STAT40800 – Data Programming with Python – Project

Strand 1 – Statistical Modelling/Machine Learning

Brian Buckley 14203480

1. Introduction

The objective of this study was to investigate the predictive accuracy and computational performance of two classification models, Random Forest and Support Vector Machine (SVM), using a large data set with many explanatory variables from the Groupware Human Activity Recognition Program (HAR) [1]. These data arise from a published study by Velloso *et al* [2]. The response variable in this case is multi class as will be explained in section 2 below.

This is an interesting area to study given the growing interest in the Internet of Things and Wearable Computing within the wider community and Sports Science in particular.

2. Data Set

In these data, 6 participants performed weight lifting exercises in 5 different ways. Data from a set of accelerometers were collected and a machine learning model used to predict the weight-lifting technique with a view to identifying common mistakes that reduce the efficacy of the exercise. Figure 1 shows the sensing arrangement that each participant wore during the exercise.

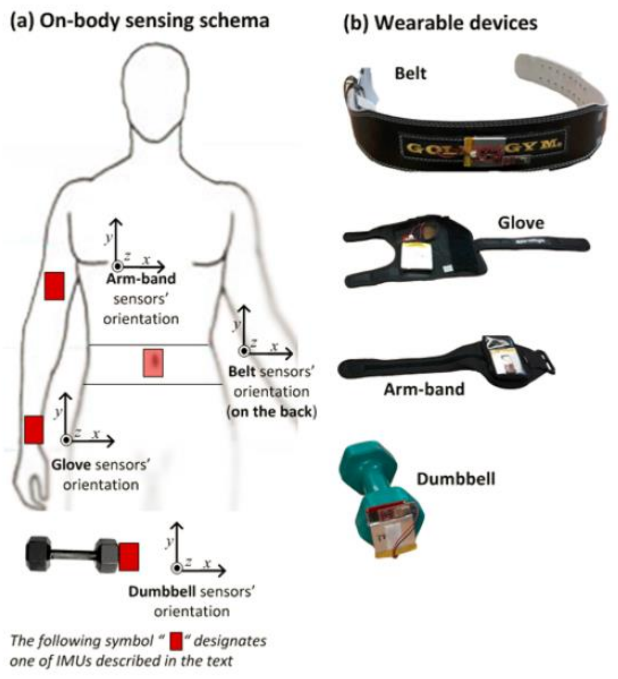

(a) On-body sensing schema
(b) Wearable devices
Belt
Glove
Arm-band
Dumbbell
Arm-band sensors' orientation
Belt sensors' orientation (on the back)
Glove sensors' orientation
Dumbbell sensors' orientation
The following symbol "🟥" designates one of IMUs described in the text

Figure 1: Sensing arrangement on each participant (from Velloso *et al*, ref [2])

The type of technique performed was defined in a categorical factor variable, 'classe'. Table 1 shows the categories with factor levels.

Exercise Performed	**'classe' factor level**
Correct weight-lifting technique	A
Throwing the elbows to the front	B
Lifting the dumbbell only half-way	C
Lowering the dumbbell only half-way	D
Throwing the hips to the front	E

Table 1: Dependent variable 'classe' factor levels

The data set was split into two separate CSV files, one a training set with 19,622 observations of 160 explanatory variables and one a small test set of 20 observations of the same 160 variables randomly extracted from the original HAR data file so that the prediction results could be easily compared. The training set was also split programatically into a cross-correlation set (25% of the traing set) in order to have sufficient data to compute the model accuracies.

3. Data Cleaning

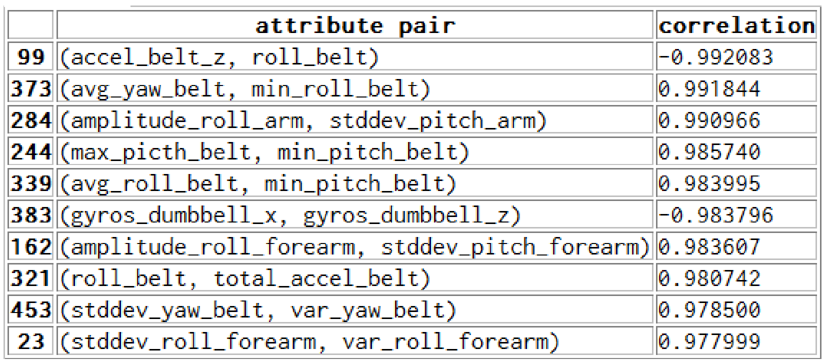
The training and testing sets were checked for missing values and a large number were found. There were also a large number of rows with Microsoft Excel divide-by-zero error strings ('#DIV/0!'). These observation elements were all filled with 0.

