In [1]:
```
#***********************************************
#WARNING : RUN ON PYTHON3 OTHERWISE SOME MODULES MAY NOT LOAD (DEPRICATE
D)
#WARNING : RUN ON PYTHON3 OTHERWISE SOME MODULES MAY NOT LOAD (DEPRICATE
D)
#WARNING : RUN ON PYTHON3 OTHERWISE SOME MODULES MAY NOT LOAD (DEPRICATE
D)
#***********************************************




import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.rc('axes', labelsize=20)
%matplotlib inline
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/IPython/html.py:14: ShimWarning: The `IPython.html` p
ackage has been deprecated. You should import from `notebook` instead.
 `IPython.html.widgets` has moved to `ipywidgets`.
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

In [2]:
```
# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix,recall_score,precision_reca
ll_curve,auc,roc_curve,roc_auc_score,classification_report
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: T
his module was deprecated in version 0.18 in favor of the model_selecti
on module into which all the refactored classes and functions are move
d. Also note that the interface of the new CV iterators are different f
rom that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Overview

(i) Section 1: This challenge is divided in two sections. In section 1, I first navigate throughout data to find important features (Section 1) to reduced the dimensionality of the data set.

(ii) Section 1: create and cross-validate Random Forrest learning model (>80% precision scores for response =1 class, model performs really well for response = 0 class with precision/recall > 0.90)

(iii) Section 2: Performed PCA decomposition on these features to further gained information on the data (Section 2)

(iv) Section 2: Finally, create and cross-validate Support Vector Machine to improve precision (~94%) for response =1 after some hyperdimensional tuning , while preserving a good prediction performance for response = 0 class.

Section 1: Data Analysis and ML testing

In [3]:
```
#upload data file
all_data = pd.read_csv("takehome1.csv",delimiter ='\t')
```

First We Need To create a Training and Test Data Frames

(i) Create Test and train files (ii) We need to keep locked away the Test data until we build Machine Learning model (avoid data snooping).

In [4]:
```
#Create training and test Data Frame 70/30 ratio
train=all_data.sample(frac=0.7,random_state=200)
test=all_data.drop(train.index)
```

In [5]:
```
# let's get a feeling for ALL Data
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 530 entries, 0 to 529
Columns: 16563 entries, response to V16562
dtypes: int64(16563)
memory usage: 67.0 MB
```

Comments

(i) Data is pretty clan (no NaN). (ii) Notice the great amount of features vs. sample size : We need to really reduced the dimensionality!!

In [6]:
```
#train = all_data.sample(frac=0.7)
```

In [5]:
```
#Let's look into a description of the file
all_data.describe()
```

Out[5]:

	response	V1	V2	V3	V4	V5	V6	V7
count	530.000000	530.000000	530.000000	530.000000	530.0	530.000000	530.0	530.000
mean	0.232075	0.009434	0.009434	0.001887	0.0	0.020755	1.0	0.00188
std	0.422556	0.096761	0.096761	0.043437	0.0	0.142697	0.0	0.04343
min	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
25%	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
50%	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
75%	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
max	1.000000	1.000000	1.000000	1.000000	0.0	1.000000	1.0	1.00000

8 rows × 16563 columns

Comments on Data description

(i) Data seems to be clean.

(ii)Entries are 0's or 1's. (This columns are not useful to create a (M)achine (L)earning model

(iii) Data is a bit off balance (mean of response values) = 0.23)

(iv) Some columns are all 1's or 0's (We can get rid of this column)

Let's drop columns that have all 1's and 0's. (They do not provide any information gain)

In [5]:
```
#get Columns that are useless
# and erase them

#Get Columns Names
col_names = train.columns.astype('str')

#Extract Columns with variance = 0.0
tmp_col = []

for col in col_names:
    col = str(col)
    #get std
    std = train[col].std()
    # std = 0 means all column values are =0's or 1's

    mean = train[col].mean()
    #if std == 0 drop column
    if std < 1e-6 and mean < 1e-6:
        tmp_col.append(col)

print("Number of useless columns: " + str(len(tmp_col)))
#Drop Useless Columns
reduced_all_train = train.drop(tmp_col,axis=1)
```

```
Number of useless columns: 6256
```

In [34]:
```
#descripion of new reduced data file
reduced_all_train.describe()
```

Out[34]:

	response	V1	V2	V3	V5	V6	V7	V8
count	371.000000	371.000000	371.000000	371.000000	371.000000	371.0	371.000000	37
mean	0.250674	0.013477	0.005391	0.002695	0.021563	1.0	0.002695	0.
std	0.433986	0.115462	0.073323	0.051917	0.145449	0.0	0.051917	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.0	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	1.0	0.000000	0.
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.0	0.000000	0.
75%	0.500000	0.000000	0.000000	0.000000	0.000000	1.0	0.000000	0.
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.0	1.000000	1.

8 rows × 10307 columns

Feature Selection

(i) We know features are 0's 1's. We can divide our training set into a positive response set (all response == 1) and and a negative response set (all response = 0). (ii) Then look how the positive and negative response sets correlate to a single feature. (iii) Because features are columns of 1's and 0's, we can measure the (positive/negative) response-feature correlation with mean value. (iv) Feature ranking could be done by selecting those features for which the difference between positive-and-negative response correlations is above some threshold (As it will be shown later) (v) Data visualization (heat map) was not that helpful (It is costly to map all columns, so I may need more time)

In [6]:
```
# Select positive and negative response sets

all_positive = reduced_all_train[reduced_all_train['response'] == 1 ]
all_negative = reduced_all_train[reduced_all_train['response'] == 0 ]
```

In [8]:
```
all_positive.describe()
```

Out[8]:

	response	V1	V2	V3	V5	V6	V7	V8	V11
count	93.0	93.000000	93.000000	93.0	93.000000	93.0	93.0	93.000000	93.000000
mean	1.0	0.021505	0.010753	0.0	0.010753	1.0	0.0	0.043011	0.010753
std	0.0	0.145848	0.103695	0.0	0.103695	0.0	0.0	0.203981	0.103695
min	1.0	0.000000	0.000000	0.0	0.000000	1.0	0.0	0.000000	0.000000
25%	1.0	0.000000	0.000000	0.0	0.000000	1.0	0.0	0.000000	0.000000
50%	1.0	0.000000	0.000000	0.0	0.000000	1.0	0.0	0.000000	0.000000
75%	1.0	0.000000	0.000000	0.0	0.000000	1.0	0.0	0.000000	0.000000
max	1.0	1.000000	1.000000	0.0	1.000000	1.0	0.0	1.000000	1.000000

