Recognition of Human Activities Using Signals from Inertial Sensors Embedded in Smartphones*

Qing Cai
qcai3@jhu.edu

Lingjie Sang
lingjiesang@jhu.edu

Abstract

In this project, we tried to predict human activities from inertial signals collected with mobile devices. We explored several machine learning techniques, including support vector machine, logistic regression, neural network, gradient boosting trees, and random forest. We tuned the model parameters and compared models in accuracy and testing time. We also applied principal component analysis, linear discriminant analysis and feature importance analysis to reduce the dimensionality of the feature vectors, which reduced the testing time and slightly improved the accuracy. Finally, we developed a new online model based on the logistic regression model to explore extra information based on the hypothesis human activities are usually consecutive. This new model also slightly increased the accuracy. Overall, we achieved better performance than the previous work.

1 Introduction and Background

Mobile technologies penetrating into everyday life is a worldwide trend. More and more people are relying on their smartphones to record and guide their life. In addition, recently launched wearable devices by Apple, Fitbit and many other companies become the new favorite of the market. On the other hand, the unbiased and detailed measurements of daily activities provided by these devices are also of interest in many health studies.

*This report is for our final project of Machine Learning EN 600.475 Fall 2015. The code base, experiments and documents were created solely for this course project.

In this project, we apply machine learning techniques to classify human activates based on the signals recorded by inertial sensors embedded in smartphones. Each sample has a 561-dimension feature vector extracted from inertial signals recorded within 2.5-second time window, and our goal is classify them into one of the six categories: walking, walking upstairs, walking downstairs, sitting, standing, lying (Figure 1, see more details in Section 3.1). We benchmark our models with accuracy, which is defined as the percentage of correct predictions in all predictions. Further, as we might want to deploy the best model on model device in the future, we also care about the computational cost, especially for that for testing. Accordingly, another benchmark is testing time in milliseconds, which is the average time to classify 1470 data points in the development datasets in our 5-fold cross-validation experiments (see more details in Section 3.1). The benchmark for testing time is a little crude because the overhead is not considered, but it gives a ballpark figure of the computational cost for various models.

Figure 1: Human activity recognition problem and solution. Adopted from (Anguita, 2013).

Previous work (Anguita, 2012) (Anguita, 2013) used a Hardware-Friendly Support Vector Machine (SVM) in order to minimize the computational cost and achieved 89% accuracy in testing. In this study,

we explored multiple models including SVM, logistic regression, neural network, random forest and gradient boosting tree. We tuned the crucial parameters of each classifier to achieve its best performance, and compared the best performance across all models. Further, we explored principle component analysis (PCA) and linear discriminant analysis (LDA) to reduce feature dimensions before classification. Amazingly, such dimension reduction techniques reduced computational cost and sometimes even slightly increased the prediction accuracy. We also used random tress to measure the importance of each of the 561 features so that we can cut out some features even before dimensionality reduction, which could further reduce the computational cost as we don't need to compute these features from the original signals. In the end, we hypothesized that human activities are usually very consistent, so that most data points should have the same label as their previous data points. We exploited such information captured the order of the data as a prior probability in the logistic regression model, which slightly increased the accuracy.

In summary, we achieved at best 96.7% accuracy, which is remarkably better than the previous studies. Our code base provided a universal API and pipeline processes for all the models largely facilitated the non-trivial work for tuning and comparison of many models we explored.

2 Motivation and Explored Machine Learning Techniques

2.1 Support Vector Machine (SVM)

As our problem is a classification problem in high-dimensional feature spaces, and we do not know whether the data are linearly separable, we started with SVM. SMV has good reputation in solving classification problems in general and can also deal with non-linear separable or even non-separable data.

Intuitively, SVM divides data points into separate classes by hyper-planes the margin of which is as wide as possible if the data is linearly separable. In addition, kernel trick is used in SVM for non-linear classification. Most SVM implementations do not assume data are perfectly separable, and introduced slack variable to penalize misclassification. Loss function, based on the margin and the error terms, is minimized during the training stage. The trade-off between margin and error terms reflect the trade-off between bias and variance, which can be tuned by the penalty parameter. In our study, we tested various linear and non-linear kernel functions and tuned the penalty parameter of the error term to achieve the best model.

