Human Activity Recognition using Smartphone Sensors

Jessica Moore[a] and Binghai Ling[b]

(Dated: 16 December 2016)

I. INTRODUCTION

Smartphones are quickly becoming a ubiquitous part of life in the Western world. Their embedded sensors have the ability to record a significant amount of data about peoples' movement. Many smartphone applications already use that data to estimate basic fitness statistics, such as daily step count. However, the simplistic metrics that these applications produce are a poor method of assessing a person's activity level.[1] As a result, many people have turned to wearable devices to track their fitness activities. However, smartphone technology is improving, and so are techniques for using the data that smartphones gather. If a phone could accurately assess a person's daily activities, smartphone applications could easily replace wearable fitness devices as a method of fitness tracking.

In this paper, we test different machine learning methods to predict the activity that a person is undertaking, based on signals from that person's cell phone. The input to each of the algorithms is a set of summaries of accelerometer and gyroscope signals and the output of each algorithm is (a prediction of) the activity being performed when those signals were recorded. While maximizing the accuracy of our predictions is the foremost goal, we also pay attention to the number of features each method requires to obtain high accuracy. Data requirements are of particular importance for smartphones, since their capacity for transmitting, receiving, and storing data is limited. We would consider a method successful if it either (a) obtains better accuracy than previous methods or (b) obtains accuracy similar to that of previous methods, but uses fewer features.

First, we discuss the data set that we employ and provide a review of the previous literature on Human Activity Recognition. We then explain the feature extraction methods we use. Next, we present the classification algorithms we employ and describe our implementation of them. Then, we review our results comparing the error rates of different feature selection/classification algorithm combinations. We also conduct error analysis of the most successful methods and suggest ways to build from them. We conclude with a general discussion of our results.

II. DATA

The data set on which we test our methods is courtesy of the UCI Machine Learning Repository.[2] It contains summaries of accelerometer and gyroscope readings from waist-worn cell phones by 30 volunteers while performing different activities. The data contain 10,299 observations with about 300-400 observations per subject, and a roughly even distribution of observations across activities (even at the subject-level). There are, altogether, 561 features in the data set; these are primarily based on the 3-axial linear acceleration and 3-axial angular velocity. The response variable is the activity that an individual was performing when the readings were obtained. Categories are Laying, Sitting, Standing, Walking, Walking Downstairs (WD), and Walking Upstairs (WU).

Each observation of the data set contains information regarding one sample window – a segment of continuous time that a particular person spent doing a particular activity. The signals from those windows were processed using various filtering techniques.[a] The features in the data set are summaries of those processed time-domain signals (captured at a constant rate of 50Hz). For example, the feature set includes the mean, standard deviation, and skewness of the gravity acceleration signal.

In preparation for our modeling, we split the data as follows:

- **training**: 5,147 observations
- **validation**: 2,206 observations[b]
- **test**: 2,947 observations

For the sake of concision, we refer to the union of the training data and validation data as the 'train-val' data.

III. RELATED LITERATURE

Most successful Human-Activity Recognition (HAR) research has focused on the recognition of relatively simple activities (e.g., sitting or walking) rather than more complex activities (e.g., cooking or cleaning).[3] Very early work in the field used data collected from sensors placed on different locations of the body.[4] These data were straightforward to analyze, but inconvenient to collect. As a result, video recordings soon became an area of significant HAR research. Scientists have used camera recordings to recognize both full-body movements and the movement of smaller body parts, e.g., hand-gestures.[5] However, video recordings are only slightly less inconvenient to collect than data from sensors attached to the body.