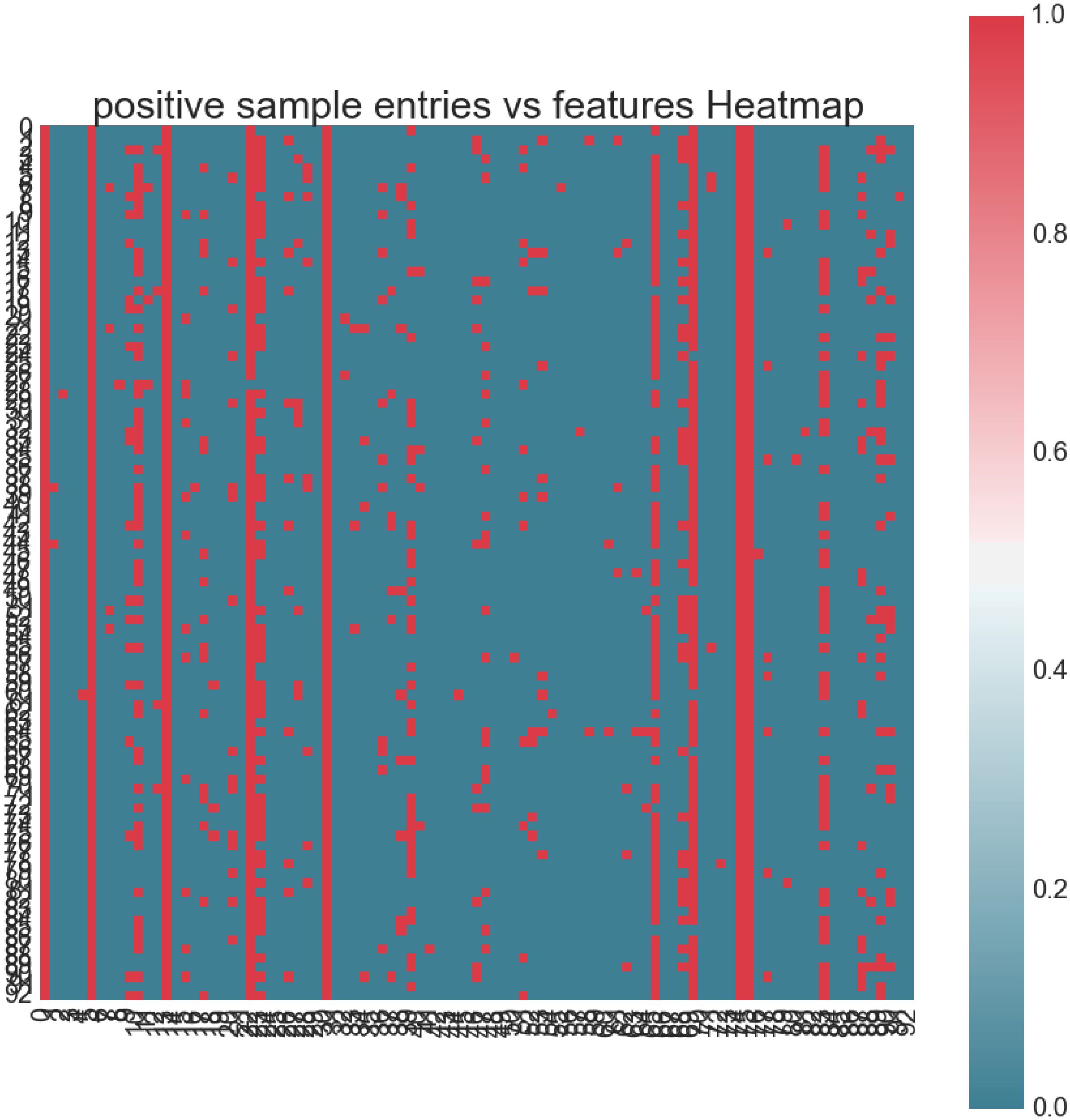
8 rows × 10307 columns

Lets use Heatmap To visualize sections of the Positive/negative response data sets We want to find relevant features for our model

