Report DAKD
Improving the use of PCA

Jordi Piqué

DAKD

FIB • UPC

January 17, 2018

Abstract

Dimensionality reduction is a process that helps to reduce the size of datasets, the execution time of learning algorithms and allows to apply visualization techniques of the data. In this paper we explain how a feature selection algorithm, in our case a wrapper, can be used to select the best components from PCA. We have applied this method with different datasets and classifiers and we show that our method outperforms PCA. That means that the first c components of PCA are not always the best components for a classifier, in terms of accuracy.

Contents

1 Introduction

In machine learning there is a huge variety of datasets. Some of them are small, but others are really big. That is because, nowadays, we have increased dramatically the capacity of collecting data. The datasets have more instances and each instance has more features. In order to decrease the required hardware to store all these data and the time needed to process them, computer scientists have developed lots of different techniques that allow reducing the number of features per instance.

This paper aims to find a method for reducing the dimensionality of the data while preserving the best possible accuracy. We will use PCA as a feature extraction tool and then we will apply a feature selection technique in order to preserve only the best principal components. That will maximize the accuracy of a given predictor, while minimizing the number of features.

Normally, PCA is used by just taking the first c components. They are the components that explain more variance in the dataset, but they are not necessary the best components for a classifier. Janecek and Gansterer (2008) [1], in their paper, show that the variance explained by the components taken from PCA are not strongly correlated with the accuracy of a classifier.

Ekenel and Sankur (2004) [2] use the same idea in the face-recognition problem. They apply a feature selection process from the features generated by ICA and PCA.

In section 2 we will explain PCA and how is commonly used. In section 3 we will explain which are the main methods for selecting features and how they work. In section 4 it is described the implementation of the technique that allows to improve the use of PCA and the evaluation of this technique. In section 5 we present the empirical results we have found and we analyze them. Finally, in section 6 there are the conclusions of this project and the future work.

2 PCA and its practical use

PCA can be considered as a **feature extraction** tool. If we have n instances and d features per instance, with $n > d$, then PCA returns a set of d orthogonal vectors. Each instance can be represented by a linear combination of these d vectors. The first vector u_1 is the one that explains more variance accross all the dataset. The second vector u_2 is the one that is in an orthogonal subspace with respect to the first one and explains more variance accross all the dataset, and so on. u_1 can be expressed as:

$$u_1 \quad s.t. \quad ||u_1|| = 1 \quad and \quad MAX\Big[||X_{nxd} \cdot u_1||\Big] \tag{1}$$

However, there exist an equivalent way to explain PCA that will illustrate better the idea behind it. Lets first introduce the "length" of an instance. The "length" of an instance is the Euclidean distance of its feature vector ($length = ||x_i||$). From this point, instead of using the length of an instance $||x_i||$, we will use the square of the length $||x_i||^2$. We can imagine the "length" of all the dataset as the sum of the lentgh of each instance.

$$totalLength = \sum_{i=0}^{n} ||x_i||^2 \tag{2}$$

$$||x_i||^2 = \sum_{j=0}^{d} x_{i,j}^2 \tag{3}$$

Another way to understand PCA is to think that we want to find the 1D subspace that contains the maximum amount of all the length of the dataset. A vector u_1 with $||u_1|| = 1$ defines a 1D subspace. When we do the dot product between u_1 and an instance x_i we have the projection of x_i to that subspace. Then, the length of each instance can be decomposed as the lenght of the projection of x_i to u_1 plus the length of the orthogonal part of x_i relative to u_1

$$||x_i||^2 = dot(x_i, u_1)^2 + (x_i - u_1 \cdot dot(x_i, u_1))^2 \tag{4}$$

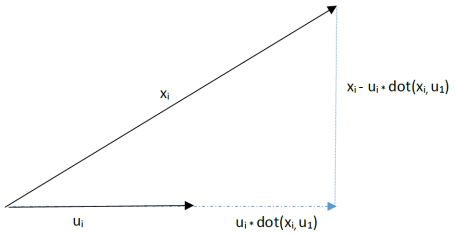

Figure 1:

The total length of the dataset contained in the 1D subspace defined by u_1 is:

$$\sum_{i=0}^{n} dot(x_i, u_1)^2 = (X \cdot u_1)^t \cdot (X \cdot u_1) = u_1^t \cdot (X^t \cdot X) \cdot u_1 \tag{5}$$

If we notice that $X^t \cdot X$ is the covariance matrix Σ, we can rewrite the equation 5 as:

$$u_1^t \cdot \Sigma \cdot u_1 \tag{6}$$

We need to remember that the goal is to maximize equation 1. We know that is the same as maximizing equation 6. The traditional way to solve this maximization problem is using Lagrange multipliers. The objective function is $u_1^t \cdot \Sigma \cdot u_1$ and the restriction is $||u_1|| = 1$.