As smartphones became more prevalent, they supplanted worn sensors and cameras as the method of choice for gathering large-scale human activity data.[6] The data set that we employ has been used previously to test different human activity recognition methods. Much of the previous work has been focused on the development of 'hardware-friendly' methods, which can be performed *on* a smartphone. The primary method resulting from this research

[a] Electronic mail: jmoore5@stanford.edu
[b] Electronic mail: binghai@stanford.edu

[a] For additional detail on filtering techniques, see the documentation ('features_info.txt') that accompanies the data set.
[b] The split into training data and validation data constitutes a 70-30 split of the initial set of 'training data' obtained from the UCI Machine Learning Repository.

is a hardware-friendly implementation of support vector machines (SVMs). It utilizes six SVM classifiers, in 'one vs. all' style (described later) to predict which observations fall into each activity category. This method obtains about 90% accuracy.[7] Absent concerns about hardware, it appears that the most effective method tried to-date is an unconstrained (i.e., non-hardware-friendly) version of the one vs. all SVM classifier, with a Gaussian kernel.[c] The method yields about 96% accuracy and serves as our performance benchmark.[8]

In recent HAR work, it seems that relatively little comparison to non-SVM methods, or even different SVM methods, has been done. Moreover, prior work with the data set that we utilize has employed all 561 features, rather than paring down the features to only those that enhance predictive accuracy. We attempt to remedy both of these gaps in the literature, in the hopes that we might obtain better algorithm performance.

IV. FEATURE ENGINEERING

Many feature engineering methods attempt to approximate the data, $X \in \mathbb{R}^{mxp}$, by finding $Z \in \mathbb{R}^{mxd}$ (with $d \ll p$), such that Z is still representative of X. This variety of dimensionality reduction can decrease computational requirements and enhance accuracy, by limiting variance. Given the large number of features in our data set, we believe variance control will be critical to obtaining an effective model. Below, we outline the two methods that we use to extract sets of features from our data.

A. Principal Components Analysis

Principal Components Analysis (PCA) attempts to identify the directions in feature space–called 'principal components' or 'PCs'–along which the data vary the most.[9] If each column of the data matrix (X) is centered and scaled, the PCs are simply the (normalized) eigenvectors of the empirical covariance matrix

$$\Sigma = \frac{1}{m}\Sigma_{i=1}^{m} x^{(i)} x^{(i)^T}$$

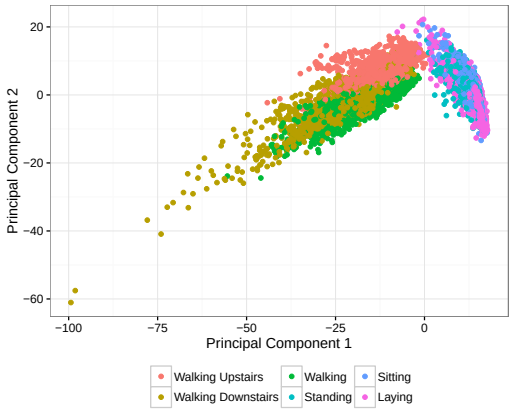
For a d-dimensional summary of the data that captures as much variance as possible, we represent the data in the basis of the first d PCs, $\{u_1, ..., u_d\}$.[10] That is, we compute the entries of Z as

$$Z_{ij} = u_j^T x^{(i)} \text{ for } j = 1, ..., d \text{ and } i = 1, ..., m$$

The entries are referred to as 'scores' or 'PC scores'. The j-th column of Z – the 'j-th PC score vector' – is the projection of the data onto the j-th PC.[9]

We obtain the PCs of the train-val data, after centering and scaling the original features. We then compute the PC-based features that we will use as model input by representing the train-val data and the test data in the basis of those PCs. In Figure 1, we graph the first and second PC scores of each observation in the data set, colored by activity. As is clear, even the first and second PC scores separate some activities from others.

