed Stochastic Neighbor Embedding (t-SNE)** algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects are assigned a higher probability, while dissimilar points are assigned a very low probability. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the KL divergence between the two distributions with respect to the locations of the points in the map. While the original algorithm uses the Euclidean distance between objects as the base of its similarity metric, this can be changed as appropriate. T-SNE is a very popular algorithm used for visualization of high-dimensional data in a low-dimensional space of two or three dimensions.

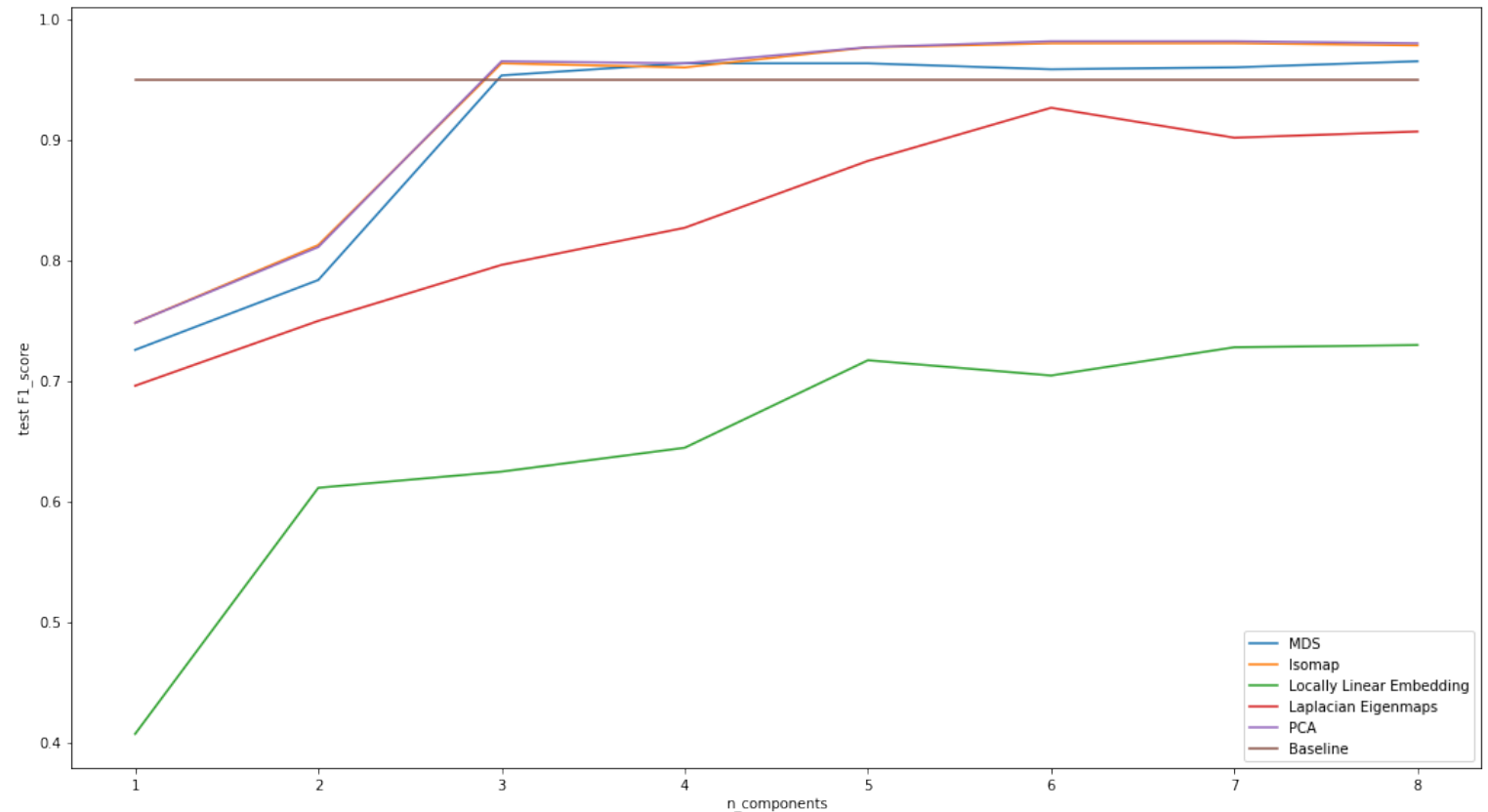
**Linear Discriminant Analysis (LDA)** is supervised feature extraction method. Similar to PCA, LDA calculates the projection of data along a direction; however, rather than maximizing the variation of data, LDA utilizes label information to get a projection maximizing the ratio of between-class variance to within-class variance.

# Comparison

For comparison of methods previously described mobile price classification dataset was used.

MDS, Isomap, LLE, Laplacian Eigenmaps and PCA methods were tested with  $n = [1, 2, 3, 4, 5, 6, 7, 8]$  number of components.