Another way to deduce which is the vector that maximizes equation 6 is to think of the eigen-vector decomposition of u_1. Because Σ is symmetric, the values of its eigen-vectors are R and its eigen-vector are orthogonal. u_1 can always be decomposed by the eigen-vectors $(a_1, a_2, ...)$ of Σ. Now, we will suppose that $||a_i|| = 1$

$$u_1 = \sum_{j=1}^{d} k_j \cdot a_j \tag{7}$$

$$u_1^t \cdot \Sigma \cdot u_1 = \Big[\sum_{j=1}^{d} k_j \cdot a_j\Big]^t \cdot \Sigma \cdot \Big[\sum_{j=1}^{d} k_j \cdot a_j\Big] = \Big[\sum_{j=1}^{d} k_j \cdot a_j\Big]^t \cdot \Big[\sum_{j=1}^{d} \lambda_j \cdot k_j \cdot a_j\Big] = \sum_{j=1}^{d} \lambda_j \cdot (k_j \cdot a_j)^2 \tag{8}$$

With the restriction $||u_1|| = 1$, the last equation is trivially maximized when $k_1 = 1$ and all the other $k_i = 0$. That means that $u_1 = a_1$. Then, the vector that is orthogonal to u_1 and maximizes equation 8 is $u_2 = a_2$, and so on.

One of the uses of PCA is to extract relevant features from a dataset. As said in the introduction, the typical way to use PCA in a classification problem is to get the first c components and then applying a classifier on this new features. However, because PCA doesn't take into account the labels of the data, the first c features are not always the best subset of components of size c.

3 Methods for selecting features

In a dataset not all the features have the same relevance for a classification problem. Usually, much of the information is within a reduced subset of features. There are algorithms that try to find out which is the "best" subset of features in a dataset. There are two main families of this type of algorithms: **filters** and **wrappers**. A good comparison between some wrapper and filter techinques can be found in the paper publish by Kohavi and H.John (1997) [4].

3.1 Filters

Filters select features independently of the classifier algorithm that will be applied later. They are based only on general characteristics like the correlation with the variable to predict. Filters run quite fast and they are robust to overfitting.

However, filter methods tend to select redundant variables because they do not consider an eventual correlation between variables, so they are mainly used as a pre-process method.

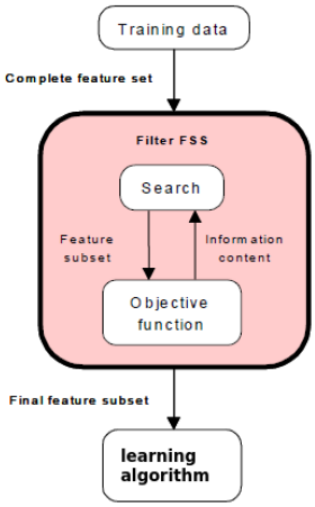

Figure 2: Diagram showing how a filter works. (Image from the slides of the DAKD course - prof. Lluís Belanche)

3.2 Wrappers

Wrappers use the same learning algorithm for the objective function and for the classification or regression task. They have the ability to detect interactions between features and they can

remove those features which are correlated with others.

The main problem is that they tend to overfit when the number of observations is not large enough and they are slower than filters.

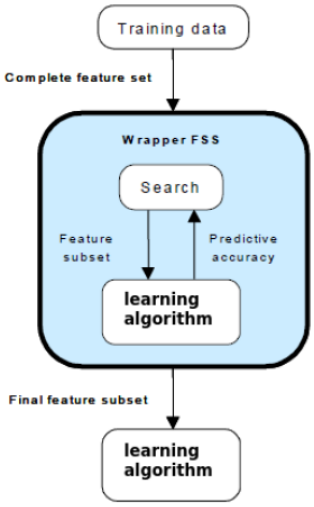

Figure 3: Diagram showing how a wrapper works. (Image from the slides of the DAKD course - prof. Lluís Belanche)

3.3 Feature subset search

In the current literature we can find lots of different methods to select the best subset of features from a dataset. For example, **sequential forward selection** starts from the empty set and at each iteration it adds the best feature to the subset. **Sequential backward selection** is similar to the last method, but it goes in the opposite direction. It starts from the full set of features and, at each iteration, it removes the worst.

One approach that combines characteristics from forward and backward selection is **bidirectional search**. Iteratively, it selects the best feature and it removes the worst feature. It cannot remove features that have already been selected and it cannot select features that have already been removed.

4 Implementation and evaluation

As said in section 1, we conjecture that we can improve the performance of PCA in a classification problem if we select the "best" components. In order to prove it empirically, we have used the **PCA** method from *scikit-learn* and we have implemented **bidirectional search** in order to find the best components. The goal is to extract $2 * c$ features and then select the best subset of size c.

We first apply PCA to all the dataset in order to know which are the first $2 * c$ components. Then, we shuffle the dataset and we divide it in the following way:

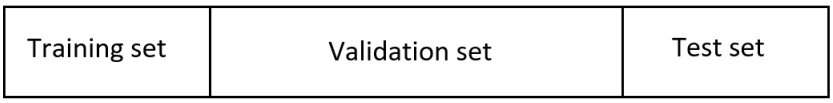

Figure 4: Representation of how the dataset is split

The first 25% of the elements (**training set**) will be used to train the learning algorithm inside the wrapper. The second 50% of the elements (**validation set**) will be used to test the accuracy of a given subset of features inside the wrapper. Finally, the last 25% will be used to compare the accuracy of the classifier using the first c components vs using the best subset of c components.