In [7]:
```
f, ax = plt.subplots(figsize=(15,15))

ax.set_title('positive sample entries vs features Heatmap')

ax.title.set_fontsize(30)
sns.heatmap(all_positive.values[:100,:len(all_positive)], cmap=sns.diver
ging_palette(220, 10, as_cmap=True),
             square=True, ax=ax)
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```

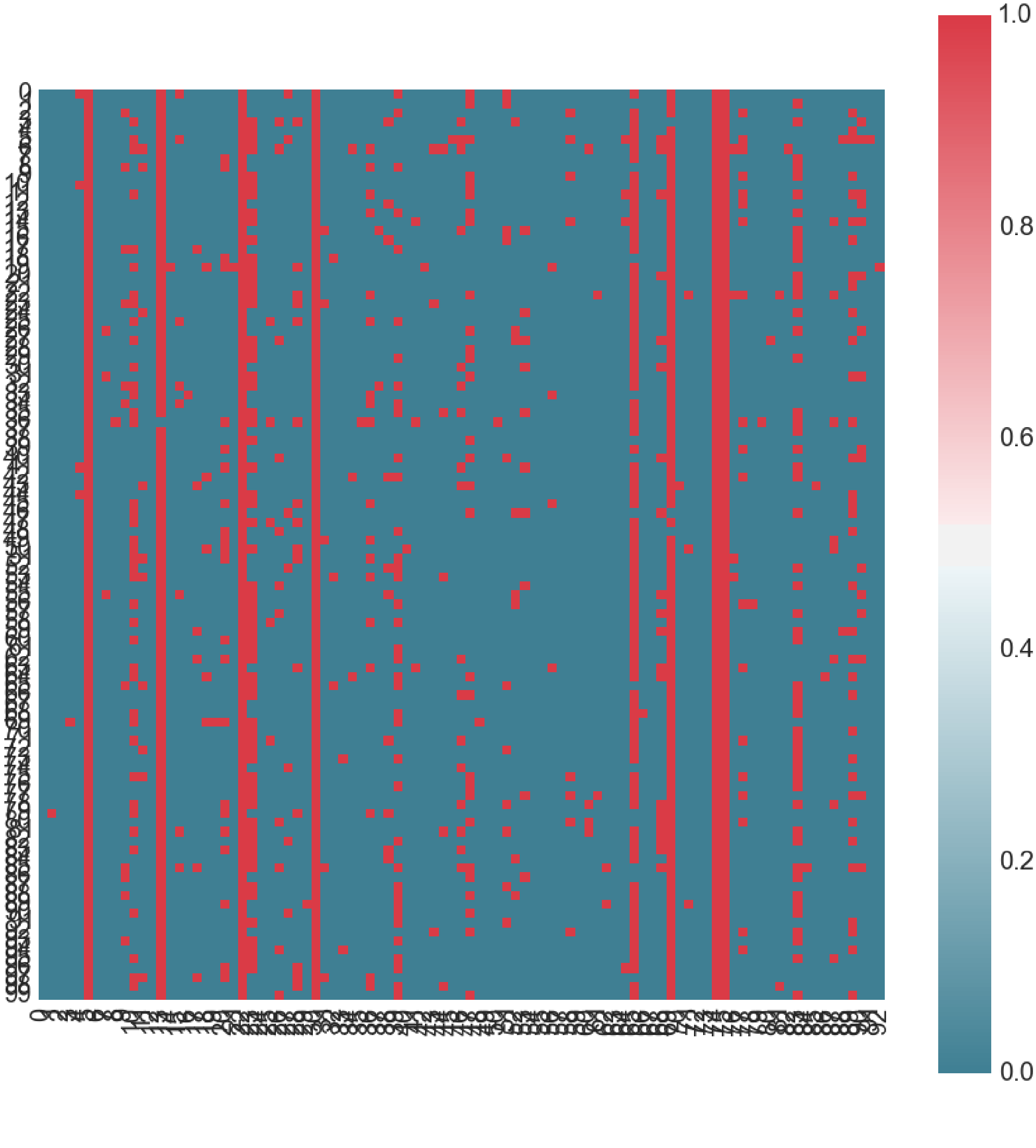
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x10b6997b8>



In [8]:
```
f, ax = plt.subplots(figsize=(15,15))
sns.heatmap(all_negative.values[:100,:len(all_positive)], cmap=sns.diver
ging_palette(220, 10, as_cmap=True),
             square=True, ax=ax)
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x10f99ec88>



IT's hard to extract much information from heat map !!

Let's look at how features correlate. One could disregard those features that correlate)

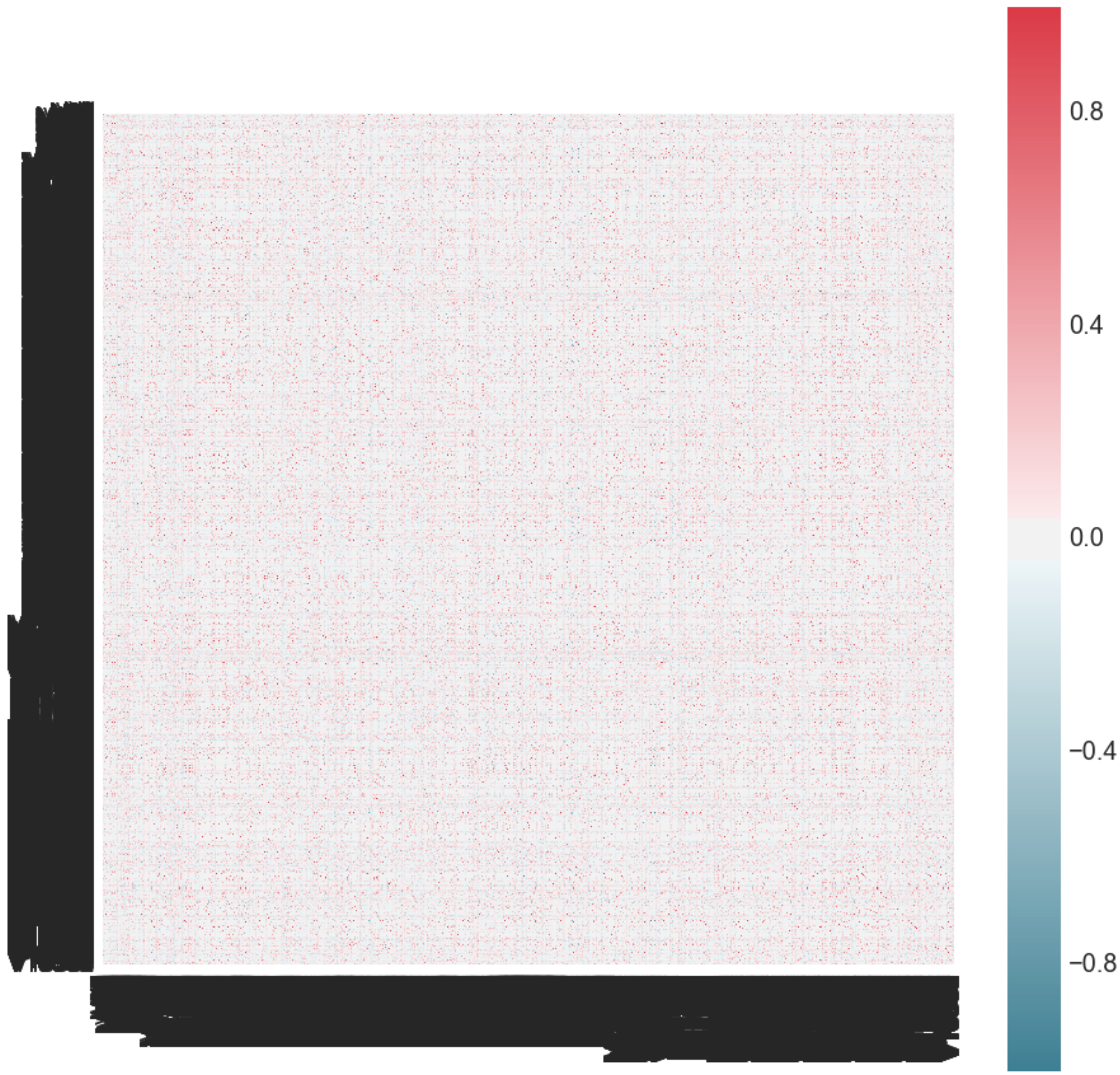
In [8]:
```
# First get rid of response column in both positive/negative response da
ta sets
all_positive = all_positive.drop('response',axis=1)
all_negative = all_negative.drop('response',axis=1)
```

