IBM Capstone project for IBM Advanced Data Science | Alex Braga

Author: Alex Braga | alexbraga101@gmail.com

This notebook summarize the work done towards the conclusion on IBM Advanced Data Science on Coursera. The purpose the is to build up a data science project for a real world use case by taking the steps taught during the specialization.

Dataset

The first step for this problem was to select a dataset with a potential application in a field of interest of the candidate.

For this project a dataset available in one of Kaggle's competition was used. https://www.kaggle.com/c/career-con-2019 (https://www.kaggle.com/c/career-con-2019)

Special thanks to Tampere University in Finland, Department of Signal Processing and Department of Automation and Mechanics Engineering for making the data available

Data was collected from a robot in 9 different surfaces taking IMU data(10 channels), each data series have 128 measurements and refer to a single surface category.

Use Case

The objetive of the competition and this project is to predict the floor surface the robot is on based on collect IMU data. This can be applied to real world cases that the surface impact the robot operation, for example vaccum cleaner robot could adapt their suction power and wheel encoder erros reading based on the surface its on, or even further could be applied to connected vehicles to rank a road condition when running on it, and the sharing this data with infrastructure management for road maintanance.

So now let's dive into the code!

First let's get all the dependencies

In [41]:

```
!pip install keras --upgrade
!pip install tensorflow
```

```
Collecting keras
```

```
  Using cached https://files.pythonhosted.org/packages/5e/10/aa32d
ad071ce52b5502266b5c659451cfd6ffcbf14e6c8c4f16c0ff5aaab/Keras-2.2.
4-py2.py3-none-any.whl
Collecting pyyaml (from keras)
Collecting h5py (from keras)
  Using cached https://files.pythonhosted.org/packages/4c/77/c4933
e12dca0f61bcdafc207c7532e1250b8d12719459fd85132f3daa9fd/h5py-2.9.0
-cp35-cp35m-manylinux1_x86_64.whl
Collecting keras-applications>=1.0.6 (from keras)
  Using cached https://files.pythonhosted.org/packages/71/e3/19762
fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Appl
ications-1.0.8-py3-none-any.whl
Collecting six>=1.9.0 (from keras)
  Using cached https://files.pythonhosted.org/packages/73/fb/00a97
6f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0
-py2.py3-none-any.whl
Collecting numpy>=1.9.1 (from keras)
  Using cached https://files.pythonhosted.org/packages/bb/ef/d5a21
cbc094d3f4d5b5336494dbcc9550b70c766a8345513c7c24ed18418/numpy-1.16
.4-cp35-cp35m-manylinux1_x86_64.whl
Collecting scipy>=0.14 (from keras)
  Using cached https://files.pythonhosted.org/packages/14/49/8f13f
a215e10a7ab0731cc95b0e9bb66cf83c6a98260b154cfbd0b55fb19/scipy-1.3.
0-cp35-cp35m-manylinux1_x86_64.whl
Collecting keras-preprocessing>=1.0.5 (from keras)
  Using cached https://files.pythonhosted.org/packages/28/6a/8c1f6
2c37212d9fc441a7e26736df51ce6f0e38455816445471f10da4f0a/Keras_Prep
rocessing-1.1.0-py2.py3-none-any.whl
pyspark 2.3.3 requires py4j==0.10.7, which is not installed.
tensorflow 1.3.0 requires tensorflow-tensorboard<0.2.0,>=0.1.0, wh
ich is not installed.
Installing collected packages: pyyaml, six, numpy, h5py, keras-app
lications, scipy, keras-preprocessing, keras
Successfully installed h5py-2.9.0 keras-2.2.4 keras-applications-1
.0.8 keras-preprocessing-1.1.0 numpy-1.16.4 pyyaml-5.1.1 scipy-1.3
.0 six-1.12.0
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/ca/f2/0931c1
94bb98398017d52c94ee30e5e1a4082ab6af76e204856ff1fdb33e/tensorflow-
1.13.1-cp35-cp35m-manylinux1_x86_64.whl (92.5MB)
    100% |################################| 92.5MB 260kB/s eta 0:0
0:01  9% |##                              | 8.6MB 41.3MB/s eta 0:0
0:03    37% |###########                     | 34.4MB 46.2MB/s eta
0:00:02
Collecting astor>=0.6.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/d1/4f/950dfa
e467b384fc96bc6469de25d832534f6b4441033c39f914efd13418/astor-0.8.0
-py2.py3-none-any.whl
Collecting tensorboard<1.14.0,>=1.13.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/0f/39/bdd75b
08a6fba41f098b6cb091b9e8c7a80e1b4d679a581a0ccd17b10373/tensorboard
-1.13.1-py3-none-any.whl (3.2MB)
    100% |################################| 3.2MB 2.6MB/s eta 0:00
```

```
:01
Collecting grpcio>=1.8.6 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/14/19/f1858e
d60786ff681a5f8681448b56bffaaa87a81e9a7ca5cd075a873b35/grpcio-1.21
.1-cp35-cp35m-manylinux1_x86_64.whl (2.2MB)
    100% |################################| 2.2MB 2.7MB/s eta 0:00
:01
Collecting numpy>=1.13.3 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/bb/ef/d5a21
cbc094d3f4d5b5336494dbcc9550b70c766a8345513c7c24ed18418/numpy-1.16
.4-cp35-cp35m-manylinux1_x86_64.whl
Collecting six>=1.10.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/73/fb/00a97
6f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0
-py2.py3-none-any.whl
Collecting keras-applications>=1.0.6 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/71/e3/19762
fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Appl
ications-1.0.8-py3-none-any.whl
Collecting termcolor>=1.1.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/8a/48/a76be5
1647d0eb9f10e2a4511bf3ffb8cc1e6b14e9e4fab46173aa79f981/termcolor-1
.1.0.tar.gz
Collecting keras-preprocessing>=1.0.5 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/28/6a/8c1f6
2c37212d9fc441a7e26736df51ce6f0e38455816445471f10da4f0a/Keras_Prep
rocessing-1.1.0-py2.py3-none-any.whl
Collecting wheel>=0.26 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/bb/10/44230d
d6bf3563b8f227dbf344c908d412ad2ff48066476672f3a72e174e/wheel-0.33.
4-py2.py3-none-any.whl
Collecting protobuf>=3.6.1 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/7c/d2/581ebc
3c41879aca2c4fce5c37cdb8d779c4ea79109b6da7f640735ea0a2/protobuf-3.
8.0-cp35-cp35m-manylinux1_x86_64.whl (1.2MB)
    100% |################################| 1.2MB 3.5MB/s eta 0:00
:01
Collecting tensorflow-estimator<1.14.0rc0,>=1.13.0 (from tensorflo
w)
  Downloading https://files.pythonhosted.org/packages/bb/48/13f49f
c3fa0fdf916aa1419013bb8f2ad09674c275b4046d5ee669a46873/tensorflow_
estimator-1.13.0-py2.py3-none-any.whl (367kB)
    100% |################################| 368kB 4.6MB/s eta 0:00
:01
Collecting gast>=0.2.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/4e/35/11749b
f99b2d4e3cceb4d55ca22590b0d7c2c62b9de38ac4a4a7f4687421/gast-0.2.2.
tar.gz
Collecting absl-py>=0.1.6 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/da/3f/9b0355
080b81b15ba6a9ffcf1f5ea39e307a2778b2f2dc8694724e8abd5b/absl-py-0.7
.1.tar.gz (99kB)
    100% |################################| 102kB 4.1MB/s ta 0:00:
```

```
01
Collecting werkzeug>=0.11.15 (from tensorboard<1.14.0,>=1.13.0->te
nsorflow)
  Downloading https://files.pythonhosted.org/packages/9f/57/92a497
e38161ce40606c27a86759c6b92dd34fcdb33f64171ec559257c02/Werkzeug-0.
15.4-py2.py3-none-any.whl (327kB)
    100% |################################| 327kB 4.3MB/s eta 0:00
:01
Collecting markdown>=2.6.8 (from tensorboard<1.14.0,>=1.13.0->tens
orflow)
  Downloading https://files.pythonhosted.org/packages/c0/4e/fd492e
91abdc2d2fcb70ef453064d980688762079397f779758e055f6575/Markdown-3.
1.1-py2.py3-none-any.whl (87kB)
    100% |################################| 92kB 4.3MB/s eta 0:00:
01
Collecting h5py (from keras-applications>=1.0.6->tensorflow)
  Using cached https://files.pythonhosted.org/packages/4c/77/c4933
e12dca0f61bcdafc207c7532e1250b8d12719459fd85132f3daa9fd/h5py-2.9.0
-cp35-cp35m-manylinux1_x86_64.whl
Collecting setuptools (from protobuf>=3.6.1->tensorflow)
  Downloading https://files.pythonhosted.org/packages/ec/51/f45cea
425fd5cb0b0380f5b0f048ebc1da5b417e48d304838c02d6288a1e/setuptools-
41.0.1-py2.py3-none-any.whl (575kB)
    100% |################################| 583kB 4.4MB/s eta 0:00
:01
Collecting mock>=2.0.0 (from tensorflow-estimator<1.14.0rc0,>=1.13
.0->tensorflow)
  Downloading https://files.pythonhosted.org/packages/05/d2/f94e68
be6b17f46d2c353564da56e6fb89ef09faeeff3313a046cb810ca9/mock-3.0.5-
py2.py3-none-any.whl
Building wheels for collected packages: termcolor, gast, absl-py
  Running setup.py bdist_wheel for termcolor ... done
  Stored in directory: /home/spark/shared/.cache/pip/wheels/7c/06/
54/bc84598ba1daf8f970247f550b175aaaee85f68b4b0c5ab2c6
  Running setup.py bdist_wheel for gast ... done
  Stored in directory: /home/spark/shared/.cache/pip/wheels/5c/2e/
7e/a1d4d4fcebe6c381f378ce7743a3ced3699feb89bcfbdadadd
  Running setup.py bdist_wheel for absl-py ... done
  Stored in directory: /home/spark/shared/.cache/pip/wheels/ee/98/
38/46cbcc5a93cfea5492d19c38562691ddb23b940176c14f7b48
Successfully built termcolor gast absl-py
pyspark 2.3.3 requires py4j==0.10.7, which is not installed.
Installing collected packages: astor, six, setuptools, protobuf, g
rpcio, wheel, werkzeug, markdown, absl-py, numpy, tensorboard, h5p
y, keras-applications, termcolor, keras-preprocessing, mock, tenso
rflow-estimator, gast, tensorflow
Successfully installed absl-py-0.7.1 astor-0.8.0 gast-0.2.2 grpcio
-1.21.1 h5py-2.9.0 keras-applications-1.0.8 keras-preprocessing-1.
1.0 markdown-3.1.1 mock-3.0.5 numpy-1.16.4 protobuf-3.8.0 setuptoo
ls-41.0.1 six-1.12.0 tensorboard-1.13.1 tensorflow-1.13.1 tensorfl
ow-estimator-1.13.0 termcolor-1.1.0 werkzeug-0.15.4 wheel-0.33.4
Target directory /home/spark/shared/user-libs/python3/h5py already
exists. Specify --upgrade to force replacement.
```

```
Target directory /home/spark/shared/user-libs/python3/numpy-1.16.4
.dist-info already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/keras_applic
ations already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/__pycache__
already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/numpy alread
y exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/h5py-2.9.0.d
ist-info already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/six.py alrea
dy exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/keras_prepro
cessing already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/Keras_Prepro
cessing-1.1.0.dist-info already exists. Specify --upgrade to force
replacement.
Target directory /home/spark/shared/user-libs/python3/six-1.12.0.d
ist-info already exists. Specify --upgrade to force replacement.
Target directory /home/spark/shared/user-libs/python3/Keras_Applic
ations-1.0.8.dist-info already exists. Specify --upgrade to force
replacement.
Target directory /home/spark/shared/user-libs/python3/bin already
exists. Specify --upgrade to force replacement.
```