FIG. 1. First and second principal component scores



B. Kernelized Principal Components Analysis

Kernelized Principal Components Analysis (kPCA) attempts to identify similar variance-maximizing directions, but does so after the data have been mapped into some alternate space, via a feature mapping ϕ.[11] By the same logic used for PCA, those directions (called 'kernelized principal components' or 'kPCs') are the eigenvectors of the matrix,

$$\tilde{\Sigma} = \frac{1}{m}\Sigma_{i=1}^{m} \phi(x^{(i)})\phi(x^{(i)})^T$$

with $\Sigma_{i=1}^{m}\phi(x_i) = 0$.[11] It can be shown that any kPC v can be written[12]

$$v = \Sigma_{i=1}^{m} \alpha_i \phi(x^{(i)})$$

where $\bar{\alpha} = [\alpha_i]_{i=1}^{m}$ is a (normalized) eigenvector of the matrix $K = [\phi(x^{(i)})^T \phi(x^{(j)})]_{i,j=1}^{m} = [K(x^{(i)}, x^{(j)})]_{i,j=1}^{m}$. Here, $K(x, y)$ is the kernel function associated with ϕ.[11]

Then, the score of an example x on the j-th kPC is[13]

$$v_j^T \phi(x) = \Sigma_{i=1}^{m} \alpha_i^j \phi(x^{(i)})^T \phi(x) = \Sigma_{i=1}^{m} \alpha_i^j K(x^{(i)}, x)$$

Thus, any kPC score can be computed without explicitly representing $\phi(x)$.[13]

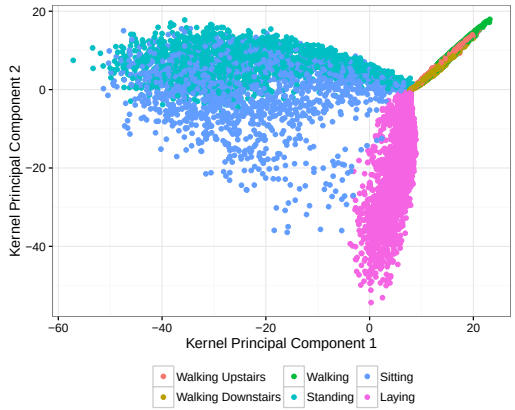
Because ϕ can map the data to a high dimensional space, kPCA can often capture nonlinear relationships among the data points, which are missed by the standard, linear version of PCA, described previously.[14] Prior authors had success with Guassian-kernel SVMs; thus, we suspect that there may be informative nonlinear structure to the data, which kPCA might capture.

We perform kPCA on the train-val data, using a Gaussian kernel,[10] $K(x, z) = \exp(-\gamma||x - z||_2^2)$.[d] We then find the kPC scores for the train-val data and the test data. Those score vectors are the kPC-based features that we will use as model input. In Figure 2, we graph the first and second kPC scores of each observation in the data set, colored by activity. As with the PC scores, these early kPC scores seem to effectively differentiate some activities from others.

[c] Unfortunately, the authors do not provide code to accompany their paper.

[d] We use $\gamma = 0.1$. For computational reasons, we do not explicitly tune γ. We suggest this as work to future authors.

FIG. 2. First and second kernelized principal component scores



V. CLASSIFICATION ALGORITHMS

We take the following steps for each feature engineering method and classification algorithm:

1. Build models $\{M_1^V, M_2^V, ..., M_N^V\}$ on the training data, using the first $\{1, 2, ..., N\}$ score vectors.
2. Use each of the models created in Step 1 to predict the classes of observations in the validation data; calculate each model's misclassification error.[e] Identify the model, M_q^V, with the lowest error.
3. Using the first q score vectors, build M^F on the train-val data.
4. Use M^F to predict the classes of observations in the test data and calculate the misclassification error.

Steps 1 and 2 identify the number of score vectors (q) expected to produce the lowest misclassification error. Steps 3 and 4 provide an unbiased estimate of the algorithm's error, when using the first q score vectors as features.

Below, we describe the classification algorithms to which we apply the above procedure.