2.2 Logistic Regression

As the linear SVM performed better than non-linear SVMs, we suspected our data might be linearly separable with a few exceptions. Thus, we thought logistic regression could also be a good choice; especially it has the privilege of low computational cost, which makes it more mobile friendly than SVM. Further, it is based probabilistic model, which made its result easier to interpret and built upon than SVM, and we indeed explore this feature in our study later on.

Logistic regression is a special case of generalized linear models. It predicts the probability of particular outcomes. In particular, the conditional distribution of the outcome given the features is a Bernoulli distribution or multinoulli distribution, and the conditional probability is a linear combination of features through logit transform function. Like other linear learning methods, the training results will become sensitive to random errors in the presence of multicollinearity of features, and we adjusted this problem using a penalty on the size of coefficients by *l2* norm. We then tested the Lasso method where *l1* penalty is used instead to reducing the number of used features, as Lasso method prefers solutions with fewer parameter values. We also tuned the penalty values to balance the bias and variance in this model.

2.3 Neural Network

We do not know if the preprocessed feature provided by the data provider will be comprehensive enough. As neural network allows learning new features, we thought it might also be a good choice.

Neural network is a supervised learning algorithm where one or more layers of latent variables are learned, which were then used to generate the output. Although neural networks, especially the deep learning, becomes extreme popular recently, it is not

easy to use. One of the difficulties is to figure out the right topology. As newbies to the neural network, we consulted the Internet, and learned that the best way to start is to use only one hidden layer and tune the number of nodes in the hidden layers, which we used as our major strategy. We also briefly tested models with two hidden layers. In addition, we tested various activation function and objective function in our model.

2.4 Non-parametric Methods

Beside the aforementioned parametric supervised learning methods, we also tried if the non-parametric methods can achieve better accuracy. Decision tree is a non-parametric supervised learning method which is commonly used for classification. It can predict the value of outcome by learning a set of decision rules, which is structured as a tree with the final prediction at the leaves. It is simple and easy to understand, but suffers the over-fitting problem. We applied the following two methods to adjust the over-fitting problem.

Random Forest construct several decision trees randomly and learns the decision rule by averaging the multiple trees. In particular, it randomly samples the training data with replacement and selects a random subset of features before training each tree.

Gradient Boosting Trees is developed with a fixed number of decision trees as base learners. Instead of averaging trees with equal weights as in random forest, gradient boosting trees averaging trees with weights learned iteratively. The basic procedure is in each iteration to sum up the weighted loss function of all current trees and optimize the weight of a newly added tree in order to minimize overall loss function. However, the problem of choosing at each step the best for an arbitrary loss function is a hard optimization problem in general, and so we'll "cheat" by solving a much easier problem instead. The idea is to apply a steepest descent step to this minimization problem. This procedue is performed multiple times iteratively. This method increases the learners performance iteratively, and serves as an alternative of random forest to solve the over-fitting problem.

2.5 Dimentionality Reduction

Our features reside in high dimension space, which is costly to train and test. To reduce the dimensionality, we performed PCA and LDA.

Principal Component analysis (PCA) is a statistical procedure to transform the observational data set into a set of linearly uncorrelated variables called principal components. The basic idea is to project the original data on some orthogonal projection directions and keep the variance of the transformed data as large as possible. These orthogonal projection directions turn out to be the eigenvector of the empirical covariance matrix of the original data matrix. As the number of principal components is usually less than the number of original dimensions, PCA is commonly used to reduce the dimension of the features.

Linear discriminant analysis (LDA) is another techniques to reduce dimentionality. While PCA only considers the features, LDA accounts for both features and labels. The basic idea of LDA is to project the original feature vectors on some directions in the space in order to maximize the ratio of between-class feature variance to the within-class feature variance.

2.6 Feature Importance Analysis by Random Trees

While PCA and LDA transform features into lower dimensions, we cannot avoid calculating the 561 features at the begining. We want to test if some of these 561 features are redudant or useless so that we don't need to compute them from the beginning. As such, we analyzed the importance of these features by random tree.