In [44]:

```
import numpy as np
import pandas as pd
import os
from time import time
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from matplotlib import rcParams
get_ipython().magic(u'matplotlib inline')
le = preprocessing.LabelEncoder()
from numba import jit
import itertools
from seaborn import countplot, barplot
from numba import jit
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn import preprocessing
from scipy.stats import randint as sp_randint
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import keras
import matplotlib.style as style
style.use('ggplot')
import warnings
warnings.filterwarnings('ignore')
import gc
gc.enable()
pd.set_option('display.max_columns', 100)
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3
from sklearn.decomposition import PCA
from scipy.stats import kurtosis
from scipy.stats import skew
import tensorflow as tf
from keras.utils import to_categorical
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from numpy import array
from numpy import argmax
from numpy.fft import *
import os
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.style as style
style.use('ggplot')
import warnings
warnings.filterwarnings('ignore')
import plotly.offline as py
from plotly.offline import init_notebook_mode, iplot
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier, GradientBoosti
ngClassifier
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import confusion_matrix
import gc
from sklearn.model_selection import train_test_split
def _kurtosis(x):
    return kurtosis(x)

def CPT5(x):
    den = len(x)*np.exp(np.std(x))
    return sum(np.exp(x))/den

def skewness(x):
```

```
    return skew(x)

def SSC(x):
    x = np.array(x)
    x = np.append(x[-1], x)
    x = np.append(x,x[1])
    xn = x[1:len(x)-1]
    xn_i2 = x[2:len(x)]    # xn+1
    xn_i1 = x[0:len(x)-2]  # xn-1
    ans = np.heaviside((xn-xn_i1)*(xn-xn_i2),0)
    return sum(ans[1:])

def wave_length(x):
    x = np.array(x)
    x = np.append(x[-1], x)
    x = np.append(x,x[1])
    xn = x[1:len(x)-1]
    xn_i2 = x[2:len(x)]    # xn+1
    return sum(abs(xn_i2-xn))

def norm_entropy(x):
    tresh = 3
    return sum(np.power(abs(x),tresh))

def SRAV(x):
    SRA = sum(np.sqrt(abs(x)))
    return np.power(SRA/len(x),2)

def mean_abs(x):
    return sum(abs(x))/len(x)

def zero_crossing(x):
    x = np.array(x)
    x = np.append(x[-1], x)
    x = np.append(x,x[1])
    xn = x[1:len(x)-1]
    xn_i2 = x[2:len(x)]    # xn+1
    return sum(np.heaviside(-xn*xn_i2,0))
```

Reading Data

Now we can load the Data into a pandas dataset, X is the raw measurement data from the sensors and Y is the surface label for each 128 step measurement

In [5]:
```
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Stora
ge. It includes your credentials.
# You might want to remove those credentials before you share your
notebook.
client_575b3298bd77482e938fcf5f0ca0035e = ibm_boto3.client(service_
name='s3',
    ibm_api_key_id='SBaaQGYuDA45dxgB6YBIqYRQLD5hsEXRYoIEr1UnujW4',
    ibm_auth_endpoint="https://iam.bluemix.net/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.netwo
rklayer.com')

body = client_575b3298bd77482e938fcf5f0ca0035e.get_object(Bucket='d
efault-donotdelete-pr-olizxj1zhbkctg',Key='Y.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like
object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType(
__iter__, body )

Y_data = pd.read_csv(body)


body = client_575b3298bd77482e938fcf5f0ca0035e.get_object(Bucket='d
efault-donotdelete-pr-olizxj1zhbkctg',Key='X.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like
object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType(
__iter__, body )

X_data = pd.read_csv(body)
```

Data Exploration

In this step we will be able to anwser questions like What data is available? What type of data is available? how much data and is it distributed?

At first, let's check their shape

In [6]:
```
X_data.shape
```

Out[6]: (487680, 13)

In [7]:
```
Y_data.shape
```

Out[7]: (3810, 3)

So we don't have the same amount of rows in X and Y data, that may be an issue. We had better that a view on these data.

In [8]:
```
X_data.head()
```

Out[8]:

	row_id	series_id	measurement_number	orientation_X	orientation_Y	orientatio
0	0_0	0	0	-0.75853	-0.63435	-0.10488
1	0_1	0	1	-0.75853	-0.63434	-0.10490
2	0_2	0	2	-0.75853	-0.63435	-0.10492
3	0_3	0	3	-0.75852	-0.63436	-0.10495
4	0_4	0	4	-0.75852	-0.63435	-0.10495

In [9]:
```
Y_data.head()
```

Out[9]:

	series_id	group_id	surface
0	0	13	fine_concrete
1	1	31	concrete
2	2	20	concrete
3	3	31	concrete
4	4	22	soft_tiles

X_data contains 10 channels measurements from the robots IMU data, each measurement series is separatable by the series_id row, and within each series we can check each measurement_number. Y_data in the order hand assign the surface label for each series, that explains the difference in the number of rows among them. And it also became evidence that further ahead we will need to transfed X_data in series_id group.

Now let's check the data type available

In [10]:
```
X_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 487680 entries, 0 to 487679
Data columns (total 13 columns):
row_id                    487680 non-null object
series_id                 487680 non-null int64
measurement_number        487680 non-null int64
orientation_X             487680 non-null float64
orientation_Y             487680 non-null float64
orientation_Z             487680 non-null float64
orientation_W             487680 non-null float64
angular_velocity_X        487680 non-null float64
angular_velocity_Y        487680 non-null float64
angular_velocity_Z        487680 non-null float64
linear_acceleration_X     487680 non-null float64
linear_acceleration_Y     487680 non-null float64
linear_acceleration_Z     487680 non-null float64
dtypes: float64(10), int64(2), object(1)
memory usage: 48.4+ MB
```

In [11]:
```
Y_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3810 entries, 0 to 3809
Data columns (total 3 columns):
series_id    3810 non-null int64
group_id     3810 non-null int64
surface      3810 non-null object
dtypes: int64(2), object(1)
memory usage: 89.4+ KB
```

Nothing very special in the result, we only need to take consideration that the label is provided in string formating, we should probably change it to int or one hot encoding afterwards to simplify the training.