In [9]:

```
#Let's visualize correlations of features for the positive response data
 set
f, ax = plt.subplots(figsize=(15,15))
corr = all_positive[:].corr()

sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.dive
rging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x10b8bacc0>

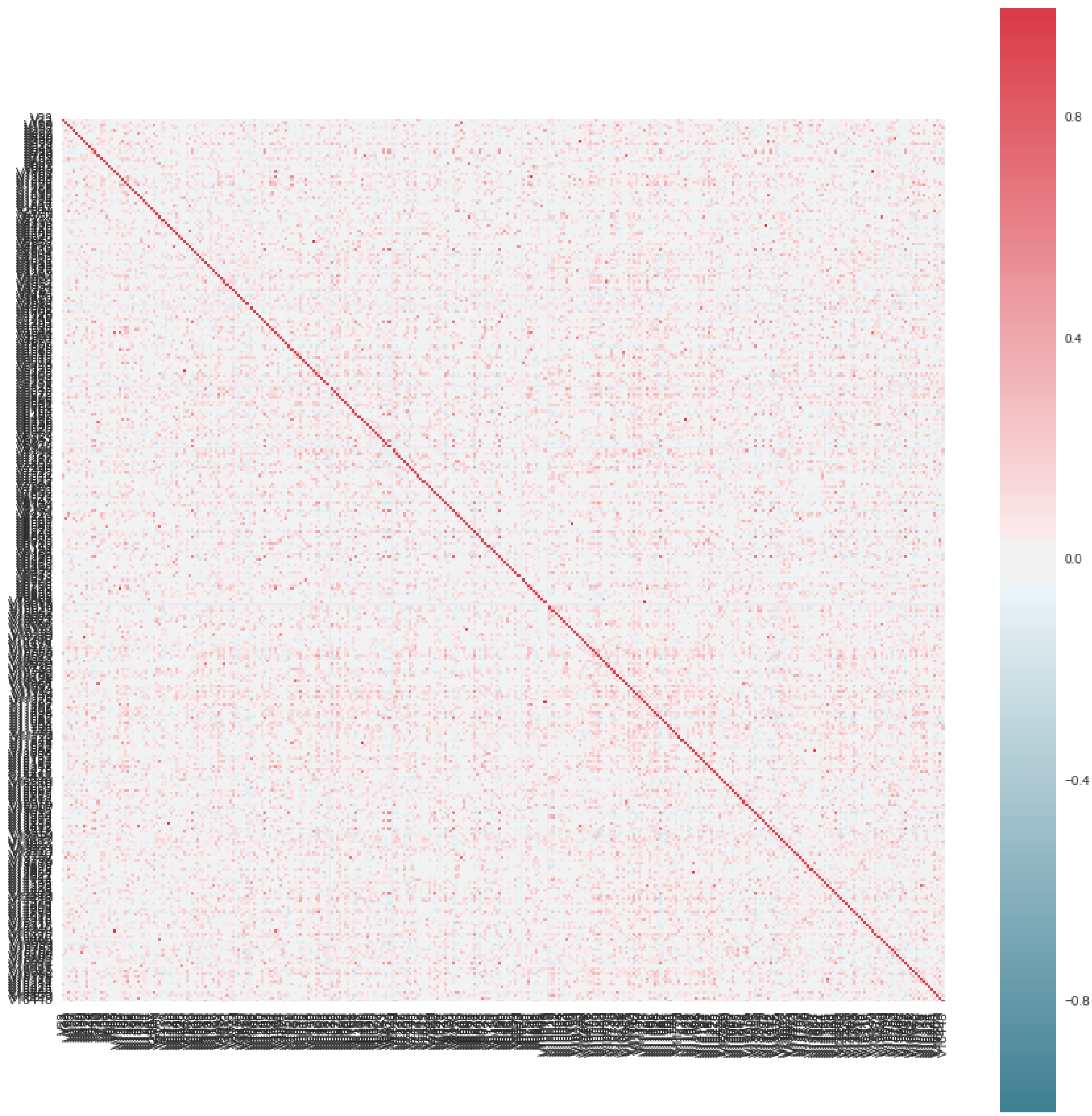


In [ ]:

In [393]:
```
#Let's visualize correlations of features for the negative response data
 set

f, ax = plt.subplots(figsize=(15,15))
corr = new_all_negative[:].corr()

sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.dive
rging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```

Out[393]: <matplotlib.axes._subplots.AxesSubplot at 0x118a2d828>



Let's try something else!

Let's calculate the mean value for a specific feature (remember they are all 0's and 1's values)for both the positive response(response=1) and negative (response=0) response data sets. If for a specific feature, one mean values is > 0.5 and the other mean is < 0.5 or vice versa, we keep that specific feature for our learning model development. This process helped me to select 169 features as shown below.