As said before, the training set and the validation set go to the wrapper in order to select the best features. The wrapper selects the best c features from the original set of $2 * c$ features.

After having the best c features we compare the accuracy of the learning algorithm using the first c components and using the best subset. In both cases, in this last part, the classifier is trained with the training set and validation set and is tested with the test set.

One of the desired characteristics of the method presented in this paper is it can be used with any dataset without worrying about what it represents. To prove that, we have tested it against 4 different datasets:

- **Mnist** A dataset of gray-scale images of hand-written digits (70000 inst x 784 feat)
- **Fashion Mnist** A dataset of grayscale images of 10 fashion categories (70000 inst x 784 feat)
- **Gas Sensor Array Drift** A dataset with the data of 16 chemical sensors utilized in simulations for drift compensation in a discrimination task of 6 gases (13910 inst x 128 feat)

- **Human Activity Recognition Using Smartphones** A dataset with the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors (10299 inst x 561 feat)

We also want to prove that this method is valid for a wide range of classifiers. That is why we have tested each of the last datasets with these classifiers:

- Logistic Regression
- Naive Bayes
- K-Neighbours
- Decision Tree
- Random Forest

In order to see the behaviour of the method for different sizes of subsets we have executed it from $c = 2$ to $c = 11$.

```
foreach classifier in setClassifiers {
    foreach dataset in setDatasets {
        for c = 2 to 11 {
            // Get 2*c principal componenets
            pca.fit(dataset, 2*c)
            dataset.X = pca.transform(dataset.X)

            // Suffle and divide dataset
            dataset.shuffle()
            (train, validation, test) = dataset.split()

            // Get best features
            bestFeatures = bidirectionalSearch(train, validation, 2*c, classifier)

            // Get the accuracy using the first c features
            X1 = selectFeatures(train + validation, [1..c])
            classifier.fit(X1)
            predicted1 = classifier.predict(test.X)
            accuracy1 = getAccuracy(test.y, predicted1)

            // Get the acccuracy using the best c features
            X2 = selectFeatures(train + validation, bestFeatures)
            classifier.fit(X2)
            predicted2 = classifier.predict(test.X)
            accuracy2 = getAccuracy(test.y, predicted1)
        }
    }
}
```

5 Results

Our results show that the method presented in this paper improves the accuracy of a classifier. The mean improvement is 2.3 points. However, this improvement varies across the number of components used, the dataset and the classifier. That's why, in the following paragraphs, we will analyze how is the behaviour of the improvement based on the number of components, the classifier and the dataset

When the number of features is low it is when selecting the best ones gives an important improvement in accuracy. When the number of features increase, then, our method is less efficient. However, it continues to improve a little bit the accuracy.

The following table shows the improvement in accuracy for different values of $2 * c$. The improvement is measured as the difference between the accuracy using the best subset and the accuracy using the first c components. The accuracy goes from 0 to 1.

nComp (2*c)	Improvement
4	0.051
6	0.055
8	0.035
10	0.018
12	0.011
14	0.013
16	0.013
18	0.013
20	0.007
22	0.015