Let's plot basic statics on the data to start to check for any quality issues

In [12]:
```
X_data.describe()
```

Out[12]:

	series_id	measurement_number	orientation_X	orientation_Y	orie
count	487680.000000	487680.000000	487680.000000	487680.000000	48768
mean	1904.500000	63.500000	-0.018050	0.075062	0.012
std	1099.853353	36.949327	0.685696	0.708226	0.105
min	0.000000	0.000000	-0.989100	-0.989650	-0.16
25%	952.000000	31.750000	-0.705120	-0.688980	-0.08
50%	1904.500000	63.500000	-0.105960	0.237855	0.031
75%	2857.000000	95.250000	0.651803	0.809550	0.122
max	3809.000000	127.000000	0.989100	0.988980	0.155

In [13]:
```
Y_data.describe()
```

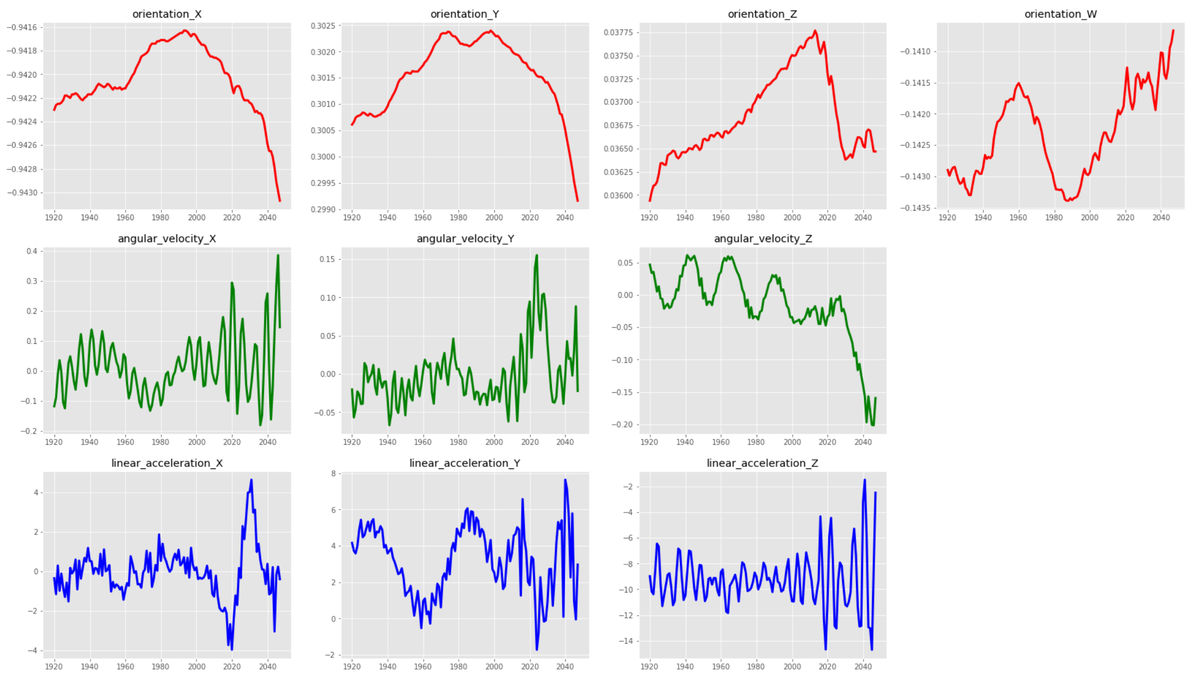
Out[13]:

	series_id	group_id
count	3810.000000	3810.000000
mean	1904.500000	37.601312
std	1099.996591	20.982743
min	0.000000	0.000000
25%	952.250000	19.000000
50%	1904.500000	39.000000
75%	2856.750000	55.000000
max	3809.000000	72.000000

By using the helper function bellow we can get a series measurement plot.

In [14]:
```
series_dict = {}
for series in (X_data['series_id'].unique()):
    series_dict[series] = X_data[X_data['series_id'] == series]

def plotSeries(series_id):
    style.use('ggplot')
    plt.figure(figsize=(28, 16))
    print(Y_data[Y_data['series_id'] == series_id]['surface'].value
s[0].title())
    for i, col in enumerate(series_dict[series_id].columns[3:]):
        if col.startswith("o"):
            color = 'red'
        elif col.startswith("a"):
            color = 'green'
        else:
            color = 'blue'
        if i >= 7:
            i+=1
        plt.subplot(3, 4, i + 1)
        plt.plot(series_dict[series_id][col], color=color, linewidt
h=3)
        plt.title(col)

id_series = 15
plotSeries(id_series)
```

```
Carpet
```



Some channels appear to have very noise data, that may be intrinsic to the sensor itslef and not the environment the robot it is running. We should have better filter these input later on.

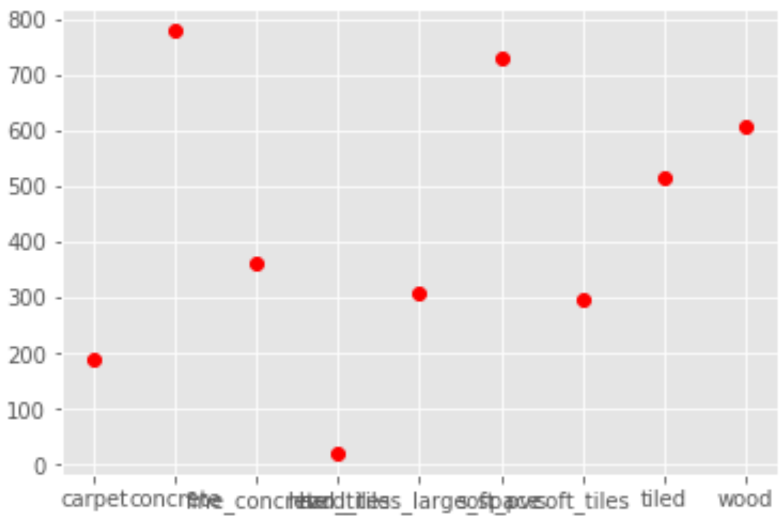
Let's see how the different types of surfaces are distributed in the given data

```
In [15]: Y_data['surface'].value_counts()
```

```
Out[15]: concrete                 779
         soft_pvc                 732
         wood                     607
         tiled                    514
         fine_concrete            363
         hard_tiles_large_space   308
         soft_tiles               297
         carpet                   189
         hard_tiles                21
         Name: surface, dtype: int64
```

```
In [16]: plt.plot(Y_data['surface'].value_counts(),'ro')
```

```
Out[16]: [<matplotlib.lines.Line2D at 0x7f3348d999e8>]
```



There are nine different floor surfaces, the dataset is not evenly distributed and hard_tiles have very few examples, which may cause a difficult on learning its pattern.

Data Cleansing

Now that an overview of the data was provided, let's go ahead and cleanup any potential quality issues that may impact our model training

Check with there is any null or duplicate measurements and labels

In [17]:
```
X_data.isnull().sum()
```

Out[17]:
```
row_id                   0
series_id                0
measurement_number       0
orientation_X            0
orientation_Y            0
orientation_Z            0
orientation_W            0
angular_velocity_X       0
angular_velocity_Y       0
angular_velocity_Z       0
linear_acceleration_X    0
linear_acceleration_Y    0
linear_acceleration_Z    0
dtype: int64
```

In [18]:
```
Y_data.isnull().sum()
```

Out[18]:
```
series_id    0
group_id     0
surface      0
dtype: int64
```

In [19]:
```
X_data.duplicated().value_counts()
```

Out[19]:
```
False    487680
dtype: int64
```

In [20]:
```
X_data.shape
```

Out[20]:
```
(487680, 13)
```

In [21]:
```
Y_data.duplicated().value_counts()
```

Out[21]:
```
False    3810
dtype: int64
```

In [22]:
```
Y_data.shape
```

Out[22]:
```
(3810, 3)
```

In [23]:
```
X_data.fillna(0, inplace = True)
X_data.replace(-np.inf, 0, inplace = True)
X_data.replace(np.inf, 0, inplace = True)
Y_data.fillna(0, inplace = True)
Y_data.replace(-np.inf, 0, inplace = True)
Y_data.replace(np.inf, 0, inplace = True)
```

No problem, looks like the data was already prechecked for the kaggle competition. Let's go create new features and transform the data for the model.

Feature Engineering

In this step we will start to create new features for our model. The first transformation is to add the equivalent rotation in euler angles given the raw data quaternion data. In robotics the use of quaternion is widespread to avoid the gimbal lock issue that may occurs when using euler representation. The gimbal lock issue refers to the lack of representativeness of the euler model when two axis allign during a rotation process. So let's add euler angles to the data:

In [24]:
```
#https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_E
uler_angles
#quaternion to eular
def quaternion_to_euler(qx,qy,qz,qw):
    import math
    # roll (x-axis rotation)
    sinr_cosp = +2.0 * (qw * qx + qy + qz)
    cosr_cosp = +1.0 - 2.0 * (qx * qx + qy * qy)
    roll = math.atan2(sinr_cosp, cosr_cosp)

    # pitch (y-axis rotation)
    sinp = +2.0 * (qw * qy - qz * qx)
    if(math.fabs(sinp) >= 1):
        pitch = copysign(M_PI/2, sinp)
    else:
        pitch = math.asin(sinp)

    # yaw (z-axis rotation)
    siny_cosp = +2.0 * (qw * qz + qx * qy)
    cosy_cosp = +1.0 - 2.0 * (qy * qy + qz * qz)
    yaw = math.atan2(siny_cosp, cosy_cosp)

    return roll, pitch, yaw

def eular_angle(data):
    x, y, z, w = data['orientation_X'].tolist(), data['orientation_
Y'].tolist(), data['orientation_Z'].tolist(), data['orientation_W']
.tolist()
    nx, ny, nz = [], [], []
    for i in range(len(x)):
        xx, yy, zz = quaternion_to_euler(x[i], y[i], z[i], w[i])
        nx.append(xx)
        ny.append(yy)
        nz.append(zz)

    data['euler_x'] = nx
    data['euler_y'] = ny
    data['euler_z'] = nz

    return data

eular_angle(X_data).head()
```

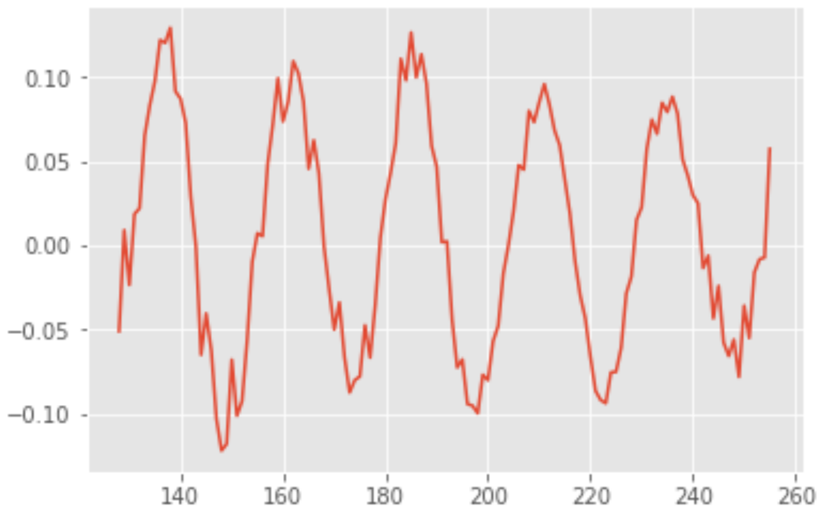
Out[24]:

	row_id	series_id	measurement_number	orientation_X	orientation_Y	orientatio
0	0_0	0	0	-0.75853	-0.63435	-0.10488
1	0_1	0	1	-0.75853	-0.63434	-0.10490
2	0_2	0	2	-0.75853	-0.63435	-0.10492
3	0_3	0	3	-0.75852	-0.63436	-0.10495
4	0_4	0	4	-0.75852	-0.63435	-0.10495

Now we will tackle the sensor noise. Since this noise don't bring usefull information for our model and may even affect its performance, we had better filter all data as described in the following steps.