In [8]:
```
#list to track relevant features
feature_list = []
#get columns names
index_list = all_positive.columns

size = len(index_list)


diff = 100

#list files for plotting
tmp_diff=[]
max_diff=[]


#loop through columns
for j in index_list[:]:
    Calculate = False

    #get positive/negative mean values for a column
    a=  all_positive[j].values
    b = all_negative[j].values
    mean_a = np.mean(a)
    mean_b = np.mean(b)

    #disregard small values

    if mean_a < 1e-6 or mean_b < 1e-6:
        continue;

    #check if one mean is > 0.5 and the other one < 0.5 and vice versa
    if mean_a > 0.5 and mean_b < 0.5:
        Calculate = True ;

    if mean_a < 0.5 and mean_b > 0.5:
        Calculate = True ;

    #if not skip to next feature
    if not Calculate:
        continue;

    #get std's
    std_a = np.std(a);
    std_b = np.std(b);
    ab_diff = abs((mean_a-mean_b)/mean_a)
    ab_diff *=100
```

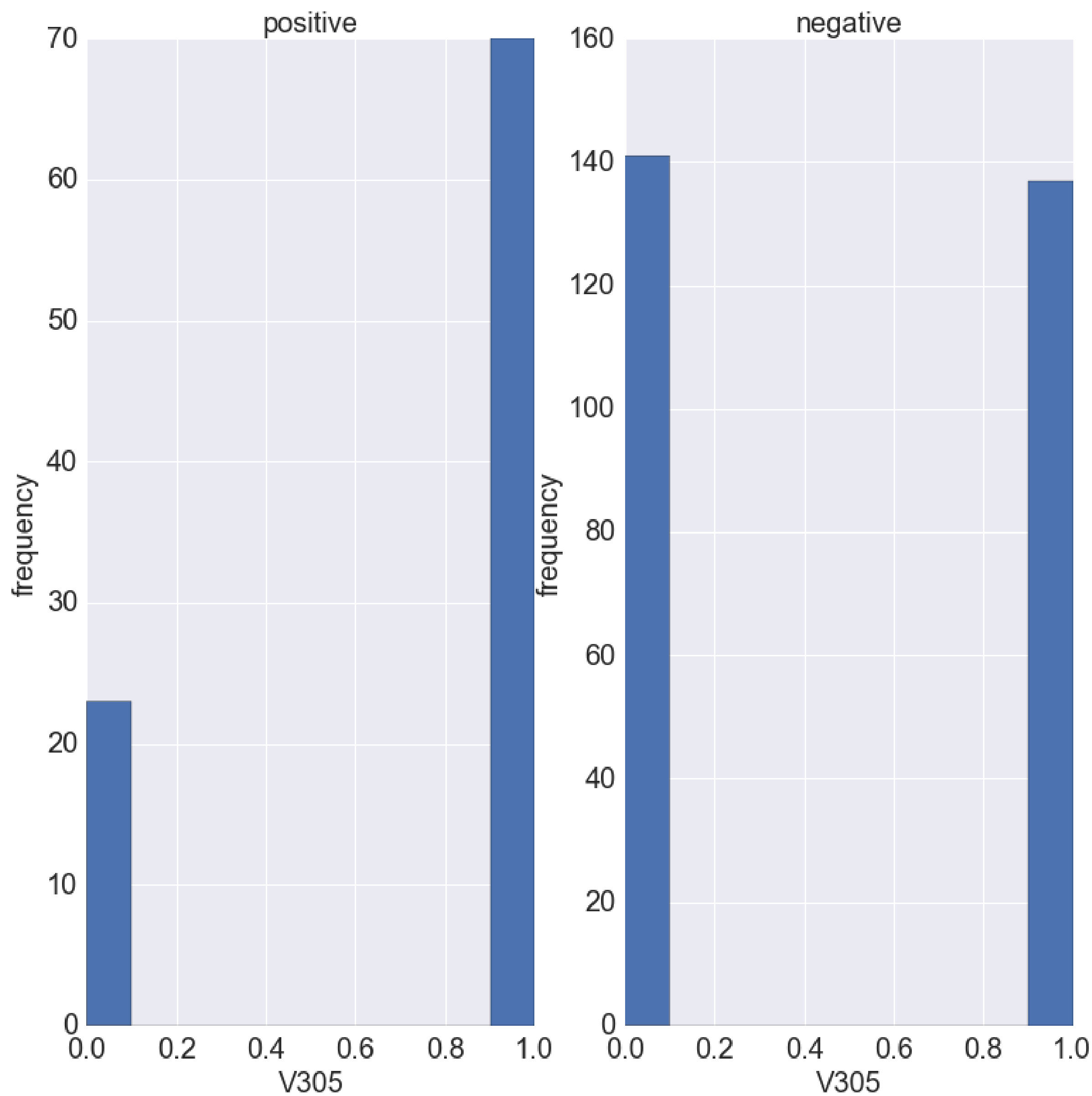
```
    # if so include feature
    if Calculate :

        tmp_diff.append([ab_diff,mean_a,mean_b])
        feature_list.append(j)
        max_diff.append([mean_a,mean_b,std_a,std_b])

max_diff = np.array(max_diff)
#tmp_diff =np.array(tmp_diff)

print('number of important features',len(feature_list))

plt.figure(figsize=(10,10))
plt.title('Some relevant Features',size=20)
plt.ylabel('mean value difference')
plt.xlabel('Features')


#let's plot 15 of these features for visualization!
num_features = 15
max_diff[:2,0].shape

plt.plot(max_diff[:num_features,0],marker='o',label='positive response')
plt.plot(max_diff[:num_features,1],marker='o',label='negative
response',color='red')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,ncol=2, mode="expa
nd", borderaxespad=0.)
```

```
number of important features 169
```