The algorithm of decision tree selects the best feature to divide the feature space at each node of the tree. In the multiple random trees, important features tend to be close to the root and show up in multiple trees. Thus, we used these criteria to measure the importance of features by forests of trees.

2.7 Online model

In all the previous models, each data point (signals from a 2.5-second time window) is viewed as an isolated event and used to predict human activities. In real life, human activities usually last much

longer than 2.5 seconds. This implies the consecutive data points tend to belong to the same category. To explore this information, we revised our logistic regression model and developed an online model, which adjusts a prior probability calculted from the original logistic regression by considering the predictions of the last few data points. Logistic regression is chosen over SVM as a start point for our modified model because it is a probabilistic model, which is easy to interpret and build upon.

Here is our algorithm: Suppose the number of testing data is n, for some integer $m > 0$ and real value α, we perform the following steps.

1. Train the original logistic regression model
2. For the i-th testing data, where $i = 1, ..., m$, predict the ith outcome variable based on the original logistic regression model
3. For the i-th testing data, where $i = m+1, ..., n$:
   - Calculate the probability of belonging to each class $p_1, ..., p_6$ based on the original logistic regression model
   - Find the most common class, say class k, in the most recent m previous predicted labels
   - Reset $p_k := p_k + \alpha \times (\sum_{i \neq k}(p_i))$ and $p_i = (1 - \alpha) \times p_i$, for all $i \neq k$
   - Predict the outcome variable by the class with the highest probability. The m and α can be tuned to achieve best performance.

3 Methods and Detailed Description of Our Work

3.1 Data Resource and Preparations

Data were downloaded from UCI Machine Learning Repository. The original data were collected by Anguita *et al.* (Anguita, 2013), and their methods are briefly summarized here: 3-axial linear acceleration and 3-axial angular velocity signals were recorded at frequency of 50 Hz by a Samsung Galaxy S II attached to the waist of volunteers (subjects) performing six activities (walking, walking upstairs, walking downstairs, sitting, standing, lying). These signals were filtered and segmented into 2.56 seconds time window with 50% overlap, and then further processed into 561 features in the time and frequency domains. Labels (which of the sixes activities were performed) were manually picked based on video recordings during the experiments. The whole data set consists of 10299 data points from 31 subjects. Each data point is a 561-dimension vector, generated from signals in a 2.56-second time window. 70% data points were ascribed to the training set, and 30% were ascribed to the testing set. The training and testing data were generated from different subjects.

To benchmark our classifiers, we used 5-fold cross validation on the training set. In particular, we divided the training set into 5 subsets. Each time we use 4 subsets for training and 1 subset for validation. We performed such process five times so that each subset was used for validation once, and obtained mean and standard deviation for our benchmarks (including accuracy and running time). Finally we selected the best classifier based on cross validation result, and ran it against the testing data set.

As the features provided by the original data provider seems to be normalized already, we did not perform data normalization. In addition, as one of our experiments explored the information from the order of the data, for easy comparison we did not perform random shuffling for all the experiments. Data normalization and random shuffling are supported by our code base, and from our experience, including them does not have a noticeable effect on our results.

3.2 Code Base and External Libraries

Our code base was written in python, which provides excellent libraries for machine learning. The dependencies of our code base include *numpy*, *scipy*, *matplotlib*, *scikit-learn*, *scikit-neuralnetwork*. The last two libraries for dedicated for machine learning.

Besides available libraries, we chose to use python also because it is a dynamically typed language, where fields can be dynamically bound to objects at runtime. With that we can easily wrap the machine learning libraries and provide a universal interface for all models we explored, with automatic I/O, pipelining, parameter testing. For example, the following code snippet run the experiment with a SVM classifier. It tests parameters *kernel* and *C* (penalty) with various given values. For

the time of sake, we greedy algorithm for parameter tuning: only one parameter is changed each time and the rest other parameters will be set to the default values given as *paramDefaultValue*. By running this function, cross validation is performed for all selected parameter combinations, and mean and standard deviation for the benchmarks are written to a file named *SVM.txt* in the experiment result folder.

```
def RunSVM():
    classifier = "SVM"
    paramValues = {"kernel" : ["poly", "rbf", "linear"],
                   "C" : [0.1, 0.5, 1.0, 2.0]}
    paramDefaultValue = {"kernel" : "linear",
                         "C" : 1.0 }
    experiment(classifier, paramValues, paramDefaultValue)
```