A. kNN

k-Nearest-Neighbors (kNN) classifies a point based on the other points that are nearest to it. The predicted probability that a query point falls into class j is proportional to the number of the k training points nearest to the query point that are in class j. Mathematically, the probability that a query point falls into class j is

$$P(y^{(i)} = j | x = x^{(i)}) = \frac{1}{k} \Sigma_{q \in N} 1\{y^{(q)} = j\}$$

where N is the set of indicies of the k points closest to $x^{(i)}$. Thus, a query point is predicted to be of the same class as the plurality of the nearest k training points.

We use the validation set to simultaneously select the number of PCs or kPCs and the optimal value for k. We consider potential $k \in \{1, 3, 5, 7, 9\}$.

B. Softmax Regression

Softmax regression is a parametric method, which assumes y has a multinomial distribution over the classes in the data set. The procedure models the probability that a query point falls into class j as

$$P(y^{(i)} = j | x = x^{(i)}) = \frac{e^{\theta_j^T x^{(i)}}}{\Sigma_{l=1}^{L} e^{\theta_l^T x^{(i)}}}$$

where each θ_j is a vector of parameters associated with a single class $j \in \{1, 2, ..., L\}$. Thus, a query point with feature vector $x^{(i)}$ is predicted to fall into the class j for which $\theta_j^T x^{(i)}$ is greatest. Estimates of the parameter vectors are typically found via maximum likelihood.

C. Support Vector Machines

Support vector machines are binary classifiers (with a response coded as $y \in \{-1, 1\}$), which attempt to find a vector of parameters α to minimize the regularized loss function

$$J(\alpha)_\lambda = \frac{1}{m} \Sigma_{i=1}^{m} [1 - y^{(i)} K^{(i)^T} \alpha] + \frac{\lambda}{2} \alpha^T K \alpha$$

where $K = [K(x^{(i)}, x^{(j)})]_{i,j=1}^{m} = [K^{(1)} \dots K^{(m)}]$, for some kernel function $K(x, y)$.[10] We use a linear kernel, i.e., $K(x, y) = x^T y$. Also, due to computational limits, we use a single regularization weight parameter, which we found worked well on the validation set.

While SVMs were developed in the context of binary classification, they can be employed in many-class classification problems in two ways:

- **One vs. one.** In the 'one vs. one' (ovo) method, $\binom{L}{2}$ SVMs are trained. Each SVM serves to separate one pair of classes from each other. A query point is predicted to be in the class that is chosen most frequently, when the point is evaluated using all of the SVMs.[15]
- **One vs. all.** In the 'one vs. all' (ova) method, L SVMs are trained. Each SVM serves to separate one class from the rest of the classes. A query point's predicted class is the one for which its signed distance to the decision boundary is greatest.[15]

VI. RESULTS & ERROR ANALYSIS

In this section, we review the performance of each of the algorithms that we test and conduct error analysis on those that perform best.

[e] The score vectors in Steps 1 and 2 are not the ones used to construct the final model (in Steps 3 and 4). When selecting the validation-error-minimizing number of score vectors, we use the observations from the training and validation data, projected onto the the PCs (or kPCs) of the training data. (The test data are not involved.) In this way, the entire modeling process–from feature engineering to algorithm application–is cross-validated.

A. Algorithm Comparison

Because we have a many-class problem with roughly uniformly distributed labels, the primary metric by which we assess our algorithms is their misclassification rate. Using the test data, we calculate the misclassification rate of each combination of feature engineering method and classificaton algorithm. These results, and the misclassification rate of each algorithm using the original (full, unmodified) data, are presented in Figure 3.