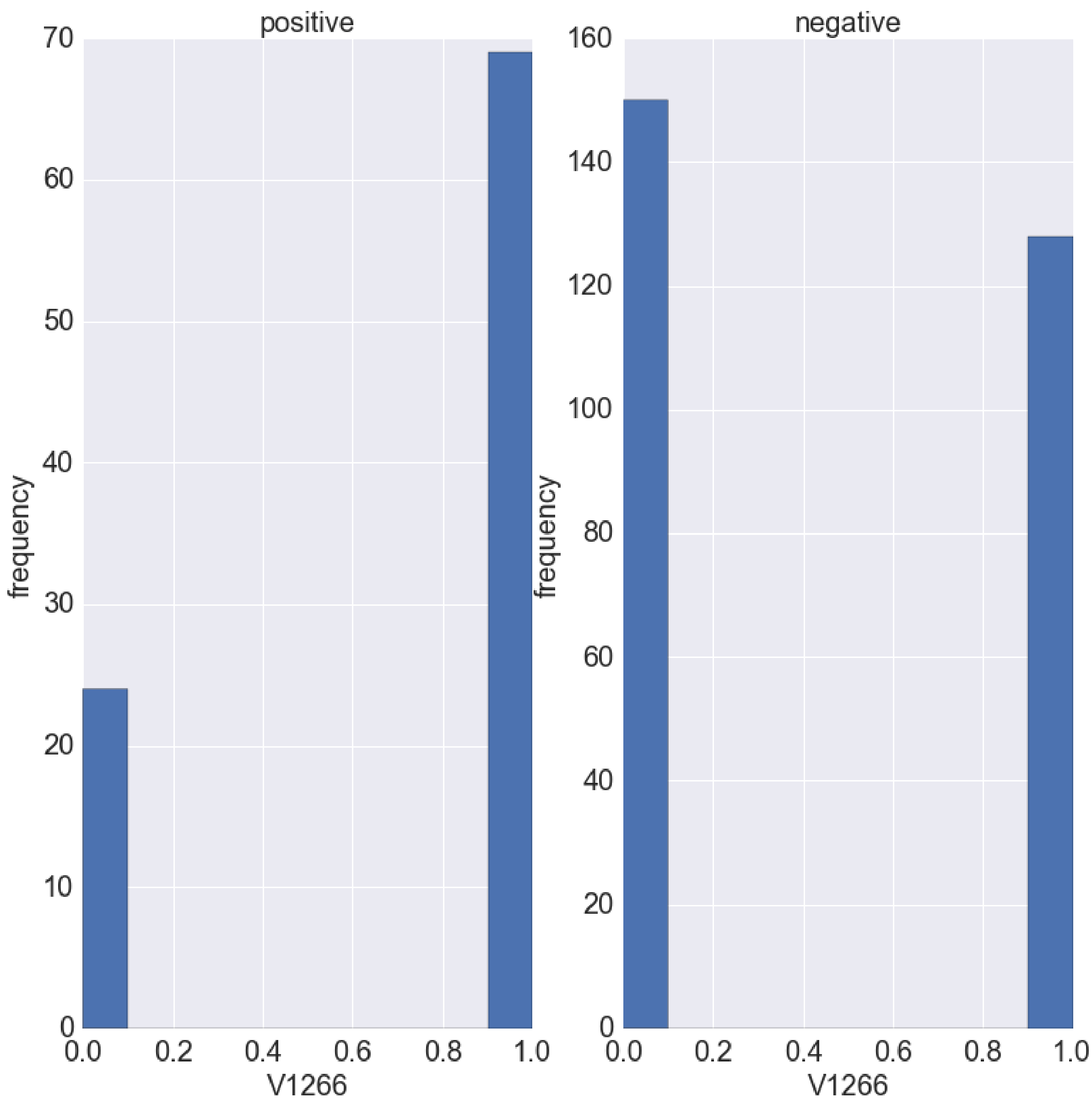
```
Out[8]: <matplotlib.legend.Legend at 0x11a095cc0>
```

Histograms of features in the positive and negative data sets should show that 1's are majority in one data set while 0's are a majority in the other set

In [23]:
```
#let;s plot the histograms of one feature
col_tr = feature_list[2]
plt.figure(1,figsize=(12.5,12.5))
plt.subplot(121)
plt.title('positive',size=20)
plt.ylabel('frequency')
plt.xlabel(col_tr)
plt.hist(all_positive[col_tr].values)
plt.subplot(122)
plt.title('negative',size=20)
plt.ylabel('frequency')
plt.xlabel(col_tr)
freq =plt.hist(all_negative[col_tr].values)
```



In [24]:
```
col_tr = feature_list[8]
plt.figure(1,figsize=(12.5,12.5))
plt.subplot(121)
plt.title('positive',size=20)
plt.ylabel('frequency')
plt.xlabel(col_tr)
plt.hist(all_positive[col_tr].values)
plt.subplot(122)
plt.title('negative',size=20)
plt.ylabel('frequency')
plt.xlabel(col_tr)
freq = plt.hist(all_negative[col_tr].values)
```



Ok, Now us these features to create ML model

In [20]:
```
#include response feature to selected features
feature_list.append('response')
```

Small routine for oversampling/undersampling. Oversampling or Undersamping did not show a signifcant improvement

In [28]:
```
positive_train_ix =train[train.response == 1]
negative_train_ix = train[train.response == 0]
rand_negative_train = negative_train_ix.sample(frac=1.0)
print(len(negative_train_ix))
print(len(positive_train_ix))

under_train = rand_negative_train.append(positive_train_ix)
#under_train = under_train.append(positive_train_ix.sample(frac=1.0))

print(len(under_train))
```

```
278
93
371
```

In [29]:
```
#Create the training and testing sets


under_train = under_train[feature_list]
X_train = pd.DataFrame(under_train)

#X_train = X_train.sample(frac=0.42)
#under_train = under_train.drop("response", axis=1)
Y_train = under_train["response"]
X_train = X_train.drop("response", axis=1)
test = test[feature_list]

X_test = test.drop("response", axis=1)
Y_test = test["response"]
```

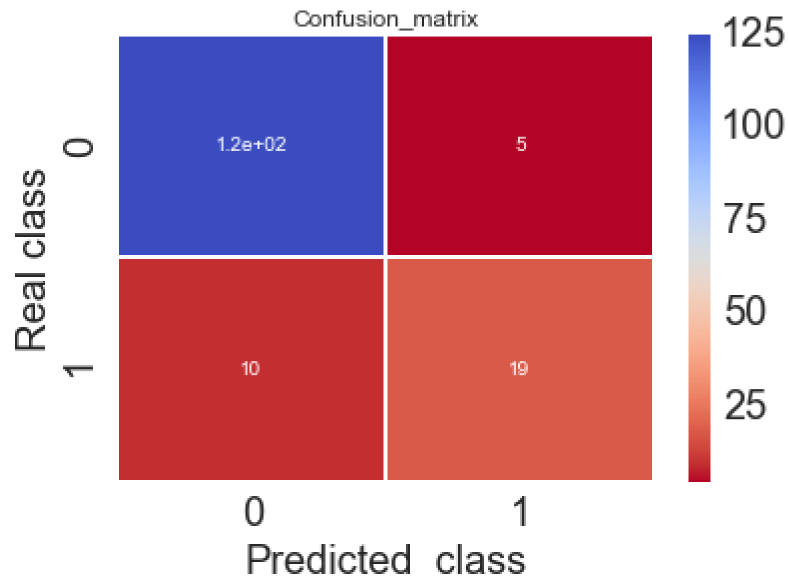
Let's Use Random Forrest:

(i) This is an ensemble approach based on a majority vote decision trees prepared in different fashion with different sample portions and features to overcome overfitting. This approach is flexible, it does well with multiple features and small data sets.