Next irrelevant columns for a classification model were deleted from the data sets. These columns contained data such as subject name and various timestamps. All of these appeared in the first seven columns so those were dropped from each data set.

4. Dimensionality/Feature Reduction

The level of autocorrelation between the explanatory variables was explored with a view to reducing the number used in the models. Highly correlated explanatory variables cause over fitting and also reduce the efficiency of the model.

A Pearson correlation coefficient was computed across the data sets using the Python pandas *corr()* function on the data frames. Figure 2 shows the top 10 unique correlated pairs returned. It is clear that there are very high levels of autocorrelation present in the data.



	attribute pair	correlation
99	(accel_belt_z, roll_belt)	-0.992083
373	(avg_yaw_belt, min_roll_belt)	0.991844
284	(amplitude_roll_arm, stddev_pitch_arm)	0.990966
244	(max_picth_belt, min_pitch_belt)	0.985740
339	(avg_roll_belt, min_pitch_belt)	0.983995
383	(gyros_dumbbell_x, gyros_dumbbell_z)	-0.983796
162	(amplitude_roll_forearm, stddev_pitch_forearm)	0.983607
321	(roll_belt, total_accel_belt)	0.980742
453	(stddev_yaw_belt, var_yaw_belt)	0.978500
23	(stddev_roll_forearm, var_roll_forearm)	0.977999

Figure 2: Pearson correlation coefficients of the top 10 unique correlated pairs

Because of the large number of explanatory variables a visual inspection was performed to get a wider qualitative perspective of the autocorrelation level. A correlation matrix plot was generated using the *seaborne* Python library. The plot clearly showed a significant level of autocorrelation present.

A correlation coefficient of 0.5 was used to compute a list of variables that could be dropped from the data set. This resulted in a reduced data set containing 63 explanatory variables from the original 153. A correlation matrix plot of the reduced data set confirmed visually less correlated data. Figure 3 shows the improvement in reducing autocorrelation and reducing the number of explanatory variables. The strength of correlation is given by the depth of colour in each matrix cell.