Table 1: Mean improvement in accuracy for different values of 2*c

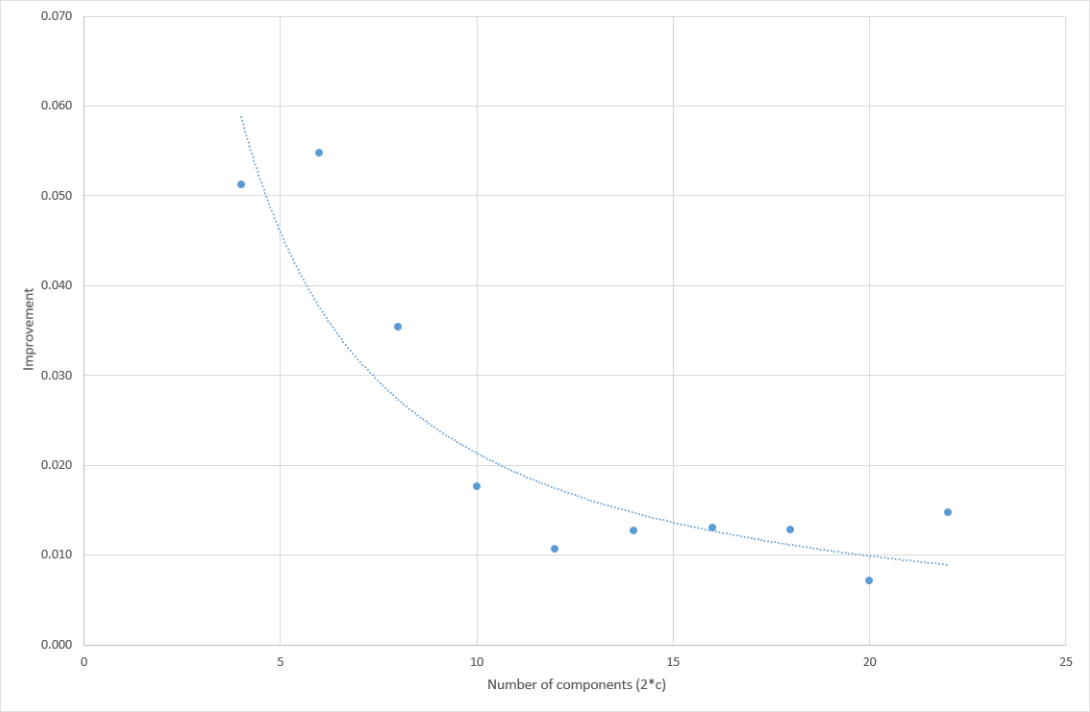

Figure 5: Plot and tendency line from table 1

One reason could be that with few features it doesn't overfit, but when we increase the number of features it starts overfitting. That means that the best subset of components found in the wrapper works specially well only with the validation set but it fails with the test set because it doesn't generalize well.

Another reason (and this is probably the most important) is that most of the information is within the first 8 principal components. It is true that the amount of information of the first 8 components is not always sorted in decreasing order. For example, sometimes the 4th component has more information than the 3rd component. That's why the algorithm works well for a few number of components. But when the number of components increase, the information that add the last components is low, so it is not very relevant if we take the 15th component instead of the 10th component.

We also can see that the efficiency of our method varies across the classifier.

Classifier	Improvement
decisionTree	0.014
kNeighbors	0.018
logisticRegression	0.035
naiveBayes	0.032
randomForest	0.016

Table 2: Mean improvement in accuracy for the classifiers used

In order to probe statistically that the improvement changes across the different classifiers, we have performed an Anova test. The p-value found is $pvalue = 0.012$. Taking a significance level of 5%, we can reject the null hypothesis and say that the classifier is a factor that affects the improvement.

Another important aspect of our results is that we can see how each classifier takes advantage of different features. The way each classifier can interpret and use the information inside the features is different and we can notice that in our results. In the complete table in the Appendix we can see which is the best subset of features for each classifier and dataset.

As we have said before, the improvement also varies across the datasets.

Dataset	Improvement
drift	0.040
fashionMnist	0.012
har	0.023
mnist	0.017

Table 3: Mean improvement in accuracy for the datasets used

Here, we have also done an Anova test to probe statistically that the improvement changes across the different datasets. The p-value found is $pvalue = 0.0004$. Taking a significance level of 5%, we can reject the null hypothesis and say that the dataset is a factor that affects the improvement.

6 Conclusions and future work

In this paper we have seen that it is possible to select, from PCA, a subset of components of size c that performs better than the first c components in a classification task. That proves that the variance explained by a subset of components is not always completely correlated with the accuracy of a classifier that uses these components.

As we have said in section 3 wrappers do a good job selecting the best subset of features, but paying the price of consuming a considerable computational time. This last statement have also been proved. It is true that the execution time has not been discussed in our results, because our main focus was on the improve in accuracy. However, in real applications it is an important factor to take into account. In our case, we spent 2 hours using all cores of an Intel i7 processor for doing all the executions that appear in the Appendix.

One of the most important aspects is that it is a general purpose method. We have seen in this paper that it works with different classifiers and different datasets. However, some classifiers and some datasets can take more advantge of this technique than others. An extension of this work could be anticipating the improve in accuracy of this method with different classifiers and datasets. That could help to avoid using this technique in datasets and classifiers where the improve is nearly negligible. That is why more execution using more datasets are needed. They will help to know more precisely the expected improve for each classifier and they will validate (or reject) the idea that this technique is a general purpose method that can work with any dataset or classifier.

Another extension could be determining more accurately the expected improve in accuracy based on the number of components selected. We have seen that the improve in accuracy decreases when the number of components increases. If we could say in advance which is the expected improve in accuracy for a given number of components, we could choose those problems which will benefit more of our method. Generally, these problems will be the ones that need a lot of instances with a few number of features (for storage reasons or computational time reasons).

References

[1] MAndreas Janecek, Wilfried Gansterer, Michael Demel, Gerhard Ecker. *On the Relationship Between Feature Selection and Classification Accuracy*. Proceedings of Machine Learning Research, 2008.

[2] H.K.Ekenel and B.Sankur. *Feature selection in the independent component subspace for face recognition*. Elsevier, Pattern Recognition Letters, Volume 25, Issue 12, Pages 1377-1388, 2004.

[3] Wikipedia: Principal Component Analysis, `https://en.wikipedia.org/wiki/Principal_component_analysis`

[4] Ron Kohavi and George H.John. *Wrappers for feature subset selection*. Elsevier, Artificial Intelligence, Volume 97, Issues 1–2, Pages 273-324, 1997.