In [374]:

```
random_forest = RandomForestClassifier(n_estimators=400,n_jobs=10,max_fe
atures=30)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100,
2)

cnf_matrix=confusion_matrix(Y_test,Y_pred)
print("score for test data",acc_random_forest)
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_class")
plt.ylabel("Real class")
plt.show()
print("\n----------Classification Report--------------------------------
----")
print(classification_report(Y_test,Y_pred))


Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
print("score for training data",acc_random_forest)
```

```
score for test data 100.0

/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

          0       0.93      0.96      0.94       130
          1       0.79      0.66      0.72        29

avg / total       0.90      0.91      0.90       159

score for training data 90.57
```

Let's cross validate with different sets!

In [373]:

```
from sklearn.model_selection import cross_val_score
#linear_svc = LinearSVC()
random_forest = RandomForestClassifier(n_estimators=400,n_jobs=10,max_fe
atures=30)
#linear_svc = LinearSVC(cl)



scores = cross_val_score(random_forest,X_train, Y_train, cv=3)
print("score mean value: ",scores.mean())
print("std of scores: ",scores.std())
```

```
score mean value:  0.873409214092
std of scores:  0.0262933929656
```

High scores and close to the test (data) score !

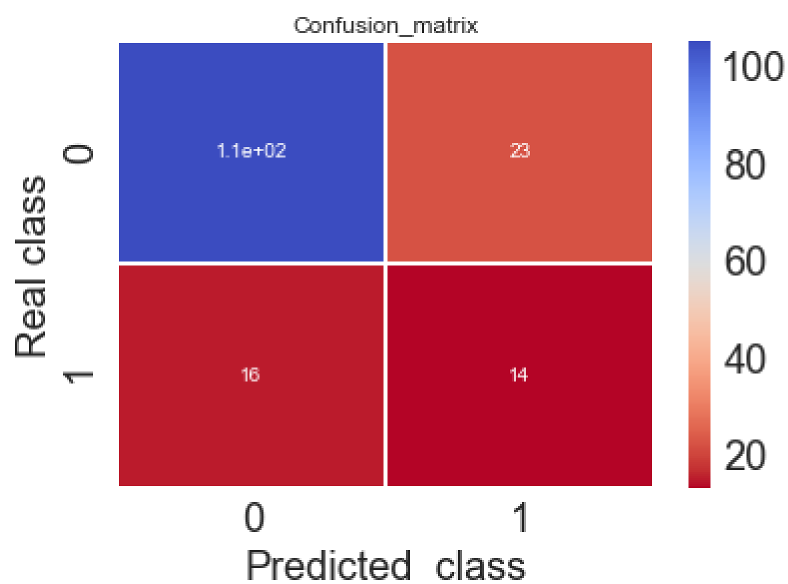
Comments

The model works really well to identify and properly label the 0's response class. The prediction performance is not bad : 80% precision and ~66% recall for response = 1 class (Remember the response=1 class is a minority in our slightly off-balance data set). One could do some hyperdimensional parameters to improve on the the model performance.

Let's give it a try to the Support Vector Machine model

In [46]:
```
linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)
Y_pred = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
cnf_matrix=confusion_matrix(Y_test,Y_pred)
print(acc_log)
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_class")
plt.ylabel("Real class")
plt.show()
print("\n----------Classification Report------------------------------------
----")
print(classification_report(Y_test,Y_pred))
```

```
97.04

/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

          0       0.87      0.82      0.84       129
          1       0.38      0.47      0.42        30

avg / total       0.78      0.75      0.76       159
```

Performance does not look that Promising for SVM!

Section 2. PCA Decomposition

Now that we have identified the important features, let's use PCA to further reduced the dimensionality in our approach. This approach identifies new dimensions and ranks them according the information they provide regarding data sparsity. In the following I will further reduce the dimensionality and use SVM to have a better precision ~ 90%(for the response = 1) in our model.

In [38]:
```
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
```

Ok just like we did in Section 1 we upload the data file and clean it a bit (Same steps)

In [10]:
```
#upload data file
all_data = pd.read_csv("takehome1.csv",delimiter ='\t')
```

In [27]:
```
#Create training and test Data Frame 70/30 ratio
train=all_data.sample(frac=0.7,random_state=120)
test=all_data.drop(train.index)
#cv_test=all_data.drop(train.index)
#test = cv_test.drop(cv_df.index)
```

In [33]:
```
train.describe()
```

Out[33]:

	response	V1	V2	V3	V4	V5	V6	V7
count	371.000000	371.000000	371.000000	371.000000	371.0	371.000000	371.0	371.000
mean	0.253369	0.008086	0.013477	0.002695	0.0	0.018868	1.0	0.00269
std	0.435528	0.089680	0.115462	0.051917	0.0	0.136242	0.0	0.05191
min	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
25%	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
50%	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
75%	1.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.0	0.00000
max	1.000000	1.000000	1.000000	1.000000	0.0	1.000000	1.0	1.00000

8 rows × 16563 columns

Get rid of useless columns with only 0's or 1's !!

In [12]:
```
#get Columns that are useless
# and erase them
#Get Columns Names
col_names = train.columns.astype('str')

#Extract Columns with variance = 0.0
tmp_col = []

for col in col_names:
    col = str(col)
    #get std
    std = train[col].std()
    # std = 0 means all column values are =0's or 1's

    if std < 1e-6:
        tmp_col.append(col)

print("Number of useless columns" + str(len(tmp_col)))
#Drop Useless Columns
train= train.drop(tmp_col,axis=1)

test = test.drop(tmp_col,axis=1)
```

```
Number of useless columns6397
```

Oversampling and Undersampling routine that did not help much.