FIG. 3. Test error for all feature engineering methods & classification algorithms

Algorithm	**PC Features**	**kPC Features**	**Original Data**
kNN	0.127	0.171	0.098
Softmax	0.052	0.129	0.109
SVM (ovo)	0.039	0.231	0.039
SVM (ova)	0.039	0.165	0.038

The F1-score is another useful evaluative metric. Because we are dealing with a many-class problem, there is a single precision, recall, and F1-score for every class. The precision and recall for class j are calculated as[16]

$$precision_j = \frac{\text{correct predictions of class } j}{\text{total predictions of class } j}$$

$$recall_j = \frac{\text{correct predictions of class } j}{\text{total observations in class } j}$$

The F1-score for class j is then calculated as

$$F1_j = 2\frac{precision_j \times recall_j}{precision_j + recall_j}$$

In Figure 4, we present the class-level F1-scores for each feature engineering method and classification algorithm.

FIG. 4. F1-scores by class for all feature engineering methods & classification algorithms

Features	**Algorithm**	WU	WD	Walk	Stand	Sit	Lay
PCs	knn	0.88	0.82	0.90	0.84	0.79	0.97
	Softmax	0.96	0.97	0.96	0.92	0.90	0.98
	SVM (ovo)	0.97	0.99	0.97	0.93	0.91	1.00
	SVM (ova)	0.97	0.99	0.98	0.93	0.90	1.00
kPCs	knn	0.79	0.78	0.82	0.83	0.78	0.95
	Softmax	0.80	0.87	0.82	0.88	0.85	0.95
	SVM (ovo)	0.61	0.81	0.65	0.84	0.75	0.89
	SVM (ova)	0.71	0.83	0.74	0.91	0.85	0.94
Original	knn	0.90	0.86	0.91	0.88	0.85	0.99
	Softmax	0.91	0.86	0.90	0.90	0.82	0.95
	SVM (ovo)	0.96	0.97	0.98	0.93	0.92	0.99
	SVM (ova)	0.97	0.98	0.98	0.93	0.91	1.00

Based on these scores, it seems that some classes are easier to predict than others. For example, the F1-scores for the 'laying' and 'walking' classes are usually higher than those for other classes, and the scores for the 'sitting' and 'standing' classes are generally lower. It seems that 'laying' and 'walking' have relatively distinct accelerometry and gyroscopic patterns, making them easier to predict. However, sitting and standing yield patterns that are similar to each other, making them harder to differentiate.

In terms of algorithm performance, one of the most surprising results is that kPC score vectors do not serve as effective features for any of the algorithms we test. In our results, an algorithm trained on PC score vectors always outperforms the same algorithm trained on kPC score vectors. It seems that the primary nonlinear variation in the data is not highly indicative of the response. Thus, the patterns captured by kPC contribute to our models' variance without reducing its bias, thereby increasing error.

We also note that kNN performs significantly worse than the other methods we tried. We attribute this failure largely to the 'curse of dimensionality': the tendency for algorithm performance to degrade in high-dimensional space.[17] Because it is explicitly based on Euclidean distance between points, kNN is affected greatly, even at a modest number of dimensions.[18]

Although Softmax regression did poorly with the original data set, obtaining a 10% error rate, its performance was markedly improved with the use of PC score vectors as features. With those features, it obtained a test error rate of about 5%. We surmise that the reduction in features (resulting in a reduction in over-fitting) is at the root of this significant drop in error.

We are quite satisfied with the results produced by the two SVM-based methods. Using PC score vectors as features, both of those algorithms perform on-par with previous methods, obtaining test error rates of less than 4%. These results suggest that the data are close to linearly separable in the space of the first 200-300 principal components. We suspect that SVMs are especially effective because their regularized nature serves to combat variance.

The methods presented above do not reduce misclassification error beyond our benchmark. However, we achieve a 4% error rate with less than half the number of features that previous authors required to do the same. Moreover, the SVMs that employ the reduced set of engineered features perform roughly on-par (both in terms of misclassification rate and F1-score) with the SVMs that use the entire data set. Thus, by performing feature engineering/selection, we can both reduce the algorithm's training time and decrease our model's variance, without significantly increasing bias. In Figure 5, we present the number of features that minimized error on the validation data–and therefore the number we use to train our final models.