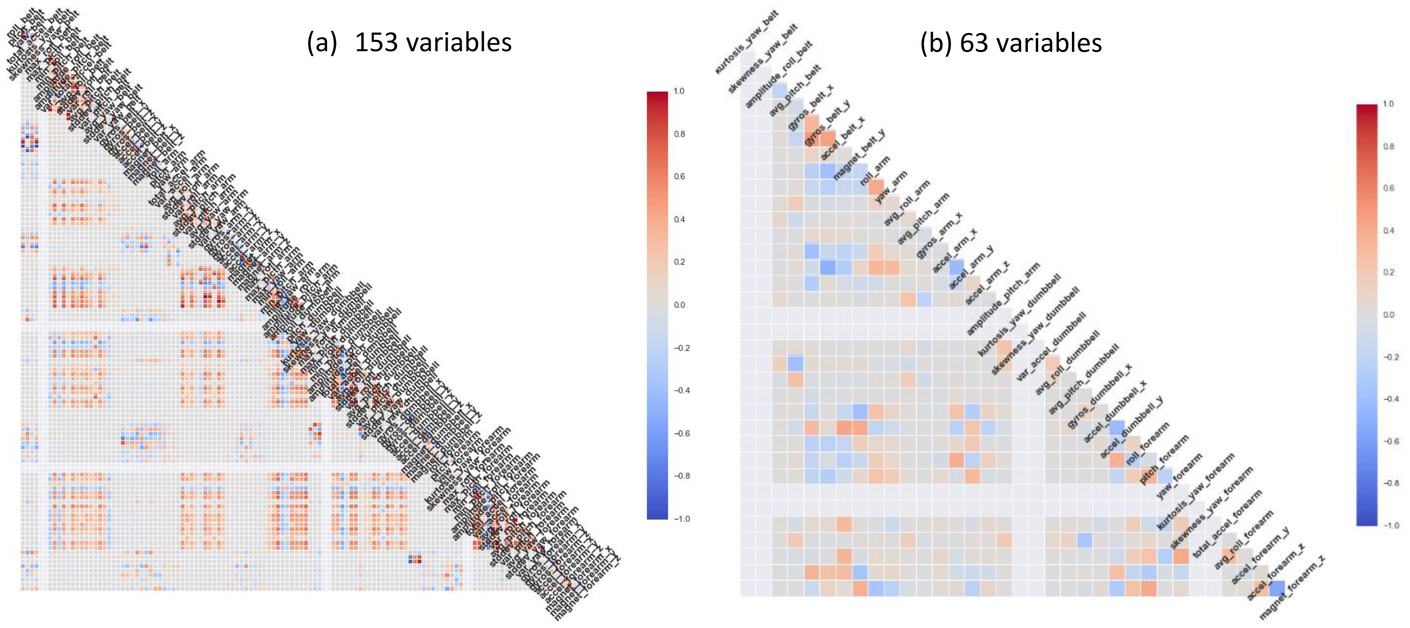


Figure 3: Correlation matrix plots of the original data (LHS) and reduced data with r = 0.5 (RHS)

5. Cross-validation

25% of the training set was randomly allocated for cross-validation to test the accuracy of the two classifier models.

6. Model Fitting and Testing

Two classifier models were fitted to the reduced training set (i.e. the training set minus the 25% cross-validation set), a Random Forest and an SVM. The 'classe' variable was the dependent variable with all remaining variables used for fitting the model.

The data was first standardised using the sklearn preprocessing library (subtract the mean and divide by the standard deviation for each observation value). The 'classe' factors A to E were converted to numerical factors 0 to 4 and each classifier in turn fitted to the reduced training data. The computational performance for model fitting was recorded for each classifier.

The cross-validation set was then used to test the accuracy of the fitted models.

Finally each fitted model was used to predict the 'classe' of the test set.

7. Results

Accuracy using cross-validation data set

The accuracy of each model was tested by comparing the predicted values from the model with the actual values in the cross-validation data. Table 2 shows that the Random Forest was a much more accurate classifier than the SVM.

Measure	**Random Forest**	**Support Vector Machine**
Model Accuracy	94%	51%
Out of sample error	6%	49%
Misclassification rate (test)	5%	40%

Table 2: Classifier model accuracy comparison

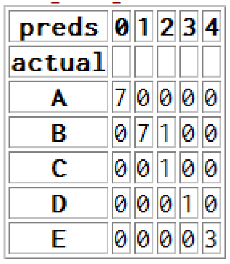
Computational Performance of model fitting

The Python *%timeit* magic method was used to compare the computational performance of the classifiers by timing how long each model took to fit. The Random Forest with *n_jobs* parameter = 2 took 238ms and the SVM took 3.83s (3830ms) so clearly the Random Forest is far more computationally efficient.

Prediction using testing set

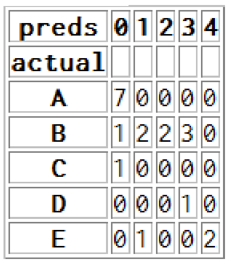
Figure 4 shows the confusion matrix for the predictive accuracy for both models when run against the test set. The Random Forest is clearly a far more accurate classifier for this large data set.

Random Forest



preds	0	1	2	3	4
actual					
A	7	0	0	0	0
B	0	7	1	0	0
C	0	0	1	0	0
D	0	0	0	1	0
E	0	0	0	0	3

SVM



preds	0	1	2	3	4
actual					
A	7	0	0	0	0
B	1	2	2	3	0
C	1	0	0	0	0
D	0	0	0	1	0
E	0	1	0	0	2

Figure 4: The test prediction confusion matrix for Random Forest (LHS) compared with SVM (RHS) for predicted 'classe' dependent variable

8. Conclusion

For this dataset containing a large volume of observations and a large number of explanatory variables the Python *sklearn* Random Forest classifier is significantly more accurate and computationally efficient than the *sklearn* Support Vector Machine classifier. The *sklearn* documentation explains that RandomForests can be fitted in parallel on many cores as each tree is built independently of the others. This is what the *n_jobs* parameter is for and may account for the difference in computational performance.