In [25]:
```
plt.plot(X_data.angular_velocity_Z[128:256])
```

Out[25]: [<matplotlib.lines.Line2D at 0x7f333308ec50>]

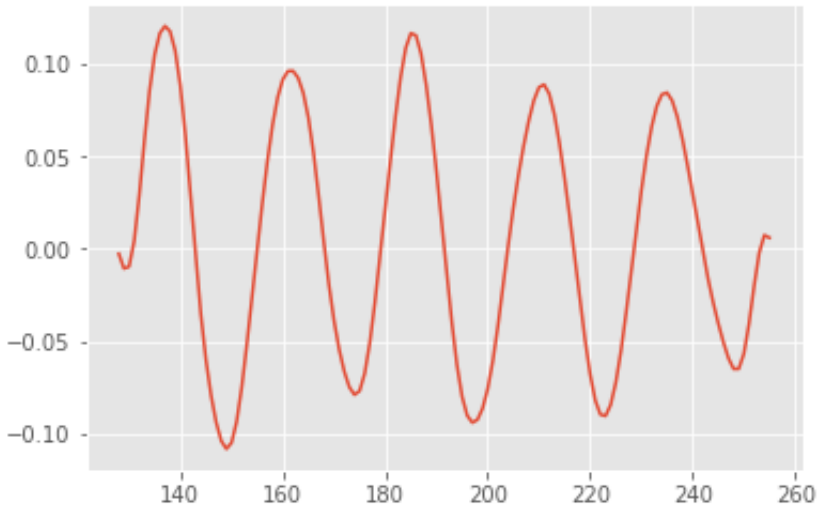


In [26]:
```
def filter_signal(signal, threshold=1e3):
    fourier = rfft(signal)
    frequencies = rfftfreq(signal.size, d=20e-3/signal.size)
    fourier[frequencies > threshold] = 0
    return irfft(fourier)

X_data_denoised = X_data.copy()

for col in X_data.columns:
    if col[0:3] == 'ang' or col[0:3] == 'lin':
        # Apply filter_signal function to the data in each series
        denoised_data = X_data.groupby(['series_id'])[col].apply(la
mbda x: filter_signal(x))

        # Assign the denoised data back to X_data
        list_denoised_data = []
        for arr in denoised_data:
            for val in arr:
                list_denoised_data.append(val)

        X_data_denoised[col] = list_denoised_data

series_dict = {}
for series in (X_data_denoised['series_id'].unique()):
    series_dict[series] = X_data_denoised[X_data_denoised['series_i
d'] == series]
```

In [27]:
```
X_data=X_data_denoised.copy()
```

In [28]:
```
plt.plot(X_data_denoised.angular_velocity_Z[128:256])
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x7f334a0b4ac8>]
```



As of now all data is separated in each axis, so we can create features calculating the magnitude a relation between them.

In [29]:
```
X_data['total_angular_vel'] = (X_data['angular_velocity_X']**2 + X_data['angular_velocity_Y']**2 + X_data['angular_velocity_Z']**2)**0.5
X_data['total_linear_acc'] = (X_data['linear_acceleration_X']**2 + X_data['linear_acceleration_Y']**2 + X_data['linear_acceleration_Z']**2)**0.5
X_data['total_orientation'] = (X_data['orientation_X']**2 + X_data['orientation_Y']**2 + X_data['orientation_Z']**2)**0.5
X_data['acc_vs_vel'] = X_data['total_linear_acc'] / X_data['total_angular_vel']
X_data['total_angle'] = (X_data['euler_x'] ** 2 + X_data['euler_y'] ** 2 + X_data['euler_z'] ** 2) ** 0.5
X_data['angle_vs_acc'] = X_data['total_angle'] / X_data['total_linear_acc']
X_data['angle_vs_vel'] = X_data['total_angle'] / X_data['total_angular_vel']
```

Finally we can bring the data from the 128 time step to a measurement domain by using basic statics of the time series from each measurement.

In [30]:
```
Input = pd.DataFrame()
for col in X_data.columns:
        if col in ['row_id','series_id','measurement_number']:
            continue
        print ("FE on column ", col, "...")
        Input[col + '_mean'] = X_data.groupby(['series_id'])[col].mean()
        Input[col + '_median'] = X_data.groupby(['series_id'])[col].median()
        Input[col + '_max'] = X_data.groupby(['series_id'])[col].max()
        Input[col + '_min'] = X_data.groupby(['series_id'])[col].min()
        Input[col + '_std'] = X_data.groupby(['series_id'])[col].std()
```

```
FE on column  orientation_X ...
FE on column  orientation_Y ...
FE on column  orientation_Z ...
FE on column  orientation_W ...
FE on column  angular_velocity_X ...
FE on column  angular_velocity_Y ...
FE on column  angular_velocity_Z ...
FE on column  linear_acceleration_X ...
FE on column  linear_acceleration_Y ...
FE on column  linear_acceleration_Z ...
FE on column  euler_x ...
FE on column  euler_y ...
FE on column  euler_z ...
FE on column  total_angular_vel ...
FE on column  total_linear_acc ...
FE on column  total_orientation ...
FE on column  acc_vs_vel ...
FE on column  total_angle ...
FE on column  angle_vs_acc ...
FE on column  angle_vs_vel ...
```

With the measurement domain data in hand, we will normalize each column

In [31]:
```
x = Input.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
Input = pd.DataFrame(x_scaled)
```

The target label data type is string so we should transform it to a unique int label.

In [32]:
```
le=LabelEncoder()
Y_data['surface'] = le.fit_transform(Y_data['surface'])
Y_data.head()
```

Out[32]:

	series_id	group_id	surface
0	0	13	2
1	1	31	1
2	2	20	1
3	3	31	1
4	4	22	6

Let's divide the data in test and train using 80/20 ratio given then small dataset.

In [33]:
```
xTrain, xTest, yTrain, yTest = train_test_split(Input, Y_data, test
_size=0.2, random_state = 0)
```

One of the models in the next chapter will be a neural network. Then the label data is transform to a one hot encoded data

In [34]:
```
encoded = to_categorical(yTrain['surface'])
yTrainOHE = pd.DataFrame(encoded)

encoded = to_categorical(yTest['surface'])
yTestOHE = pd.DataFrame(encoded)

yTrainOHE.head()
```

Out[34]:

	0	1	2	3	4	5	6	7	8
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Model | Neural Network

First model to be included is a neural network with a sequential topology using a sigmoid activation function and a one hot encoded output.