FIG. 5. Validation-error minimizing number of score vectors

Algorithm	**PC Features**	**kPC Features**
kNN	290	380
Softmax	299	212
SVM (ovo)	220	390
SVM (ova)	204	397

B. Error Analysis & Possible Future Work

Having established which algorithms perform best (ovo and ova SVMs using PCs as features), we conduct some further analysis to determine what improvements might be made. For concision, we only provide graphs/charts for ovo

SVMs. Those for ova SVMs are similar and we believe the same conclusions apply to that algorithm, as well.

In Figure 6, we present precision, recall, and F1-scores, by activity category. As before, we see that the 'sitting' and 'standing' classes are the most problematic. They have the lowest scores on nearly all fronts. Per the confusion matrix in Figure 7, it seems that 'sitting' and 'standing' are the labels most often confused for one another. This seems logical; there is relatively little difference between the position and movement of a cell phone on the waist of a standing person and one on the waist of a sitting person. One potential algorithm improvement might come from testing different ways to separate only 'sitting' and 'standing' observations. The best of those classifiers could be integrated into either of the SVM-based classifiers that we have developed here.

FIG. 6. Error metrics for one vs. one SVMs

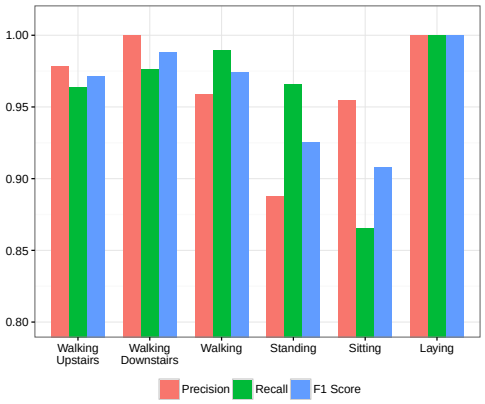


FIG. 7. Confusion matrix for one vs. one SVMs

		Predicted					
		WU	WD	Walk	Stand	Sit	Lay
Actual	WU	454	0	17	0	0	0
	WD	5	410	3	0	2	0
	Walk	4	0	491	1	0	0
	Stand	0	0	0	514	18	0
	Sit	1	0	1	64	425	0
	Lay	0	0	0	0	0	537

We also analyze training error and test error rates, using different amounts of training data. The results are presented in Figure 8. Training error is very low, regardless of the number of training observations. Even when using all observations in the train-val data to train the model, there is a relatively large gap between training error and test error; thus, it seems that the algorithm may be over-fitting to the training data (in spite of the SVMs' regularization), and that most of the remaining error is variance-related.[10]

Consequently, future authors should consider using an even smaller set of features than the one that we have utilized.[10] Specifically, we think that applying the 'one-standard-error rule' when choosing the number of score vectors would be a good first step.[15] We also suspect that a greater number of training examples would be of use.[10] Additionally, future researchers might consider a more methodological tuning of the regularization weight parameter in the SVMs, possibly using distinct parameters for each SVM.[18] All of these alterations will likely serve to decrease the models' variance and hopefully thereby enhance predictive accuracy.

FIG. 8. Test and training ('train-val') error for one vs. one SVMs

VII. CONCLUSIONS

We demonstrate that it is possible to obtain error rates on par with previous HAR research, while using 60% fewer features. We obtain a 4% misclassification rate using both one vs. one and one vs. all SVMs with linear kernels. This is the same rate obtained by previous authors, using one vs. all SVMs with Gaussian kernels.[8] We believe that we find success with only a relatively small number of PC-based features, because the data are close to linearly separable in the space of the first ∼200 PCs. Additionally, we believe that SVMs are an effective algorithm because their regularization penalty helps combat the primary issue we faced: high variance (due to a large feature space).

Because the majority of the errors in our final model are due to confusion of the 'sitting' and 'standing' labels, we suggest that future authors should focus on creating a more effective classifier for only those classes. We also find that our algorithm likely still suffers from high variance; we believe that there are a number of avenues that future authors may pursue in order to remedy this, including regularization parameter tuning and even further feature reduction.