[5] Lluis Belanche. *Slides for the DAKD course*. FIB, 2017.

7 Appendix

classifier	dataset	nComp	bestFeat	halfPca	bestPca	difference
decisionTree	drift	4	[2,3]	0.718	0.774	0.056
kNeighbors	drift	4	[1,3]	0.738	0.786	0.047
logisticRegression	drift	4	[1,3]	0.400	0.536	0.136
naiveBayes	drift	4	[1,3]	0.527	0.602	0.075
randomForest	drift	4	[2,3]	0.743	0.805	0.063
decisionTree	fashionMnist	4	[0,1]	0.461	0.460	-0.001
kNeighbors	fashionMnist	4	[0,1]	0.511	0.511	0.000
logisticRegression	fashionMnist	4	[0,1]	0.477	0.477	0.000
naiveBayes	fashionMnist	4	[0,1]	0.504	0.504	0.000
randomForest	fashionMnist	4	[0,1]	0.498	0.496	-0.003
decisionTree	har	4	[0,2]	0.553	0.678	0.125
kNeighbors	har	4	[0,2]	0.596	0.721	0.125
logisticRegression	har	4	[0,2]	0.573	0.705	0.132
naiveBayes	har	4	[0,2]	0.570	0.716	0.146
randomForest	har	4	[0,2]	0.583	0.706	0.123
decisionTree	mnist	4	[0,1]	0.376	0.377	0.001
kNeighbors	mnist	4	[0,1]	0.429	0.429	0.000
logisticRegression	mnist	4	[0,1]	0.438	0.438	0.000
naiveBayes	mnist	4	[0,1]	0.439	0.439	0.000
randomForest	mnist	4	[0,1]	0.409	0.408	-0.001
decisionTree	drift	6	[0,3,4]	0.841	0.924	0.083
kNeighbors	drift	6	[2,3,4]	0.857	0.927	0.070
logisticRegression	drift	6	[1,3,4]	0.449	0.640	0.192
naiveBayes	drift	6	[0,3,4]	0.532	0.746	0.214
randomForest	drift	6	[0,3,4]	0.875	0.939	0.064
decisionTree	fashionMnist	6	[1,2,5]	0.554	0.558	0.004
kNeighbors	fashionMnist	6	[0,1,5]	0.619	0.632	0.012
logisticRegression	fashionMnist	6	[0,1,4]	0.580	0.588	0.008
naiveBayes	fashionMnist	6	[0,1,5]	0.565	0.594	0.029
randomForest	fashionMnist	6	[1,2,5]	0.610	0.626	0.016
decisionTree	har	6	[0,2,5]	0.716	0.731	0.015
kNeighbors	har	6	[0,2,5]	0.751	0.775	0.024
logisticRegression	har	6	[0,2,5]	0.770	0.754	-0.016
naiveBayes	har	6	[0,2,5]	0.745	0.778	0.034
randomForest	har	6	[0,2,5]	0.737	0.776	0.039
decisionTree	mnist	6	[0,1,5]	0.432	0.475	0.043
kNeighbors	mnist	6	[0,1,3]	0.486	0.537	0.051
logisticRegression	mnist	6	[0,1,3]	0.446	0.524	0.079
naiveBayes	mnist	6	[0,1,3]	0.462	0.538	0.076
randomForest	mnist	6	[0,1,5]	0.473	0.529	0.056
decisionTree	drift	8	[0,3,4,6]	0.920	0.954	0.034
kNeighbors	drift	8	[2,3,4,6]	0.917	0.949	0.032
logisticRegression	drift	8	[3,4,5,6]	0.594	0.704	0.110
naiveBayes	drift	8	[1,2,3,4]	0.655	0.748	0.093
randomForest	drift	8	[0,3,4,6]	0.935	0.972	0.037