In [35]:
```
[xTrain.shape, xTest.shape, yTrainOHE.shape, yTestOHE.shape]
```

Out[35]:
```
[(3048, 100), (762, 100), (3048, 9), (762, 9)]
```

In [47]:
```
model = keras.Sequential()
model.add(keras.layers.Dense(64, input_shape=(100,), activation=tf.
nn.sigmoid))
model.add(keras.layers.Dense(32,activation=tf.nn.sigmoid))
model.add(keras.layers.Dense(16,activation=tf.nn.sigmoid))
model.add(keras.layers.Dense(9,activation=tf.nn.softmax))

model.compile(optimizer=tf.train.RMSPropOptimizer(0.01),
              loss=keras.losses.categorical_crossentropy,
              metrics=[keras.metrics.categorical_accuracy])

model.fit(xTrain, yTrainOHE, epochs=50)
```

```
Epoch 1/50
3048/3048 [==============================] - 0s 134us/step - loss:
2.2840 - categorical_accuracy: 0.1401
Epoch 2/50
3048/3048 [==============================] - 0s 60us/step - loss:
1.9881 - categorical_accuracy: 0.2303
Epoch 3/50
3048/3048 [==============================] - 0s 54us/step - loss:
1.7919 - categorical_accuracy: 0.3264
Epoch 4/50
3048/3048 [==============================] - 0s 56us/step - loss:
1.6180 - categorical_accuracy: 0.4029
Epoch 5/50
3048/3048 [==============================] - 0s 57us/step - loss:
1.5153 - categorical_accuracy: 0.4360
Epoch 6/50
3048/3048 [==============================] - 0s 58us/step - loss:
1.4382 - categorical_accuracy: 0.4797
Epoch 7/50
3048/3048 [==============================] - 0s 58us/step - loss:
1.3281 - categorical_accuracy: 0.5249
Epoch 8/50
3048/3048 [==============================] - 0s 59us/step - loss:
1.2514 - categorical_accuracy: 0.5384
Epoch 9/50
3048/3048 [==============================] - 0s 66us/step - loss:
1.1767 - categorical_accuracy: 0.5846
Epoch 10/50
3048/3048 [==============================] - 0s 57us/step - loss:
1.1120 - categorical_accuracy: 0.6175
Epoch 11/50
```

```
3048/3048 [==============================] - 0s 58us/step - loss:
1.0506 - categorical_accuracy: 0.6447
Epoch 12/50
3048/3048 [==============================] - 0s 55us/step - loss:
1.0036 - categorical_accuracy: 0.6621
Epoch 13/50
3048/3048 [==============================] - 0s 55us/step - loss:
0.9653 - categorical_accuracy: 0.6683
Epoch 14/50
3048/3048 [==============================] - 0s 55us/step - loss:
0.9250 - categorical_accuracy: 0.6880
Epoch 15/50
3048/3048 [==============================] - 0s 63us/step - loss:
0.8950 - categorical_accuracy: 0.6972
Epoch 16/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.8636 - categorical_accuracy: 0.7113
Epoch 17/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.8334 - categorical_accuracy: 0.7172
Epoch 18/50
3048/3048 [==============================] - 0s 54us/step - loss:
0.8088 - categorical_accuracy: 0.7241
Epoch 19/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.7831 - categorical_accuracy: 0.7359
Epoch 20/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.7631 - categorical_accuracy: 0.7425
Epoch 21/50
3048/3048 [==============================] - 0s 69us/step - loss:
0.7452 - categorical_accuracy: 0.7438
Epoch 22/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.7240 - categorical_accuracy: 0.7500
Epoch 23/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.6983 - categorical_accuracy: 0.7566
Epoch 24/50
3048/3048 [==============================] - 0s 61us/step - loss:
0.6748 - categorical_accuracy: 0.7657
Epoch 25/50
3048/3048 [==============================] - 0s 55us/step - loss:
0.6523 - categorical_accuracy: 0.7720
Epoch 26/50
3048/3048 [==============================] - 0s 68us/step - loss:
0.6277 - categorical_accuracy: 0.7779
Epoch 27/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.6065 - categorical_accuracy: 0.7828
Epoch 28/50
3048/3048 [==============================] - 0s 54us/step - loss:
0.5864 - categorical_accuracy: 0.7943
```

```
Epoch 29/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.5725 - categorical_accuracy: 0.8035
Epoch 30/50
3048/3048 [==============================] - 0s 56us/step - loss:
0.5679 - categorical_accuracy: 0.7992
Epoch 31/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.5388 - categorical_accuracy: 0.8136
Epoch 32/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.5239 - categorical_accuracy: 0.8173
Epoch 33/50
3048/3048 [==============================] - 0s 52us/step - loss:
0.5052 - categorical_accuracy: 0.8219
Epoch 34/50
3048/3048 [==============================] - 0s 53us/step - loss:
0.5121 - categorical_accuracy: 0.8189
Epoch 35/50
3048/3048 [==============================] - 0s 54us/step - loss:
0.4810 - categorical_accuracy: 0.8383
Epoch 36/50
3048/3048 [==============================] - 0s 54us/step - loss:
0.4763 - categorical_accuracy: 0.8346
Epoch 37/50
3048/3048 [==============================] - 0s 60us/step - loss:
0.4530 - categorical_accuracy: 0.8383
Epoch 38/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.4482 - categorical_accuracy: 0.8435
Epoch 39/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.4463 - categorical_accuracy: 0.8419
Epoch 40/50
3048/3048 [==============================] - 0s 55us/step - loss:
0.4426 - categorical_accuracy: 0.8392
Epoch 41/50
3048/3048 [==============================] - 0s 59us/step - loss:
0.4201 - categorical_accuracy: 0.8494
Epoch 42/50
3048/3048 [==============================] - 0s 53us/step - loss:
0.4185 - categorical_accuracy: 0.8596
Epoch 43/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.4201 - categorical_accuracy: 0.8507
Epoch 44/50
3048/3048 [==============================] - 0s 60us/step - loss:
0.4054 - categorical_accuracy: 0.8589
Epoch 45/50
3048/3048 [==============================] - 0s 59us/step - loss:
0.3994 - categorical_accuracy: 0.8537
Epoch 46/50
3048/3048 [==============================] - 0s 71us/step - loss:
```

```
0.4045 - categorical_accuracy: 0.8537
Epoch 47/50
3048/3048 [==============================] - 0s 60us/step - loss:
0.3748 - categorical_accuracy: 0.8671
Epoch 48/50
3048/3048 [==============================] - 0s 57us/step - loss:
0.3811 - categorical_accuracy: 0.8658
Epoch 49/50
3048/3048 [==============================] - 0s 67us/step - loss:
0.3800 - categorical_accuracy: 0.8661
Epoch 50/50
3048/3048 [==============================] - 0s 58us/step - loss:
0.3560 - categorical_accuracy: 0.8694
```

Out[47]: `<keras.callbacks.History at 0x7f3333248198>`

In [48]:

```
model.evaluate(xTest,yTestOHE)
```

```
762/762 [==============================] - 0s 50us/step
```

Out[48]: `[0.78956017594324945, 0.75853018435280462]`

Even with a small model and few iterations of parameter tuning it was possible to achieve over 75% of accuracy.

Model | Random Forest

Second model is the Random Forest, which creates a structure of decision trees to absorb the data patterns and represent the model. It is well now for the good performance on multiclassification problems.

In [49]:

```
yTrain.head()
```

Out[49]:

	series_id	group_id	surface
3257	3257	70	5
3017	3017	62	1
3355	3355	38	8
1763	1763	72	2
1044	1044	33	4

In [50]:
```
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=60)
predicted = np.zeros((xTest.shape[0],9))
measured= np.zeros((Input.shape[0]))
score = 0

for times, (trn_idx, val_idx) in enumerate(folds.split(Input.values
,Y_data['surface'].values)):
    model2 = RandomForestClassifier(n_estimators=300, max_depth=5,
min_samples_split=5, n_jobs=-1)
    model2.fit(Input.iloc[trn_idx],Y_data['surface'][trn_idx])
    measured[val_idx] = model2.predict(Input.iloc[val_idx])
    predicted += model2.predict_proba(xTest)/folds.n_splits
    score += model2.score(Input.iloc[val_idx],Y_data['surface'][val
_idx])
    print("Fold: {} score: {}".format(times,model2.score(Input.iloc
[val_idx],Y_data['surface'][val_idx])))

    gc.collect()
```

```
Fold: 0 score: 0.6945169712793734
Fold: 1 score: 0.7477124183006536
Fold: 2 score: 0.6902887139107612
Fold: 3 score: 0.6934210526315789
Fold: 4 score: 0.726552179656539
```

In [51]:
```
print('Average score', score / folds.n_splits)
```

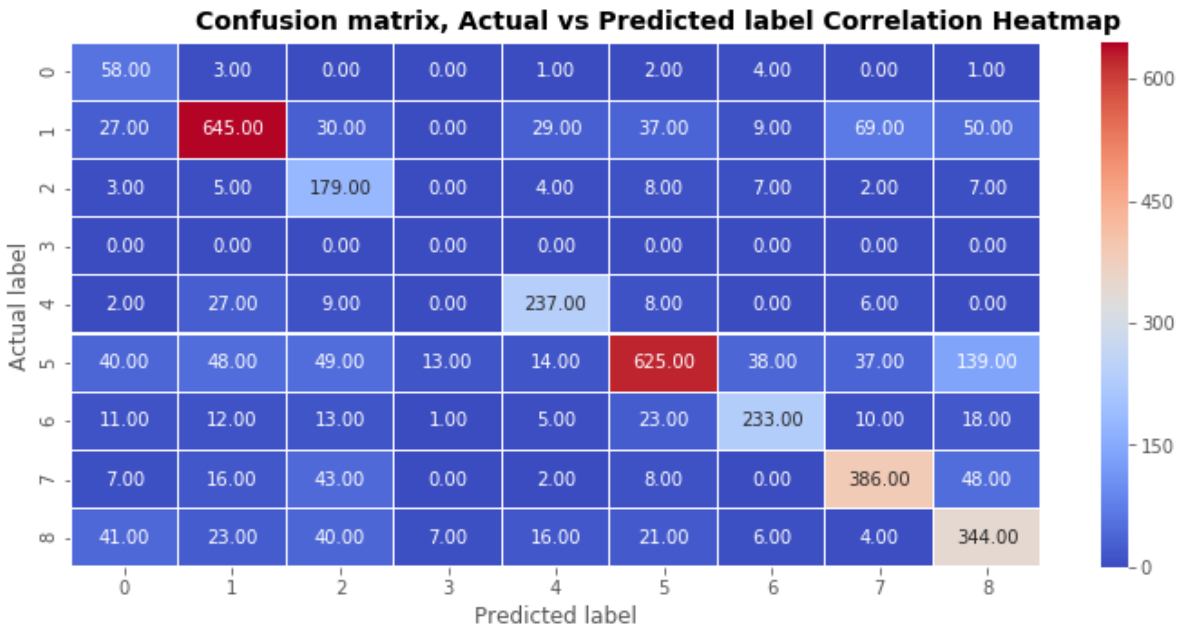
```
Average score 0.710498267156
```

Work on tuning was more demanding than from the neural network and even so the perfomance is still lower than the previous model.