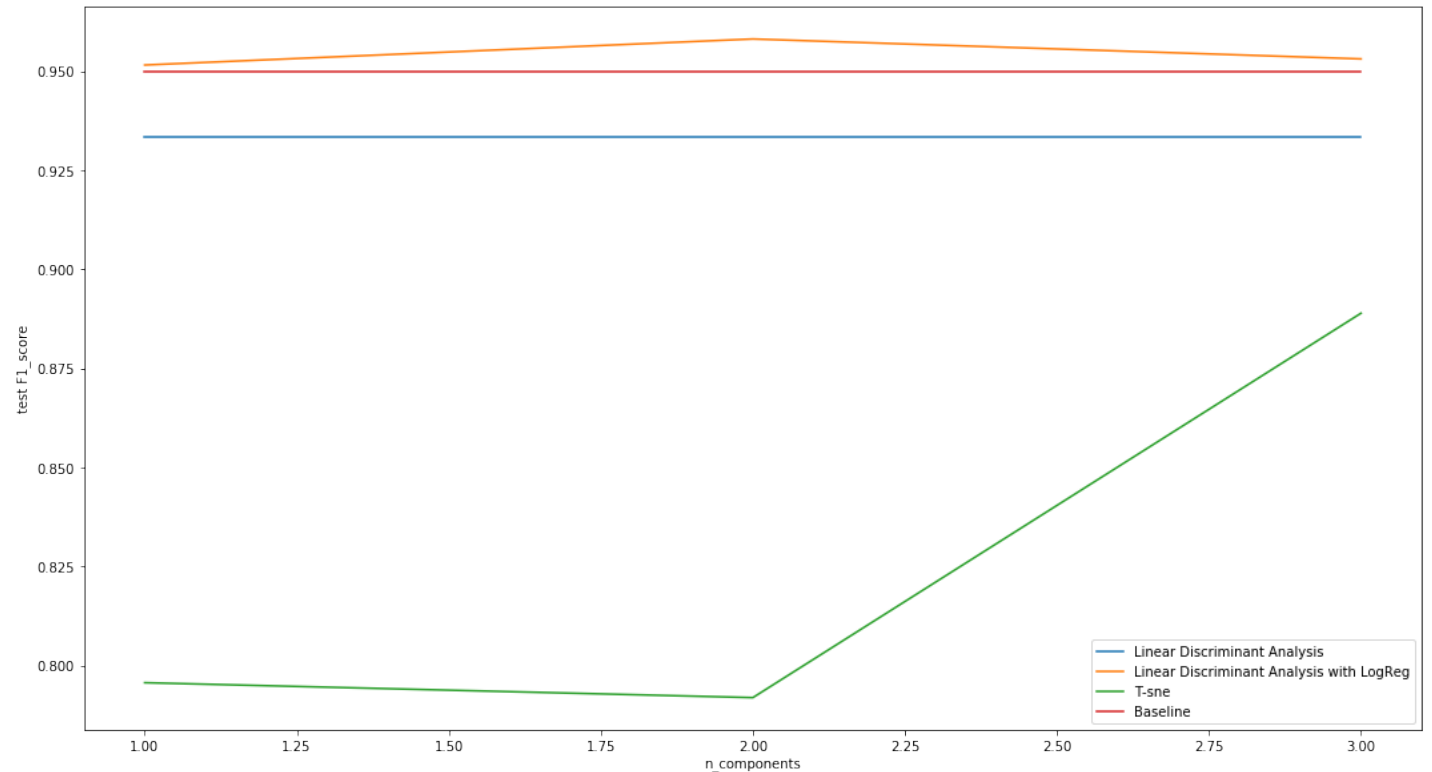
- MDS, PCA, Isomap noticeably outperform LLE and Laplacian Eigenmaps for any given number of components
- For given dataset 3 features is enough for MDS, PCA and Isomap to reach and exceed baseline score, while LLE and Laplacian Eigenmaps do not reach baseline even with larger amount of features



# Comparison

T-SNE and LDA can only be used with  $n = [1, 2, 3]$  components (for LDA  $n\_components = \min(n\_classe-1, n\_features)$ ). Also for testin LDA it was first used as linear classificator on its own and also for dimension reduction before classification with logistic regression.

- LDA performs better when used for dimension reduction for logistic regression classification
- Both LDA methods outperform t-SNE however only using Logistic Regression as classificator method exceeds baseline
- T-SNE works significantly better with 3 components (features) than with lower number



# Conclusion

Results of the following experiments showed that:

- PCA, MDS and Isomap are one of the best unsupervised algorithms for feature extraction, allowing to significantly downscale dimension of data while maintaining or even increasing score of the base model
- LDA in couple with good classifier is a good supervised algorithm which is also capable of maintaining and exceeding baseline scores, while using even lower amount of extracted features.

To highlight significant features from the entire set of initial ones, we decided to use a neural network model of auto-encoders. The experiments were carried out on models AE and VAE. Actually, these models with fully connected layers gradually squeezed the space of features to a smaller one. In the first case, we got the space right away, in the second we squeezed it to a probabilistic distribution, from which we then sampled the latent representation. This approach shows itself well on simple datasets, for example, mnist. We decided to test this approach to the data with which we worked. The conclusions that have been obtained show that the combination of the initial features and the addition of a nonlinear activation function greatly spoils the quality on datasets with a relatively small number of features. So for the dataset with the classification of the prices of mobile phones. Quality dropped by more than 50 percent in all experiments. The various possible sizes of the full-swap layers were checked and the best result was with compression from 20 to 12 signs and amounted to 0.57. Also, these methods were applied to the dataset with the classification of hotels and did not show decent results. Thus, a simple model with fully connected AE or VAE layers is not suitable for the feature extraction task and loses to standard methods. It is also worth noting that a possible increase in quality is not disfigured by the time spent on training. Also, to work with the loss function, we needed to scale all the data, which can also greatly spoil the quality.