decisionTree	fashionMnist	8	[0,1,2,5]	0.636	0.630	-0.006
kNeighbors	fashionMnist	8	[1,2,4,5]	0.699	0.701	0.003
logisticRegression	fashionMnist	8	[0,1,4,5]	0.651	0.655	0.004
naiveBayes	fashionMnist	8	[1,2,4,5]	0.619	0.651	0.033
randomForest	fashionMnist	8	[0,1,4,5]	0.695	0.692	-0.002
decisionTree	har	8	[0,2,4,6]	0.740	0.742	0.002
kNeighbors	har	8	[0,2,5,7]	0.772	0.816	0.044
logisticRegression	har	8	[0,1,2,4]	0.775	0.765	-0.010
naiveBayes	har	8	[0,2,5,6]	0.732	0.789	0.057
randomForest	har	8	[0,2,4,5]	0.780	0.819	0.040
decisionTree	mnist	8	[0,1,3,5]	0.546	0.583	0.036
kNeighbors	mnist	8	[0,1,3,5]	0.616	0.652	0.036
logisticRegression	mnist	8	[0,1,3,4]	0.560	0.621	0.061
naiveBayes	mnist	8	[0,1,3,5]	0.560	0.627	0.068
randomForest	mnist	8	[0,1,3,5]	0.620	0.655	0.035
decisionTree	drift	10	[0,1,3,4,6]	0.958	0.965	0.007
kNeighbors	drift	10	[1,2,3,4,6]	0.939	0.955	0.016
logisticRegression	drift	10	[1,3,4,5,6]	0.699	0.761	0.062
naiveBayes	drift	10	[0,1,2,3,4]	0.823	0.823	0.000
randomForest	drift	10	[0,3,4,8,9]	0.971	0.980	0.009
decisionTree	fashionMnist	10	[0,1,2,3,9]	0.659	0.675	0.016
kNeighbors	fashionMnist	10	[0,1,2,4,5]	0.728	0.735	0.007
logisticRegression	fashionMnist	10	[0,1,2,5,7]	0.684	0.686	0.002
naiveBayes	fashionMnist	10	[0,1,2,4,5]	0.653	0.668	0.016
randomForest	fashionMnist	10	[0,1,2,4,5]	0.725	0.735	0.010
decisionTree	har	10	[0,2,4,6,9]	0.775	0.774	-0.001
kNeighbors	har	10	[0,2,4,6,7]	0.809	0.846	0.037
logisticRegression	har	10	[0,1,2,5,8]	0.814	0.825	0.011
naiveBayes	har	10	[0,2,4,5,7]	0.782	0.816	0.035
randomForest	har	10	[0,2,4,5,7]	0.807	0.833	0.026
decisionTree	mnist	10	[0,1,3,4,5]	0.648	0.667	0.019
kNeighbors	mnist	10	[0,1,2,3,5]	0.728	0.743	0.015
logisticRegression	mnist	10	[0,1,3,4,5]	0.641	0.673	0.032
naiveBayes	mnist	10	[0,1,3,4,6]	0.651	0.667	0.016
randomForest	mnist	10	[0,1,2,3,5]	0.716	0.735	0.019
decisionTree	drift	12	[0,2,3,4,5,6]	0.960	0.961	0.001
kNeighbors	drift	12	[1,2,3,4,5,6]	0.952	0.967	0.015
logisticRegression	drift	12	[0,3,4,6,8,10]	0.708	0.803	0.095
naiveBayes	drift	12	[0,2,3,4,6,10]	0.818	0.809	-0.009
randomForest	drift	12	[0,1,3,4,6,9]	0.973	0.983	0.011
decisionTree	fashionMnist	12	[0,1,2,4,5,9]	0.680	0.704	0.024
kNeighbors	fashionMnist	12	[1,2,3,5,9,10]	0.753	0.770	0.017
logisticRegression	fashionMnist	12	[0,1,2,4,5,7]	0.696	0.710	0.014
naiveBayes	fashionMnist	12	[0,1,4,5,7,8]	0.677	0.689	0.012
randomForest	fashionMnist	12	[0,1,2,4,5,9]	0.747	0.760	0.013
decisionTree	har	12	[0,2,3,5,7,8]	0.789	0.790	0.001
kNeighbors	har	12	[0,2,4,5,7,8]	0.850	0.855	0.005
logisticRegression	har	12	[0,1,2,5,6,9]	0.819	0.832	0.013
naiveBayes	har	12	[0,2,5,6,7,8]	0.817	0.820	0.003