In [52]:
```
confusion_matrix(measured,Y_data['surface'])
```

Out[52]:
```
array([[ 58,   3,   0,   0,   1,   2,   4,   0,   1],
       [ 27, 645,  30,   0,  29,  37,   9,  69,  50],
       [  3,   5, 179,   0,   4,   8,   7,   2,   7],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  2,  27,   9,   0, 237,   8,   0,   6,   0],
       [ 40,  48,  49,  13,  14, 625,  38,  37, 139],
       [ 11,  12,  13,   1,   5,  23, 233,  10,  18],
       [  7,  16,  43,   0,   2,   8,   0, 386,  48],
       [ 41,  23,  40,   7,  16,  21,   6,   4, 344]])
```

```
In [54]: fig, ax = plt.subplots(1,1,figsize=(12,5))
         sns.heatmap(pd.DataFrame(confusion_matrix(measured,Y_data['surface'
         ])),
                     ax = ax,
                     cmap = 'coolwarm',
                     annot = True,
                     fmt = '.2f',
                     linewidths = 0.05)
         fig.subplots_adjust(top=0.93)
         fig.suptitle('Confusion matrix, Actual vs Predicted label Correlati
         on Heatmap',
                      fontsize=14,
                      fontweight='bold')
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')

Out[54]: Text(0.5,24,'Predicted label')
```



Feature Engineering 2

One actions to improve the perfomance of the classifiers is improving the data in itself. So before making anymore tunings in the model, let's create additional features.

In [55]:
```
Input2=Input.copy()
for col in X_data.columns:
        if col in ['row_id','series_id','measurement_number']:
            continue
        print ("FE on column ", col, "...")
        #Input2[col + '_range'] = Input2[col + '_max'] - Input2[col
+ '_min']
        #Input2[col + '_maxtoMin'] = Input2[col + '_max'] / Input2[
col + '_min']
        Input2[col + '_mad'] = X_data.groupby(['series_id'])[col].a
pply(lambda x: np.median(np.abs(np.diff(x))))
        Input2[col + '_abs_max'] = X_data.groupby(['series_id'])[co
l].apply(lambda x: np.max(np.abs(x)))
        Input2[col + '_abs_min'] = X_data.groupby(['series_id'])[co
l].apply(lambda x: np.min(np.abs(x)))
        Input2[col + '_abs_avg'] = (Input2[col + '_abs_min'] + Inpu
t2[col + '_abs_max'])/2
        Input2[col + '_skew'] = X_data.groupby(['series_id'])[col].
skew()
        Input2[col + '_mad'] = X_data.groupby(['series_id'])[col].m
ad()
        Input2[col + '_q25'] = X_data.groupby(['series_id'])[col].q
uantile(0.25)
        Input2[col + '_q75'] = X_data.groupby(['series_id'])[col].q
uantile(0.75)
        Input2[col + '_q95'] = X_data.groupby(['series_id'])[col].q
uantile(0.95)
        Input2[col + '_iqr'] = Input2[col + '_q75'] - Input2[col +
'_q25']
        Input2[col + '_SSC'] = X_data.groupby(['series_id'])[col].a
pply(SSC)
        Input2[col + '_skewness'] = X_data.groupby(['series_id'])[c
ol].apply(skewness)
        Input2[col + '_wave_lenght'] = X_data.groupby(['series_id']
)[col].apply(wave_length)
        Input2[col + '_norm_entropy'] = X_data.groupby(['series_id'
])[col].apply(norm_entropy)
        Input2[col + '_SRAV'] = X_data.groupby(['series_id'])[col].
apply(SRAV)
        Input2[col + '_kurtosis'] = X_data.groupby(['series_id'])[c
ol].apply(_kurtosis)
        Input2[col + '_zero_crossing'] = X_data.groupby(['series_id
'])[col].apply(zero_crossing)
```

```
FE on column  orientation_X ...
FE on column  orientation_Y ...
FE on column  orientation_Z ...
FE on column  orientation_W ...
FE on column  angular_velocity_X ...
FE on column  angular_velocity_Y ...
FE on column  angular_velocity_Z ...
FE on column  linear_acceleration_X ...
FE on column  linear_acceleration_Y ...
FE on column  linear_acceleration_Z ...
FE on column  euler_x ...
FE on column  euler_y ...
FE on column  euler_z ...
FE on column  total_angular_vel ...
FE on column  total_linear_acc ...
FE on column  total_orientation ...
FE on column  acc_vs_vel ...
FE on column  total_angle ...
FE on column  angle_vs_acc ...
FE on column  angle_vs_vel ...
```

We normalize the data again and encode the label

In [56]:
```
x2 = Input2.values
min_max_scaler2 = preprocessing.MinMaxScaler()
x_scaled2 = min_max_scaler2.fit_transform(x2)
Input2 = pd.DataFrame(x_scaled2)
```

In [57]:
```
Y_data2=Y_data.copy()
le2=LabelEncoder()
Y_data2['surface'] = le.fit_transform(Y_data['surface'])
Y_data2.head()
```

Out[57]:

	series_id	group_id	surface
0	0	13	2
1	1	31	1
2	2	20	1
3	3	31	1
4	4	22	6

Separated in test and train, additionally transform the label into a one hot encoded shape.

In [58]:
```
xTrain2, xTest2, yTrain2, yTest2 = train_test_split(Input2, Y_data2
, test_size=0.2, random_state = 0)
```

In [59]:
```
encoded2 = to_categorical(yTrain2['surface'])
yTrainOHE2 = pd.DataFrame(encoded2)

encoded2 = to_categorical(yTest2['surface'])
yTestOHE2 = pd.DataFrame(encoded2)

yTrainOHE2.head()
```

Out[59]:

	0	1	2	3	4	5	6	7	8
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Model | Neural Network 2

We repeat the previous chapter steps and bring on improvements in to model to accomodate the new data.

In [60]:
```
[xTrain2.shape, xTest2.shape, yTrainOHE2.shape, yTestOHE2.shape]
```