The feature reduction that we achieve is a important improvement for a number of algorithmic reasons, including variance control and algorithm training time. It is particularly significant in the context of cell phones, because the devices have limited input/output and storage capabilities. Most significantly, if a fitness tracking application has lower data requirements, it will be less likely to cause problems with battery, storage, and data limits. As a result, the application will be more likely to be broadly adopted, as that is ultimately the goal.

VIII. ACKNOWLEDGMENTS & REFERENCES

We would like to thank Professor Ng and Professor Duchi for their instruction throughout the quarter. We would also like to thank our Project TA, Francois, for his comments as we completed this paper.

We use the following R packages to conduct our analyses: ggplot2[19], dplyr[20], psych[21], nnet[22], class[22], and e1071[23]. We greatly appreciate work done by the authors of these packages.

[1]B. Rose, No, Phones Aren't More Accurate Than Fitness Wearables. Wired. (Mar. 18, 2015).
[2]D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, eds., *A Public Domain Dataset for Human Activity Recognition Using Smartphones.*, 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. (Bruges, Belgium, 2013).
[3]S. H. E. Kim and D. Cook, Human Activity Recognition and Pattern Discovery, In *IEEE Pervasive Computing* (2009).
[4]L. Bao and S. S. Intille, Activity Recognition from User-Annotated Acceleration Data. PERVASIVE 2004, LNCS 3001.
[5]O. D. Lara and M. A. Labrador, IEEE Communications Surveys & Tutorials: A Survey on Human Activity Recognition using Wearable Sensors (Third Quarter 2013).
[6]O. D. Incel, M. Kose, and C. Ersoy, A review and taxonomy of activity recognition on mobile phones. *In* BioNanoScience **3**, 145 (2013).
[7]D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine. In *Ambient Assisted Living and Home Care* (2012).
[8]D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, c ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) (24-26 April 2013).
[9]T. H. G. James, D. Witten and R. Tibshirani, *Introduction to Statistical Learning with Applications in R*, 6th ed. (Springer, 2013).
[10]A. Ng and J. Duchi, CS229: Machine Learning - Course Notes. (Autumn 2016).
[11]B. Scholkopf, A. Smola, and K. Muller, Kernel Principal Components Analysis (Web: Accessed Nov. 28, 2016).
[12]S. Raschka, Kernel Tricks and Nonlinear Dimensionality Reduction via RBF Kernel PCA (Web: Accessed Nov. 16, 2016).
[13]M. Welling, Kernel Principal Components Analysis (Web: Accessed Nov. 12, 2016).
[14]R. Osadchy, Unsupervised Learning 2011: Kernel PCA (Web: Originated: 2011, Accessed Nov. 1, 2016).
[15]L. Mackey, STATS202: Data Mining and Analysis - Course Notes. (Autumn 2015).
[16]K. Ganesan, Computing Precision and Recall for Multi-Class Classification Problems (Web: Accessed Nov. 28, 2016).
[17]E. P. L. van der Maaten and J. van den Herik, Dimensionality Reduction: A Comparative Review, In *Technical Report TiCC TR* (October 26, 2009).
[18]T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning*, 3rd ed. (Springer, 2013).
[19]H. Wickham, *ggplot2: Elegant Graphics for Data Analysis* (Springer-Verlag New York, 2009).
[20]H. Wickham and R. Francois, *dplyr: A Grammar of Data Manipulation* (2016), r package version 0.5.0.
[21]W. Revelle, *psych: Procedures for Psychological, Psychometric, and Personality Research*, Northwestern University, Evanston, Illinois (2016), r package version 1.6.9.
[22]W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. (Springer, New York, 2002) iSBN 0-387-95457-0.
[23]D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch, *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien* (2015), r package version 1.6-7.