Similarly, in the following example, experiments are performed with logistic regression as the classifier. Before classification, feature dimension is reduced by LDA. Parameter *n_components* of LDA is tested. Again, cross validation is performed and the results are written in *LDA_LogisticRegression.txt* file.

```
def RunLDA_LogisticRegression():
    classifier = "LogisticRegression"
    learner = "LDA"
    learnerParamValues = {"n_components": range(1,6)}
    learnerParamDefaultValue = {"n_components": 5}
    classifierParams = {"penalty": "l2",
                        "C": 2.0}
    experiment_withReduceDimension(classifier, classifierParams,
                learner, learnerParamValues, learnerParamDefaultValue)
```

3.3 Experiment Environment

The experiments were performed on a Macbook Air (Early 2015), with 1.6 GHz Intel Core i5 and 4 GB memories. Python 2.7 were used and all dependent packages were installed through *pip*.

4 Results

4.1 Comparison of Classifiers

4.1.1 SVM

First of all, we tested various classifiers by 5-fold cross validation. We started with SVM. We tested multiple kernels, including polynomial, rbf, linear and sigmoid. Sigmoid performs poorly, with the accuracy less than 20%, which we did not include it in the figure. All other three achieved more than 90% accuracy. Linear kernel achieved the best accuracy ($93.9\% \pm 2.1\%$) with minimal testing time (Figure 2 **a**). Then we tuned C (penalty parameter) for linear kernel. It turned out with the range we tested (0.1 2.0), C has minimal effect on the accuracy; however, larger C tent to use less testing time (Figure 2 **b**). Accordingly, we picked linear kernel with $C = 1.0$ (default value) to compare with other classifiers.

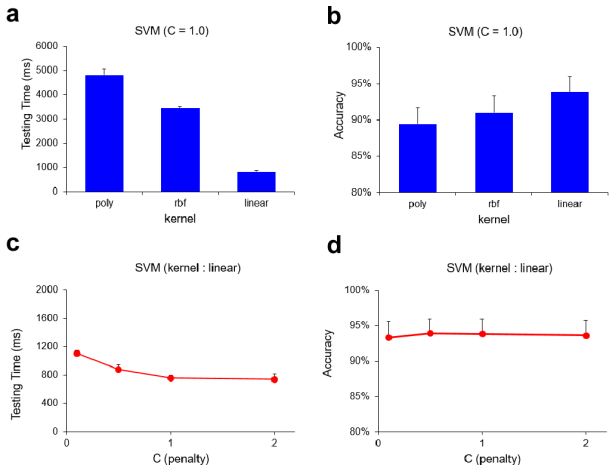

Figure 2: Tuning kernel function and penalty values for SVM.

4.1.2 Logistic Regression

Linear kernel gave the best performance for SVM, which made us wonder if our data are linearly separable so that simpler models requiring less computational resource might also perform well. According, we tested logistic regression. We compared the performance between $l1$ and $l2$ norm and found they had similar accuracy but $l2$ norm outperformed in computational cost (Figure 3 **a**). Thus, we used $l2$ norm and tuned the penalty parameter C. We found larger C tent to slightly outperform in computational cost while the accuracy were very similar among all C values we tested (Figure 3 **b**).

4.1.3 Neural Network

We then used neural network as our classifier. As stated in the method, we used neural network with single hidden layer here. We tried 'maxout' and 'sigmoid' as our activation function and found 'maxout' tends to outperform in accuracy (Figure 4 **a**). We varied the number of hidden nodes from 20 to 200, and found that had minimal effect on the accuracy, although more nodes tends to require larger testing time (Figure 4 **b**).