Out[60]:
```
[(3048, 420), (762, 420), (3048, 9), (762, 9)]
```

In [61]:
```
model3 = keras.Sequential()
model3.add(keras.layers.Dense(256, input_shape=(420,), activation=t
f.nn.sigmoid))
model3.add(keras.layers.Dense(128,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(64,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(32,activation=tf.nn.sigmoid))
model3.add(keras.layers.Dense(9,activation=tf.nn.softmax))

model3.compile(loss='categorical_crossentropy', optimizer='adam', m
etrics=['accuracy'])


model3.fit(xTrain2, yTrainOHE2, epochs=200)
```

```
Epoch 1/200
3048/3048 [==============================] - 1s 242us/step - loss:
```

```
2.0347 - acc: 0.1949
Epoch 2/200
3048/3048 [==============================] - 0s 144us/step - loss:
1.9499 - acc: 0.2808
Epoch 3/200
3048/3048 [==============================] - 0s 135us/step - loss:
1.7333 - acc: 0.3944
Epoch 4/200
3048/3048 [==============================] - 0s 144us/step - loss:
1.6046 - acc: 0.4242
Epoch 5/200
3048/3048 [==============================] - 0s 132us/step - loss:
1.5142 - acc: 0.4665
Epoch 6/200
3048/3048 [==============================] - 1s 168us/step - loss:
1.4176 - acc: 0.5095
Epoch 7/200
3048/3048 [==============================] - 0s 153us/step - loss:
1.3405 - acc: 0.5390
Epoch 8/200
3048/3048 [==============================] - 0s 141us/step - loss:
1.2962 - acc: 0.5551
Epoch 9/200
3048/3048 [==============================] - 0s 132us/step - loss:
1.2316 - acc: 0.5915
Epoch 10/200
3048/3048 [==============================] - 0s 131us/step - loss:
1.1807 - acc: 0.6224
Epoch 11/200
3048/3048 [==============================] - 0s 148us/step - loss:
1.1389 - acc: 0.6296
Epoch 12/200
3048/3048 [==============================] - 0s 127us/step - loss:
1.1012 - acc: 0.6503
Epoch 13/200
3048/3048 [==============================] - 0s 133us/step - loss:
1.0683 - acc: 0.6578
Epoch 14/200
3048/3048 [==============================] - 0s 134us/step - loss:
1.0346 - acc: 0.6788
Epoch 15/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.9966 - acc: 0.6877
Epoch 16/200
3048/3048 [==============================] - 1s 171us/step - loss:
0.9800 - acc: 0.6916
Epoch 17/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.9403 - acc: 0.7172
Epoch 18/200
3048/3048 [==============================] - 0s 146us/step - loss:
0.9424 - acc: 0.7064
Epoch 19/200
```

```
3048/3048 [==============================] - 0s 156us/step - loss:
0.9014 - acc: 0.7218
Epoch 20/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.8864 - acc: 0.7310
Epoch 21/200
3048/3048 [==============================] - 0s 152us/step - loss:
0.8613 - acc: 0.7359
Epoch 22/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.8272 - acc: 0.7477
Epoch 23/200
3048/3048 [==============================] - 0s 150us/step - loss:
0.8096 - acc: 0.7526
Epoch 24/200
3048/3048 [==============================] - 0s 163us/step - loss:
0.7900 - acc: 0.7579
Epoch 25/200
3048/3048 [==============================] - 0s 153us/step - loss:
0.7629 - acc: 0.7641
Epoch 26/200
3048/3048 [==============================] - 0s 130us/step - loss:
0.7566 - acc: 0.7648
Epoch 27/200
3048/3048 [==============================] - 0s 161us/step - loss:
0.7411 - acc: 0.7700
Epoch 28/200
3048/3048 [==============================] - 0s 151us/step - loss:
0.7083 - acc: 0.7815
Epoch 29/200
3048/3048 [==============================] - 1s 192us/step - loss:
0.6945 - acc: 0.7825
Epoch 30/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.6662 - acc: 0.7992
Epoch 31/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.6490 - acc: 0.8035
Epoch 32/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.6362 - acc: 0.8097
Epoch 33/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.6257 - acc: 0.8071
Epoch 34/200
3048/3048 [==============================] - 0s 161us/step - loss:
0.5891 - acc: 0.8251
Epoch 35/200
3048/3048 [==============================] - 0s 152us/step - loss:
0.5828 - acc: 0.8245
Epoch 36/200
3048/3048 [==============================] - 0s 141us/step - loss:
0.5723 - acc: 0.8222
```

```
Epoch 37/200
3048/3048 [==============================] - 1s 188us/step - loss:
0.5523 - acc: 0.8264
Epoch 38/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.5229 - acc: 0.8386
Epoch 39/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.5140 - acc: 0.8360
Epoch 40/200
3048/3048 [==============================] - 1s 168us/step - loss:
0.5036 - acc: 0.8448
Epoch 41/200
3048/3048 [==============================] - 0s 143us/step - loss:
0.4975 - acc: 0.8438
Epoch 42/200
3048/3048 [==============================] - 0s 157us/step - loss:
0.4629 - acc: 0.8615
Epoch 43/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.4450 - acc: 0.8635
Epoch 44/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.4376 - acc: 0.8619
Epoch 45/200
3048/3048 [==============================] - 1s 186us/step - loss:
0.4245 - acc: 0.8675
Epoch 46/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.4055 - acc: 0.8747
Epoch 47/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.3966 - acc: 0.8793
Epoch 48/200
3048/3048 [==============================] - 0s 154us/step - loss:
0.3789 - acc: 0.8888
Epoch 49/200
3048/3048 [==============================] - 0s 142us/step - loss:
0.3750 - acc: 0.8839
Epoch 50/200
3048/3048 [==============================] - 0s 145us/step - loss:
0.3543 - acc: 0.8898
Epoch 51/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.3476 - acc: 0.8934
Epoch 52/200
3048/3048 [==============================] - 0s 148us/step - loss:
0.3495 - acc: 0.8937
Epoch 53/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.3336 - acc: 0.8973
Epoch 54/200
3048/3048 [==============================] - 0s 143us/step - loss:
```

```
0.3152 - acc: 0.9029
Epoch 55/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.3145 - acc: 0.8983
Epoch 56/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.3313 - acc: 0.8914
Epoch 57/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.2895 - acc: 0.9098
Epoch 58/200
3048/3048 [==============================] - 0s 152us/step - loss:
0.2859 - acc: 0.9104
Epoch 59/200
3048/3048 [==============================] - 0s 142us/step - loss:
0.2727 - acc: 0.9170
Epoch 60/200
3048/3048 [==============================] - 0s 138us/step - loss:
0.2637 - acc: 0.9216
Epoch 61/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.2618 - acc: 0.9199
Epoch 62/200
3048/3048 [==============================] - 0s 144us/step - loss:
0.2569 - acc: 0.9236
Epoch 63/200
3048/3048 [==============================] - 1s 167us/step - loss:
0.2404 - acc: 0.9272
Epoch 64/200
3048/3048 [==============================] - 0s 138us/step - loss:
0.2459 - acc: 0.9242
Epoch 65/200
3048/3048 [==============================] - 0s 137us/step - loss:
0.2391 - acc: 0.9219
Epoch 66/200
3048/3048 [==============================] - 0s 158us/step - loss:
0.2301 - acc: 0.9245
Epoch 67/200
3048/3048 [==============================] - 0s 150us/step - loss:
0.2229 - acc: 0.9278
Epoch 68/200
3048/3048 [==============================] - 1s 166us/step - loss:
0.2190 - acc: 0.9281
Epoch 69/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.2085 - acc: 0.9308
Epoch 70/200
3048/3048 [==============================] - 0s 138us/step - loss:
0.2010 - acc: 0.9354
Epoch 71/200
3048/3048 [==============================] - 0s 161us/step - loss:
0.2288 - acc: 0.9232
Epoch 72/200
```

```
3048/3048 [==============================] - 0s 142us/step - loss:
0.1969 - acc: 0.9360
Epoch 73/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.1870 - acc: 0.9409
Epoch 74/200
3048/3048 [==============================] - 1s 175us/step - loss:
0.1925 - acc: 0.9400
Epoch 75/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.1816 - acc: 0.9429
Epoch 76/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.1643 - acc: 0.9501
Epoch 77/200
3048/3048 [==============================] - 0s 154us/step - loss:
0.1673 - acc: 0.9475
Epoch 78/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.1635 - acc: 0.9488
Epoch 79/200
3048/3048 [==============================] - 0s 142us/step - loss:
0.1597 - acc: 0.9514
Epoch 80/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.1529 - acc: 0.9518
Epoch 81/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.1560 - acc: 0.9544
Epoch 82/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.1480 - acc: 0.9570
Epoch 83/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.1472 - acc: 0.9551
Epoch 84/200
3048/3048 [==============================] - 0s 139us/step - loss:
0.1412 - acc: 0.9557
Epoch 85/200
3048/3048 [==============================] - 1s 168us/step - loss:
0.1460 - acc: 0.9518
Epoch 86/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.1377 - acc: 0.9610
Epoch 87/200
3048/3048 [==============================] - 1s 170us/step - loss:
0.1404 - acc: 0.9511
Epoch 88/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.1384 - acc: 0.9583
Epoch 89/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.1195 - acc: 0.9646
```

```
Epoch 90/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.1281 - acc: 0.9629
Epoch 91/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.1296 - acc: 0.9596
Epoch 92/200
3048/3048 [==============================] - 0s 156us/step - loss:
0.1151 - acc: 0.9659
Epoch 93/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.1064 - acc: 0.9701
Epoch 94/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.1137 - acc: 0.9669
Epoch 95/200
3048/3048 [==============================] - 0s 143us/step - loss:
0.1197 - acc: 0.9616
Epoch 96/200
3048/3048 [==============================] - 0s 151us/step - loss:
0.0981 - acc: 0.9728
Epoch 97/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.1034 - acc: 0.9698
Epoch 98/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.1051 - acc: 0.9682
Epoch 99/200
3048/3048 [==============================] - 0s 150us/step - loss:
0.1107 - acc: 0.9659
Epoch 100/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.1208 - acc: 0.9629
Epoch 101/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.0931 - acc: 0.9724
Epoch 102/200
3048/3048 [==============================] - 0s 145us/step - loss:
0.0871 - acc: 0.9744
Epoch 103/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0756 - acc: 0.9800
Epoch 104/200
3048/3048 [==============================] - 0s 148us/step - loss:
0.0770 - acc: 0.9777
Epoch 105/200
3048/3048 [==============================] - 0s 158us/step - loss:
0.1009 - acc: 0.9669
Epoch 106/200
3048/3048 [==============================] - 0s 130us/step - loss:
0.1014 - acc: 0.9675
Epoch 107/200
3048/3048 [==============================] - 0s 132us/step - loss:
```

```
0.0879 - acc: 0.9747
Epoch 108/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0727 - acc: 0.9813
Epoch 109/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.0699 - acc: 0.9793
Epoch 110/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0647 - acc: 0.9846
Epoch 111/200
3048/3048 [==============================] - 0s 159us/step - loss:
0.0624 - acc: 0.9797
Epoch 112/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.0757 - acc: 0.9790
Epoch 113/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.0664 - acc: 0.9803
Epoch 114/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.0675 - acc: 0.9826
Epoch 115/200
3048/3048 [==============================] - 0s 130us/step - loss:
0.0611 - acc: 0.9826
Epoch 116/200
3048/3048 [==============================] - 0s 155us/step - loss:
0.0750 - acc: 0.9780
Epoch 117/200
3048/3048 [==============================] - 1s 165us/step - loss:
0.0689 - acc: 0.9793
Epoch 118/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.0785 - acc: 0.9754
Epoch 119/200
3048/3048 [==============================] - 0s 159us/step - loss:
0.0583 - acc: 0.9836
Epoch 120/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0587 - acc: 0.9836
Epoch 121/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0544 - acc: 0.9833
Epoch 122/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.0686 - acc: 0.9767
Epoch 123/200
3048/3048 [==============================] - 1s 166us/step - loss:
0.0737 - acc: 0.9780
Epoch 124/200
3048/3048 [==============================] - 1s 166us/step - loss:
0.0658 - acc: 0.9800
Epoch 125/200
```

```
3048/3048 [==============================] - 1s 187us/step - loss:
0.0453 - acc: 0.9865
Epoch 126/200
3048/3048 [==============================] - 0s 133us/step - loss:
0.0590 - acc: 0.9823
Epoch 127/200
3048/3048 [==============================] - 0s 139us/step - loss:
0.0870 - acc: 0.9695
Epoch 128/200
3048/3048 [==============================] - 0s 139us/step - loss:
0.0532 - acc: 0.9859
Epoch 129/200
3048/3048 [==============================] - 0s 150us/step - loss:
0.0636 - acc: 0.9793
Epoch 130/200
3048/3048 [==============================] - 0s 130us/step - loss:
0.0491 - acc: 0.9843
Epoch 131/200
3048/3048 [==============================] - 0s 138us/step - loss:
0.0380 - acc: 0.9898
Epoch 132/200
3048/3048 [==============================] - 1s 165us/step - loss:
0.0379 - acc: 0.9908
Epoch 133/200
3048/3048 [==============================] - 1s 176us/step - loss:
0.0308 - acc: 0.9918
Epoch 134/200
3048/3048 [==============================] - 1s 168us/step - loss:
0.0370 - acc: 0.9895
Epoch 135/200
3048/3048 [==============================] - 1s 170us/step - loss:
0.0410 - acc: 0.9895
Epoch 136/200
3048/3048 [==============================] - 0s 164us/step - loss:
0.0345 - acc: 0.9918
Epoch 137/200
3048/3048 [==============================] - 0s 142us/step - loss:
0.0463 - acc: 0.9872
Epoch 138/200
3048/3048 [==============================] - 0s 130us/step - loss:
0.0873 - acc: 0.9675
Epoch 139/200
3048/3048 [==============================] - 1s 198us/step - loss:
0.0383 - acc: 0.9895
Epoch 140/200
3048/3048 [==============================] - 0s 148us/step - loss:
0.0402 - acc: 0.9882
Epoch 141/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.0613 - acc: 0.9816
Epoch 142/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0556 - acc: 0.9806
```

```
Epoch 143/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.0589 - acc: 0.9820
Epoch 144/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.0456 - acc: 0.9859
Epoch 145/200
3048/3048 [==============================] - 1s 175us/step - loss:
0.0447 - acc: 0.9849
Epoch 146/200
3048/3048 [==============================] - 0s 146us/step - loss:
0.0286 - acc: 0.9921
Epoch 147/200
3048/3048 [==============================] - 0s 128us/step - loss:
0.0235 - acc: 0.9944
Epoch 148/200
3048/3048 [==============================] - 0s 154us/step - loss:
0.0453 - acc: 0.9856
Epoch 149/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0215 - acc: 0.9951
Epoch 150/200
3048/3048 [==============================] - 0s 137us/step - loss:
0.0194 - acc: 0.9951
Epoch 151/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0390 - acc: 0.9865
Epoch 152/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.0750 - acc: 0.9767
Epoch 153/200
3048/3048 [==============================] - 0s 139us/step - loss:
0.0543 - acc: 0.9800
Epoch 154/200
3048/3048 [==============================] - 1s 168us/step - loss:
0.0243 - acc: 0.9944
Epoch 155/200
3048/3048 [==============================] - 0s 147us/step - loss:
0.0207 - acc: 0.9941
Epoch 156/200
3048/3048 [==============================] - 1s 175us/step - loss:
0.0179 - acc: 0.9961
Epoch 157/200
3048/3048 [==============================] - 1s 170us/step - loss:
0.0188 - acc: 0.9957
Epoch 158/200
3048/3048 [==============================] - 0s 148us/step - loss:
0.0150 - acc: 0.9974
Epoch 159/200
3048/3048 [==============================] - 0s 143us/step - loss:
0.0134 - acc: 0.9977
Epoch 160/200
3048/3048 [==============================] - 1s 165us/step - loss:
```

```
0.0523 - acc: 0.9826
Epoch 161/200
3048/3048 [==============================] - 0s 139us/step - loss:
0.0268 - acc: 0.9925
Epoch 162/200
3048/3048 [==============================] - 0s 161us/step - loss:
0.0488 - acc: 0.9826
Epoch 163/200
3048/3048 [==============================] - 0s 137us/step - loss:
0.0476 - acc: 0.9859
Epoch 164/200
3048/3048 [==============================] - 0s 159us/step - loss:
0.0742 - acc: 0.9747
Epoch 165/200
3048/3048 [==============================] - 1s 176us/step - loss:
0.0212 - acc: 0.9957
Epoch 166/200
3048/3048 [==============================] - 0s 143us/step - loss:
0.0105 - acc: 0.9980 0s - loss: 0.0113 - acc:
Epoch 167/200
3048/3048 [==============================] - 0s 137us/step - loss:
0.0105 - acc: 0.9984
Epoch 168/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0110 - acc: 0.9980
Epoch 169/200
3048/3048 [==============================] - 0s 150us/step - loss:
0.0091 - acc: 0.9997
Epoch 170/200
3048/3048 [==============================] - 0s 143us/step - loss:
0.0099 - acc: 0.9984
Epoch 171/200
3048/3048 [==============================] - 0s 157us/step - loss:
0.0094 - acc: 0.9984
Epoch 172/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0460 - acc: 0.9856
Epoch 173/200
3048/3048 [==============================] - 1s 183us/step - loss:
0.1034 - acc: 0.9692
Epoch 174/200
3048/3048 [==============================] - 1s 170us/step - loss:
0.0380 - acc: 0.9862
Epoch 175/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0180 - acc: 0.9954
Epoch 176/200
3048/3048 [==============================] - 0s 162us/step - loss:
0.0206 - acc: 0.9941
Epoch 177/200
3048/3048 [==============================] - 1s 170us/step - loss:
0.0078 - acc: 0.9990
Epoch 178/200
```

```
3048/3048 [==============================] - 0s 155us/step - loss:
0.0082 - acc: 0.9993
Epoch 179/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.0078 - acc: 0.9987
Epoch 180/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.0083 - acc: 0.9984
Epoch 181/200
3048/3048 [==============================] - 0s 159us/step - loss:
0.0118 - acc: 0.9977
Epoch 182/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0065 - acc: 0.9993
Epoch 183/200
3048/3048 [==============================] - 1s 165us/step - loss:
0.0174 - acc: 0.9948
Epoch 184/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0354 - acc: 0.9892
Epoch 185/200
3048/3048 [==============================] - 0s 135us/step - loss:
0.0977 - acc: 0.9678
Epoch 186/200
3048/3048 [==============================] - 0s 132us/step - loss:
0.0286 - acc: 0.9928
Epoch 187/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0098 - acc: 0.9984
Epoch 188/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0071 - acc: 1.0000
Epoch 189/200
3048/3048 [==============================] - 1s 165us/step - loss:
0.0079 - acc: 0.9990
Epoch 190/200
3048/3048 [==============================] - 0s 129us/step - loss:
0.0073 - acc: 0.9987
Epoch 191/200
3048/3048 [==============================] - 0s 134us/step - loss:
0.0248 - acc: 0.9928
Epoch 192/200
3048/3048 [==============================] - 0s 136us/step - loss:
0.0656 - acc: 0.9760
Epoch 193/200
3048/3048 [==============================] - 0s 131us/step - loss:
0.0689 - acc: 0.9764
Epoch 194/200
3048/3048 [==============================] - 1s 181us/step - loss:
0.0370 - acc: 0.9879
Epoch 195/200
3048/3048 [==============================] - 0s 161us/step - loss:
0.0120 - acc: 0.9977
```

```
Epoch 196/200
3048/3048 [==============================] - 0s 138us/step - loss:
0.0069 - acc: 0.9993
Epoch 197/200
3048/3048 [==============================] - 1s 186us/step - loss:
0.0044 - acc: 1.0000
Epoch 198/200
3048/3048 [==============================] - 0s 140us/step - loss:
0.0040 - acc: 1.0000
Epoch 199/200
3048/3048 [==============================] - 0s 137us/step - loss:
0.0036 - acc: 1.0000
Epoch 200/200
3048/3048 [==============================] - 1s 167us/step - loss:
0.0034 - acc: 1.0000
```

```
Out[61]: <keras.callbacks.History at 0x7f3329fbfdd8>
```

```
In [62]: model3.evaluate(xTest2,yTestOHE2)
```

```
762/762 [==============================] - 0s 94us/step
```

```
Out[62]: [0.37475377196089177, 0.9120734901878776]
```