randomForest	har	12	[0,2,4,5,7,8]	0.845	0.852	0.007
decisionTree	mnist	12	[0,1,2,3,4,5]	0.723	0.725	0.002
kNeighbors	mnist	12	[0,1,2,3,4,5]	0.817	0.817	0.000
logisticRegression	mnist	12	[0,1,2,3,4,5]	0.705	0.705	0.000
naiveBayes	mnist	12	[0,1,3,4,5,6]	0.708	0.709	0.001
randomForest	mnist	12	[0,1,2,3,5,7]	0.795	0.783	-0.011
decisionTree	drift	14	[0,1,3,4,5,6,9]	0.963	0.968	0.004
kNeighbors	drift	14	[2,3,4,5,6,7,8]	0.957	0.963	0.006
logisticRegression	drift	14	[1,2,3,4,5,6,13]	0.776	0.870	0.095
naiveBayes	drift	14	[2,3,4,5,6,10,13]	0.814	0.844	0.030
randomForest	drift	14	[0,2,3,4,6,8,9]	0.981	0.984	0.003
decisionTree	fashionMnist	14	[0,1,2,3,4,5,8]	0.697	0.704	0.007
kNeighbors	fashionMnist	14	[1,2,3,5,7,8,10]	0.769	0.792	0.023
logisticRegression	fashionMnist	14	[0,1,2,4,5,9,13]	0.693	0.720	0.027
naiveBayes	fashionMnist	14	[0,1,4,5,7,8,10]	0.672	0.695	0.024
randomForest	fashionMnist	14	[1,2,3,5,6,9,10]	0.759	0.774	0.015
decisionTree	har	14	[0,2,3,4,7,8,12]	0.801	0.788	-0.012
kNeighbors	har	14	[0,2,4,5,6,7,9]	0.862	0.860	-0.002
logisticRegression	har	14	[0,1,2,3,4,5,6]	0.852	0.852	0.000
naiveBayes	har	14	[0,2,4,5,7,8,9]	0.816	0.834	0.019
randomForest	har	14	[0,2,3,4,5,7,10]	0.850	0.840	-0.010
decisionTree	mnist	14	[0,1,2,3,4,5,7]	0.751	0.759	0.008
kNeighbors	mnist	14	[0,1,2,3,4,5,11]	0.859	0.864	0.005
logisticRegression	mnist	14	[0,1,2,3,4,5,7]	0.727	0.736	0.009
naiveBayes	mnist	14	[0,1,3,4,5,6,7]	0.729	0.735	0.007
randomForest	mnist	14	[0,1,2,3,4,5,7]	0.828	0.827	-0.001
decisionTree	drift	16	[0,1,3,4,6,8,9,13]	0.965	0.965	0.000
kNeighbors	drift	16	[1,2,3,4,5,6,8,10]	0.952	0.960	0.008
logisticRegression	drift	16	[0,1,2,3,4,5,8,13]	0.776	0.861	0.085
naiveBayes	drift	16	[0,2,3,4,5,6,10,13]	0.807	0.840	0.033
randomForest	drift	16	[0,1,3,4,5,6,9,10]	0.983	0.987	0.004
decisionTree	fashionMnist	16	[0,1,2,4,5,9,10,14]	0.710	0.719	0.010
kNeighbors	fashionMnist	16	[1,2,4,5,6,8,9,14]	0.778	0.793	0.015
logisticRegression	fashionMnist	16	[0,1,2,4,5,7,8,14]	0.712	0.736	0.024
naiveBayes	fashionMnist	16	[0,1,4,5,7,8,14,15]	0.698	0.712	0.014
randomForest	fashionMnist	16	[0,1,2,4,5,8,11,14]	0.767	0.782	0.015
decisionTree	har	16	[0,2,4,5,7,8,9,12]	0.800	0.806	0.006
kNeighbors	har	16	[0,2,3,4,5,6,7,12]	0.867	0.878	0.011
logisticRegression	har	16	[0,1,2,3,4,5,6,8]	0.856	0.858	0.003
naiveBayes	har	16	[0,2,4,5,6,7,8,9]	0.827	0.841	0.013
randomForest	har	16	[0,2,4,5,6,7,8,15]	0.844	0.851	0.008
decisionTree	mnist	16	[0,1,2,3,4,5,7,11]	0.776	0.780	0.005
kNeighbors	mnist	16	[0,1,2,3,4,5,6,11]	0.887	0.888	0.001
logisticRegression	mnist	16	[0,1,2,3,4,5,6,7]	0.762	0.762	0.000
naiveBayes	mnist	16	[0,1,2,3,4,5,7,11]	0.759	0.761	0.003
randomForest	mnist	16	[0,1,2,3,4,5,6,7]	0.848	0.851	0.003
decisionTree	drift	18	[0,1,2,3,4,6,8,12,13]	0.974	0.970	-0.004
kNeighbors	drift	18	[2,3,4,5,6,7,8,10,11]	0.955	0.963	0.008
logisticRegression	drift	18	[0,1,2,3,4,6,9,13,14]	0.824	0.883	0.059