9. References

[1] Human Activity Recognition program website

[2] Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. **Qualitative Activity Recognition of Weight Lifting Exercises**. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Appendix

Python code

```
# Data Programming with Python Project
#
# Brian Buckley 14203480

# Strand 1 - Statistical Modelling/Machine Learning
#
# Comparison of two classifiers for accuracy and computational efficiency.


## Objective

# The purpose of this exercise is to use the weight-lifting exercise data set of the
# Human Activity Recognition program from Groupware (http://groupware.les.inf.pucrio.br/har).

# In these data, 6 participants performed weight-lifting exercises in 5 different techniques,
# one of which is the correct technique.  Data from a set of accelerometers was collected and
# a machine learning algorithm used to predict the technique performed from the telemetry
# data.

# Our models will use the "classe" variable (classifying the 5 different techniques) in the
# test set to predict the Groupware results


## 1: Data Exploration  *****************************************************************

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Mac path = '/Volumes/BrianB/Courses/UCD Data Analytics/STAT40800 Data
# Programming with Python/Assessments/Project/'

path = 'F:/Courses/UCD Data Analytics/STAT40800 Data Programming with Python/Assessments/Project/'
df_training = pd.read_csv(path + "har-training.csv")
df_testing = pd.read_csv(path + "har-testing.csv")

np.shape(df_training)                                           # (19622, 160)

# Check for NaNs
df_training.isnull().sum().sum()                                # 1921600 NaNs present

# When we look at the data using np.shape() we see that it has 19622 observations on 160
# variables.
# We also note a lot of NAN and empty columns plus some irrelevant variables like user
# name, timestamp, etc.

## 2: Data Cleansing *****************************************************************

# First we cleaned it up by removing all NANs

df1 = df_training.fillna(0)
df1.columns.values

# Next we removed identifier columns such as name, timestamps etc.
```

```
df2 = df1.drop(df1.columns[:7], axis=1)
np.shape(df2)                                                              # (19622,153)

# There are also some divide by zero error cells ("#DIV/0!") that we want to set to 0

df3 = df2.replace({'#DIV/0!': 0}, regex=True)

# do the same for the test set
test1 = df_testing.fillna(0)
test2 = test1.drop(test1.columns[:7], axis=1)

## 3: Cross-validation Set ******************************************************************

# We allocated 25% of the training data for cross-validation to test the accuracy of our models.

np.random.seed(100)                                          # For reproducability of the CV set split
msk = np.random.rand(len(df3)) < 0.75
train = df3[msk]
crossval = df3[~msk]


## 4: Autocorrelation identification ***********************************************************

# As we still have a lot of explanatory variables (153) we used a correlation matrix to identify
# if there are any highly correlated variables in the data (collinearity).
# Highly correlated variables can be removed thereby improving the efficiency of our process
# and reducing overfitting.

corr_pearson = train.corr(method='pearson')
# assume paired variable is the last, then remove correlation with itself
corr_pair = corr_pearson.ix[-1][:-1]
# variables sorted from the most predictive
corr_pair.sort(ascending=False)
vars = corr_pearson.iloc[:-1,:-1]

# We used a correlation coefficient r > 0.5 to remove highly correlated variables.
r = 0.5
significant_corrs = (vars[abs(vars) > r][vars != 1.0]).unstack().dropna().to_dict()

uniq_sig_corrs = pd.DataFrame(list(set([(tuple(sorted(key)), significant_corrs[key]) for key in
significant_corrs])), columns=['attribute pair', 'correlation'])

# sort by absolute value to account for negative correlations
uniq_sig_corrs = uniq_sig_corrs.ix[abs(uniq_sig_corrs['correlation']).argsort()[::-1]]

len(uniq_sig_corrs)                                    # We can see 545 highly correlated pairs

# Look at the top 10
uniq_sig_corrs.head(n=10)

# Visualise using a cross-correlation plot

import seaborn as sns
fig, ax = plt.subplots(figsize=(10, 10))
sns.corrplot(train, ax=ax, annot=False)