With the tuned and topology changes it was achieved a 90.15% accuraccy.

Model | Random Forest 2

Same as the previous, we adjust model shape and tune parameters to improve the performance.

In [63]:
```
folds2 = StratifiedKFold(n_splits=10, shuffle=True, random_state=60
)
predicted2 = np.zeros((xTest.shape[0],9))
measured2= np.zeros((Input.shape[0]))
score2 = 0

for times, (trn_idx, val_idx) in enumerate(folds2.split(Input2.valu
es,Y_data2['surface'].values)):
    model4 = RandomForestClassifier(n_estimators=700, max_depth=20,
min_samples_split=5, n_jobs=-1)
    model4.fit(Input2.iloc[trn_idx],Y_data2['surface'][trn_idx])
    measured2[val_idx] = model4.predict(Input2.iloc[val_idx])
    predicted2 += model4.predict_proba(xTest2)/folds2.n_splits
    score2 += model4.score(Input2.iloc[val_idx],Y_data2['surface'][
val_idx])
    print("Fold: {} score: {}".format(times,model4.score(Input2.ilo
c[val_idx],Y_data2['surface'][val_idx])))

    gc.collect()
```

```
Fold: 0 score: 0.9064935064935065
Fold: 1 score: 0.9140625
Fold: 2 score: 0.9112271540469974
Fold: 3 score: 0.9293193717277487
Fold: 4 score: 0.8792650918635171
Fold: 5 score: 0.8871391076115486
Fold: 6 score: 0.916010498687664
Fold: 7 score: 0.9023746701846965
Fold: 8 score: 0.9259259259259259
Fold: 9 score: 0.9095744680851063
```

In [64]:
```
print('Average score', score2 / folds2.n_splits)
```

```
Average score 0.908139229463
```

The final accuracy with tuned parameters and additional data features the accuracy reached 90.60%

In [65]:
```
confusion_matrix(measured2,Y_data2['surface'])
```

Out[65]:
```
array([[160,   6,   0,   0,   3,   2,   0,   3,   1],
       [ 13, 707,  12,   0,  16,  13,   5,  22,  15],
       [  0,   6, 323,   0,   3,   6,   1,   0,   5],
       [  0,   0,   0,  13,   0,   0,   0,   0,   0],
       [  0,   5,   1,   0, 278,   5,   0,   4,   1],
       [  3,  19,   6,   1,   1, 684,   6,   4,  21],
       [  2,   7,   0,   1,   0,   6, 279,   7,   0],
       [  1,  11,   3,   0,   0,   3,   0, 468,  16],
       [ 10,  18,  18,   6,   7,  13,   6,   6, 548]])
```

In [66]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
sns.heatmap(pd.DataFrame(confusion_matrix(measured2,Y_data2['surfac
e'])),
            ax = ax,
            cmap = 'coolwarm',
            annot = True,
            fmt = '.2f',
            linewidths = 0.05)
fig.subplots_adjust(top=0.93)
fig.suptitle('Confusion matrix, Actual vs Predicted label Correlati
on Heatmap',
             fontsize=14,
             fontweight='bold')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[66]: `Text(0.5,24,'Predicted label')`