naiveBayes	drift	18	[0,2,3,4,5,6,10,16,17]	0.786	0.823	0.037
randomForest	drift	18	[0,2,3,4,6,7,8,9,12]	0.985	0.986	0.001
decisionTree	fashionMnist	18	[0,1,2,4,5,6,7,8,14]	0.716	0.724	0.007
kNeighbors	fashionMnist	18	[1,2,4,5,7,9,10,12,17]	0.794	0.804	0.010
logisticRegression	fashionMnist	18	[0,1,2,3,4,5,7,8,17]	0.726	0.743	0.017
naiveBayes	fashionMnist	18	[0,1,2,4,5,7,8,15,17]	0.704	0.725	0.021
randomForest	fashionMnist	18	[0,1,2,4,5,6,9,16,17]	0.783	0.798	0.015
decisionTree	har	18	[0,1,2,3,4,5,6,7,15]	0.813	0.823	0.010
kNeighbors	har	18	[0,2,3,4,5,6,7,12,15]	0.868	0.878	0.010
logisticRegression	har	18	[0,1,2,3,4,5,6,9,15]	0.860	0.874	0.014
naiveBayes	har	18	[0,2,4,5,7,8,9,14,17]	0.844	0.844	0.000
randomForest	har	18	[0,2,4,5,7,8,9,12,13]	0.862	0.854	-0.007
decisionTree	mnist	18	[0,1,2,3,4,5,6,7,11]	0.782	0.792	0.010
kNeighbors	mnist	18	[0,1,2,3,4,5,6,7,11]	0.899	0.910	0.010
logisticRegression	mnist	18	[0,1,2,3,4,5,6,7,11]	0.764	0.783	0.019
naiveBayes	mnist	18	[0,1,2,3,4,5,6,7,15]	0.766	0.778	0.012
randomForest	mnist	18	[0,1,2,3,4,5,6,7,11]	0.857	0.865	0.008
decisionTree	drift	20	[0,1,2,3,4,6,8,11,12,14]	0.965	0.960	-0.005
kNeighbors	drift	20	[2,3,4,5,6,7,8,9,10,16]	0.956	0.969	0.014
logisticRegression	drift	20	[0,1,2,3,4,5,6,9,13,16]	0.881	0.859	-0.022
naiveBayes	drift	20	[0,2,3,4,5,6,8,10,16,17]	0.771	0.799	0.028
randomForest	drift	20	[0,1,3,4,6,8,9,13,16,18]	0.988	0.984	-0.004
decisionTree	fashionMnist	20	[0,1,2,4,5,8,9,10,11,14]	0.725	0.741	0.016
kNeighbors	fashionMnist	20	[1,2,4,5,6,9,10,14,16,17]	0.799	0.815	0.015
logisticRegression	fashionMnist	20	[0,1,2,4,5,7,8,13,14,17]	0.732	0.753	0.020
naiveBayes	fashionMnist	20	[0,1,2,4,5,7,8,14,15,17]	0.703	0.723	0.019
randomForest	fashionMnist	20	[0,1,2,4,5,6,9,10,16,17]	0.787	0.799	0.011
decisionTree	har	20	[0,1,2,4,5,6,7,9,12,15]	0.816	0.808	-0.009
kNeighbors	har	20	[0,2,4,5,6,7,8,9,12,13]	0.894	0.896	0.003
logisticRegression	har	20	[0,1,2,3,4,5,6,8,9,15]	0.872	0.878	0.006
naiveBayes	har	20	[0,2,4,5,7,8,9,10,14,15]	0.832	0.838	0.006
randomForest	har	20	[0,2,3,4,5,6,7,8,9,12]	0.863	0.861	-0.002
decisionTree	mnist	20	[0,1,2,3,4,5,6,7,11,13]	0.796	0.802	0.006
kNeighbors	mnist	20	[0,1,2,3,4,5,6,7,9,12]	0.917	0.919	0.002
logisticRegression	mnist	20	[0,1,2,3,4,5,6,7,9,15]	0.778	0.792	0.014
naiveBayes	mnist	20	[0,1,2,3,4,5,6,7,15,19]	0.766	0.787	0.021
randomForest	mnist	20	[0,1,2,3,4,5,6,7,9,15]	0.873	0.876	0.003
decisionTree	drift	22	[0,1,2,3,4,5,6,9,13,14,17]	0.965	0.974	0.009
kNeighbors	drift	22	[2,3,4,5,6,7,8,9,10,11,16]	0.957	0.959	0.002
logisticRegression	drift	22	[0,1,2,3,4,6,8,9,10,11,13]	0.851	0.895	0.044
naiveBayes	drift	22	[0,2,3,4,5,6,10,13,14,16,17]	0.783	0.831	0.048
randomForest	drift	22	[0,1,3,4,5,6,7,9,11,18,19]	0.986	0.986	0.000
decisionTree	fashionMnist	22	[0,1,2,4,5,8,9,11,13,14,17]	0.734	0.732	-0.002
kNeighbors	fashionMnist	22	[1,2,4,5,8,9,10,11,16,17,20]	0.806	0.823	0.017
logisticRegression	fashionMnist	22	[0,1,2,3,4,5,7,8,12,17,20]	0.748	0.767	0.019
naiveBayes	fashionMnist	22	[0,1,2,4,5,7,8,14,15,17,20]	0.711	0.733	0.022
randomForest	fashionMnist	22	[1,2,3,5,6,8,9,10,11,14,16]	0.801	0.799	-0.002
decisionTree	har	22	[0,2,3,4,5,6,7,8,9,10,21]	0.807	0.821	0.014
kNeighbors	har	22	[0,2,3,4,5,7,8,9,10,12,21]	0.887	0.893	0.006

logisticRegression	har	22	[0,1,2,3,4,5,6,8,9,15,21]	0.881	0.889	0.008
naiveBayes	har	22	[0,2,3,5,7,8,9,10,14,15,21]	0.844	0.852	0.008
randomForest	har	22	[0,1,2,4,5,6,7,8,9,12,21]	0.867	0.879	0.011
decisionTree	mnist	22	[0,1,2,3,4,5,6,7,11,12,15]	0.790	0.806	0.016
kNeighbors	mnist	22	[0,1,2,3,4,5,6,7,11,12,15]	0.922	0.930	0.008
logisticRegression	mnist	22	[0,1,2,3,4,5,6,7,11,15,19]	0.778	0.807	0.029
naiveBayes	mnist	22	[0,1,2,3,4,5,6,7,11,15,19]	0.773	0.801	0.028
randomForest	mnist	22	[0,1,2,3,4,5,6,7,9,11,15]	0.874	0.884	0.010