## 5: Dimensionality reduction by removing highly autocorrelated variables **********************

# The correlation plot above identifies correlation by the strength of the colour.
# As can be seen in the plot it looks like we do have quite a few autocorrelated variables.

## Now we drop the correlated variables from the data

drop_list = []
for i in uniq_sig_corrs.values:
drop_list.append(i[0][1])

# Drop duplicates
drop_list2 = list(set(drop_list))
train_reduced = train.drop(drop_list2, axis=1)
np.shape(train_reduced)                                                    # (14755,63)

# Now we have reduced our explanatory variables to 63

# Check correlation for train_reduced
# If we plot the correlation matrix again using train_reduced we confirm the data is
# not so highly correlated now.
fig, ax = plt.subplots(figsize=(10, 10))
sns.corrplot(train_reduced, ax=ax)

# drop the same correlated columns from the test set and correlation set
test_reduced = test2.drop(drop_list2, axis=1)
cv_reduced = crossval.drop(drop_list2, axis=1)

## 6: Model fitting – comparing two algorithms, Random Forest and SVM ************************

# Data Scaling / Standardisation
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

train_scaled = train_reduced.drop('classe',axis=1)
test_scaled = test_reduced.drop('classe',axis=1)
cv_scaled = cv_reduced.drop('classe',axis=1)

train_scaled = scaler.fit_transform(train_scaled)  # compute mean, std and transform
test_scaled = scaler.transform(test_scaled)
cv_scaled = scaler.fit_transform(cv_scaled)

# We fit a random forest model to predict the 'classe' variable using everything else as a
# predictor. We used random forest as that is said to be the most accurate classifier where
# a large data set is available (we will compare with SVM later).

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_jobs=2)                    # n_jobs enables parallelisation
y, _ = pd.factorize(train_reduced['classe'])              # factor A-E into 0-4
clf.fit(train_scaled, y)

# Test the computational performance
%timeit clf.fit(train_scaled, y)                          #1 loops, best of 3: 238 ms per loop


## 7: Model Accuracy using the Cross-validation Data **************************************

# Now we want to test the accuracy of our model so we crossvalidate the model using the
# remaining 25% of data.
y_cv = clf.predict(cv_scaled)

# The accuracy of our model is tested by comparing the predicted values from our model to
# the actual values in the CV data.
compare = pd.Categorical(crossval['classe'])
accuracy = float(sum(y_cv == compare.labels))/float(len(cv_scaled))
accuracy                                                                    # 0. 9407301066447908

# Our model accuracy is 94%.
# We also want to know the out-of-sample error. This is the complement of accuracy.
oos_error = 1-accuracy                                                      # 0.05926989335520916

# In our case the out-of-sample error is 6%.

## Test the model against the test set - drop the same correlated columns from the test set
y_pred = clf.predict(test_scaled)
ct = pd.crosstab(test_reduced['classe'], y_pred, rownames=['actual'], colnames=['preds'])

# Misclassification rate for the test set
mr_test = 1 - float(np.trace(ct.values))/float(np.sum(ct.values))
mr_test                                                                     # 0.05



## 8: Compare with a Support Vector Machine ********************************************
from sklearn import svm
clf = svm.LinearSVC()
y, _ = pd.factorize(train_reduced['classe'])
clf.fit(train_scaled, y)

%timeit clf.fit(cv_scaled, y)                                   # 1 loops, best of 3.83 s per loop
# The SVM is significantly less performant compared with the Random Forest in this case.

# Cross-validation test
y_cv = clf.predict(cv_scaled)
compare = pd.Categorical(crossval['classe'])
accuracy = float(sum(y_cv == compare.labels))/float(len(cv_scaled))
accuracy                                                                    # 0.5133305988515177
oos_error = 1-accuracy
oos_error                                                                   # 48666940114848234

# SVM accuracy = 51% and out-of-sample error = 49%

# Test the model against the test set
y_pred = clf.predict(test_scaled)
ct = pd.crosstab(test_reduced['classe'], y_pred, rownames=['actual'], colnames=['preds'])

# Misclassification rate for the test set
mr_test = 1 - float(np.trace(ct.values))/float(np.sum(ct.values))
mr_test                                                                     # 0.4

# The SVM has much less predictive accuracy than the Random Forest for this data analysis.
```