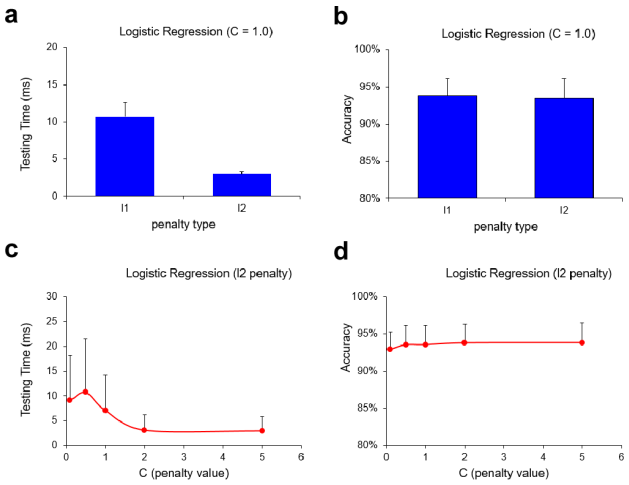

Figure 3: Tuning number of estimators for logistic regression.

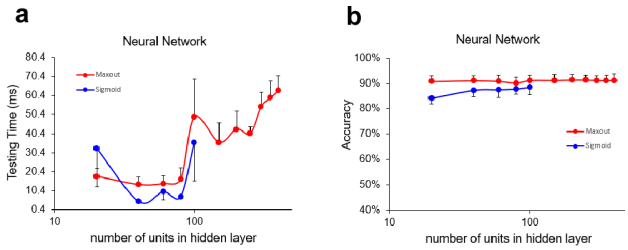

Figure 4: Tuning the number of units in hidden layer and activation function type for neural network.

4.1.4 Non-parametric Methods

We also tried the non-parameter models including random forest (Figure 5) and gradient boosting trees (Figure 6). We tuned parameters like number of estimators for random forest and maximal depth for gradient boosting trees to select the best models for compasiron (Figure 7).

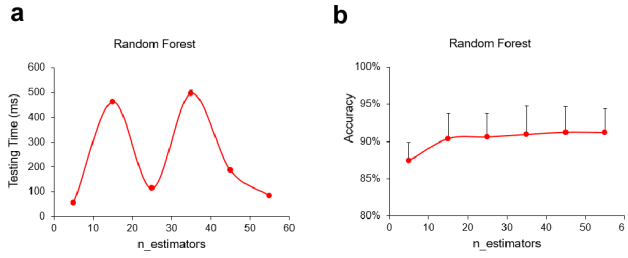

Figure 5: Tuning the number of estimators for random forest.

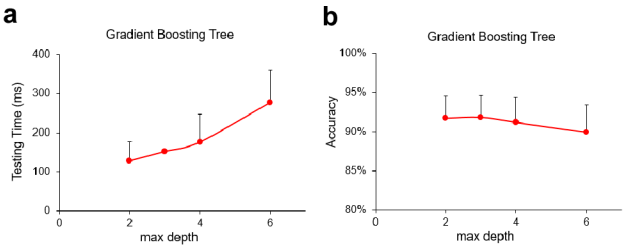

Figure 6: Tuning the maximal depth for gradient boosting tree .

4.1.5 Comparison

In summary, linear SVM has the highest accuracy among all the models, while logistic regression is almost as good as SMV (Figure 7 **c**, the first five bars). However, logistic regression uses two-magnitude lower testing time than SVM (Figure 7 **b**). The other models have lower performance than logistic regression in both accuracy and computational cost (Figure 7 **b, c**, the first five bars). While in general we don't care training time too much as we expect to only deploy testing on mobile device with trained model, we measured the training time for our best models of each classifier. Remarkably, SVM used lowest training time, although its testing time is larger than all other models (Figure 7 **a**, the first five bars). From this result, we learned that reducing testing time is crucial for the performance of SVM when the computational resource is scarce. Originally with the Professor Ilya Shpitser's suggestion, we proposed to try both the primal and dual form SVM, but the library we used does not provide an option for that. Alternatively, we applied reduced dimensionality to largely reduce testing time, which we will describe in the next.

4.2 Dimensionality Reduction

4.2.1 PCA and SVM

To reduce the testing time of our classifiers, especially for SVM, we applied techniques to reduce dimensionality of the features before training and testing with the classifier. We first tried PCA. We tuned the number of components from 1 to 100. Not too surprisingly, with the increasing number of components, the model accuracy increases, and the testing time also tends to increase (Figure 8). However, even with 100 components, the model accuracy is still less 3% less than the original SVM model, while

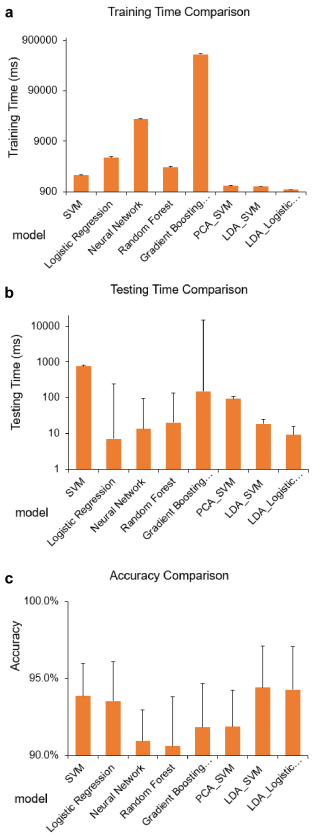

Figure 7: Comparison of various models.

the good news is the testing time is still much lower than the original model.

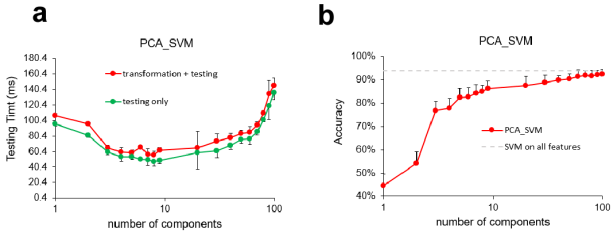

Figure 8: Tuning the number of components in PCA, and test the PCA-generated features by SVM.

4.2.2 LDA and SVM

We then sought help from LDA. The maximal number of components for LDA is 5, so we tuned the number of components from 1 to 5. Amazingly, with 5 components, not only the testing time is minimal comparing to the original model, the accuracy is actually slightly better than the original model (Figure 9).

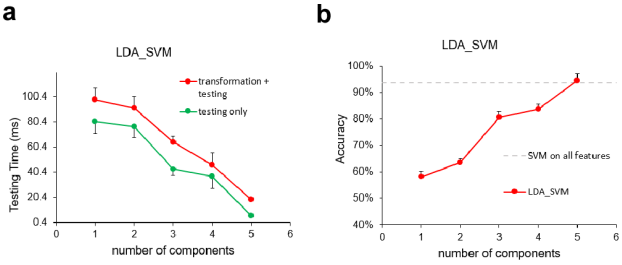

Figure 9: Tuning the number of components in LDA, and test the LDA-generated features by SVM.

We tested the percentage of PCA/LDA transformation time in the total testing time of the new model, and found the transformation time is very small comparing to the testing time (Figure 8 and 9 **a**, comparing the green and red curves). Thus, reducing dimensionality, especially by LDA improved our model both in accuracy and computation time.

4.2.3 LDA and Logistic Regression

Next, we tested whether LDA can improve the logistic regression model. We found that with the number of components set to 5, LDA also slightly increased the accuracy (Figure 10 **b**) for logic regression model. Although the testing time of the logistic regression model itself using the LDA-generated features is largely reduced comparing to the original logistic regression model (Figure 10 **a**, green), the transformation time taken by LDA is much larger than the testing time (Figure 10 **a**, red), and the overall testing time is slightly increased (Figure 7 **b**).

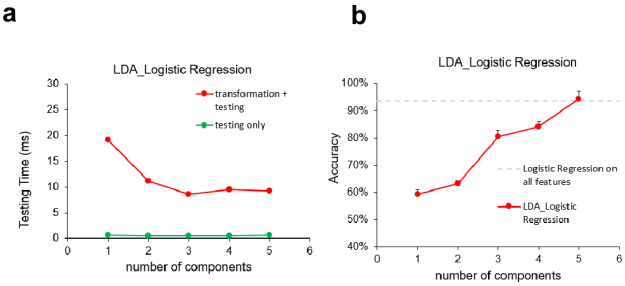

Figure 10: Tuning the number of components in LDA, and test the LDA-generated features by logistic regression.

4.2.4 Features Importance Analysis by Random Trees

All the above dimension reduction techniques still used the 561 features for transforming into lower dimension. We wonder if some of the 561 features are purely redundant or useless and dont need to be extracted from the beginning. That can save more computational resource on the device. To test this, we used random tree to measure the importance of all the features (Figure 11). We found that if we only used the 100 most important features picked by random trees in logistic regression model, we can still achieve 91.5% accuracy (Table 1).

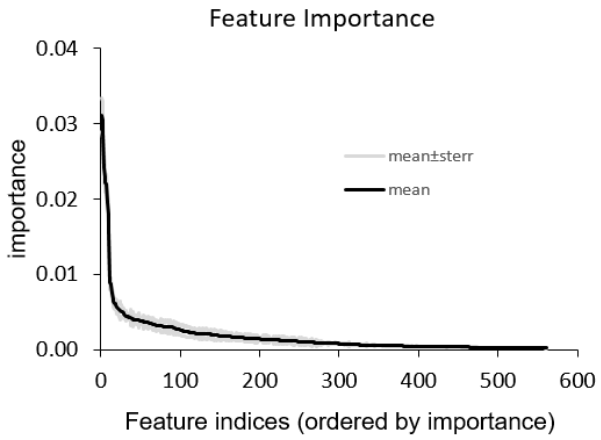

Figure 11: Features importance analysis by random trees

4.3 Online Model

As explained in Section 2.7, human activities are usually consistence, and we exploited this feature with an online model based on the logistic model. For more details, please refer to to Section 2.7. Here we chose $m = 5$ and $\alpha = 0.3$, and achieved a slightly better results than the original logistic model (Table 1). Our model is still crude, but it testifies that the order of data provides some extra information, which is worth exploiting.

4.4 Testing and Comparison with Previous Work

For most our experiments, we only did cross validation on the training dataset. We want to verify if the model also worked well with the testing data set. To do that, we trained our best models with all the training data and tested with the testing data. The performance of this test is actually about 2% better than the cross validation (Table 1), which might due to the larger data set for training and testing. This verifies the generality of our models and experiments.

Table 1: Comparison between development (cross-validation) and testing

model	test	dev
SVM	96.4%	93.9%
logistic regression	96.2%	93.5%
LDA, SVM	96.4%	94.4%
LDA, logistic regression	96.3%	94.3%
logistic regression (100 features)	91.5%	-
logistic regression with prior	96.7%	-

5 Discussion and Comparison to Proposal

Overall, we did pretty much all of the work we proposed, and even more.

For *must achieve*, we proposed to explore SVM and neural network as the classifier for our problem. We actually explored SVM, neural network, logistic regression, gradient boosting trees and random forest. The only thing we did not stick with the proposal is we did not test the primal and dual form of the SVM in order to minimize the testing time as the library we used did not allow, but we found dimensionality reduction can reduce testing time by two magnitude.

For *expected to achieve*, we proposed to adjust parameters in our models to achieve best fit with reasonable computational cost. We wrote a package with universal API and pipelining to achieve that.

For *expected to achieve*, we proposed to perform PCA analysis on the features and test if we can eliminate some of the features. We actually perform PCA and LDA analysis and combined them with our best classifier to achieve very good result. In addition, we also performed feature importance analysis to test if features can be eliminated before performing dimensionality reduction.

For *would like to achieve*, we proposed to explore the order of the data and build a online learning model, which we did, and found it is slightly better than all other models we had.

Through working on this project, we not only

gained the hands-on real data analysis experience by applying the machine learning knowledge we learned in class but also explored, taught ourselves many other techniques, and even developed our own methods. This is a valuable experience about learning, using and developing machine learning algorithms.

Acknowledgments

We would like to thank Professor Ilya Shpitser, Tuo Zhao and Hainan Xu for their support and help with this project and throughout this sememster.

References

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. Dec 2012. *Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine*. International Workshop of Ambient Assisted Living (IWAAL 2012). Vitoria-Gasteiz, Spain.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. 2013. *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.