In [23]:
```
positive_train_ix =train[train.response == 1]
negative_train_ix = train[train.response == 0]
rand_negative_train = negative_train_ix.sample(frac=1.0)
print(len(negative_train_ix))
print(len(positive_train_ix))

under_train = rand_negative_train.append(positive_train_ix)
#under_train = under_train.append(positive_train_ix.sample(frac=1.0,rand
om_state=200))
#under_train = under_train.append(positive_train_ix.sample(frac=0.75,ran
dom_state=50))
#under_train = under_train.append(positive_train_ix.sample(frac=0.2,rand
om_state=22))
#nbr_pos = int(len(pos_indices))
#times = 1
#rand_neg_indices = np.array(np.random.choice(neg_indices,(times* nbr_po
s),replace= False))
#under_sample_indices = np.concatenate((rand_neg_indices, pos_indices))
total_negative = len( under_train[under_train.response == 0])
total_positive = len( under_train[under_train.response == 1])
print("ratio positives to negatives: ",total_positive/total_negative)
#print(neg_indices)
#under_sample_train = train.iloc[neg_indices,:]
print(len(under_train))
```

```
277
94
ratio positives to negatives:  0.33935018050541516
371
```

Now use the 169 features stored in feature_list and found in Section 1 after data analysis

In [26]:
```
test.head()
```

Out[26]:

	V17	V140	V305	V439	V584	V655	V810	V947	V1266	V1298	...	V15417	V15487	V1
4	1	1	1	1	1	1	1	1	0	0	...	1	0	1
10	1	1	1	1	0	1	1	1	1	0	...	1	1	1
11	0	1	1	1	1	0	1	1	0	0	...	0	0	0
15	1	0	1	1	0	1	1	1	1	0	...	1	0	0
21	1	1	1	0	0	0	1	1	1	0	...	1	1	1

5 rows × 169 columns

In [29]:
```
#Use the relevant features found in section 1
under_train = under_train[feature_list]
test = test[feature_list]
```

In [30]:
```
X_train = under_train.drop('response',axis=1)
Y_train = under_train['response']
X_train.shape
```

Out[30]:
```
(371, 169)
```

In [31]:
```
X_test = test.drop('response',axis=1)
Y_test = test['response']
X_test.shape
```

Out[31]:
```
(159, 169)
```

In [32]:
```
#Center and Rescale features by the mean
#This is good practice!

#Find mean value of data
X = np.array(X_train.values)
X_test = np.array(X_test.values)
means = np.mean(X,axis=0)

#Center Data
X = X-means
X_test = X_test-means
#Get the std of features and use it to normalized
stds = np.std(X,axis=0)

#Normalized features
inv_stds = 1./stds
X = X * inv_stds
X_test = X_test * inv_stds
#X_cv = X_cv * inv_stds
```

Performed PCA Decomposition

In [33]:
```
pca = PCA(n_components= 50)
pca.fit(X)
```

Out[33]:
```
PCA(copy=True, iterated_power='auto', n_components=50, random_state=Non
e,
  svd_solver='auto', tol=0.0, whiten=False)
```

Some benchmarking with n_components is need for the future. Map features to the new 50 dimensions

In [34]:
```
trans_X_train = pca.transform(X)
trans_X_train.shape
```

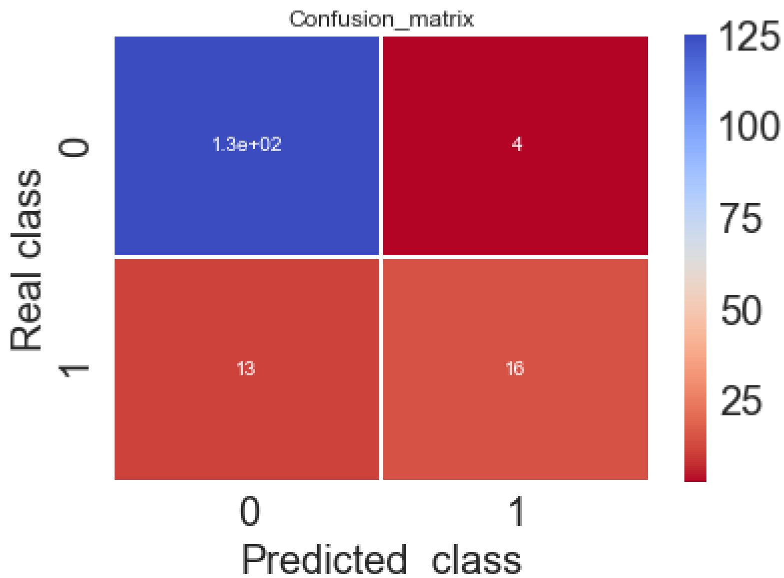
Out[34]:
```
(371, 50)
```

In [35]:
```
trans_X_test = pca.transform(X_test)
trans_X_test.shape
```

Out[35]: (159, 50)

In [36]:
```
random_forest = RandomForestClassifier(n_estimators=1000,n_jobs=10,max_f
eatures=40)
random_forest.fit(trans_X_train, Y_train)
Y_pred = random_forest.predict(trans_X_test)
random_forest.score(trans_X_train, Y_train)
acc_random_forest = round(random_forest.score(trans_X_train, Y_train) *
100, 2)

cnf_matrix=confusion_matrix(Y_test,Y_pred)
print(acc_random_forest)
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_class")
plt.ylabel("Real class")
plt.show()
print("\n----------Classification Report------------------------------------
----")
print(classification_report(Y_test,Y_pred))


Y_pred = random_forest.predict(trans_X_test)
random_forest.score(trans_X_test, Y_test)
acc_random_forest = round(random_forest.score(trans_X_test, Y_test) * 10
0, 2)
print(acc_random_forest)
```

```
100.0

/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

          0       0.91      0.97      0.94       130
          1       0.80      0.55      0.65        29

avg / total       0.89      0.89      0.89       159

89.31
```

I did not observed much improvement with Random Forrest Approach.

Let's Try SVM: SVM tries to find a hyperplane that 'best' separate the data classes in space. IT works well wihgh higher dimensional data. First some cross-validation to see scores

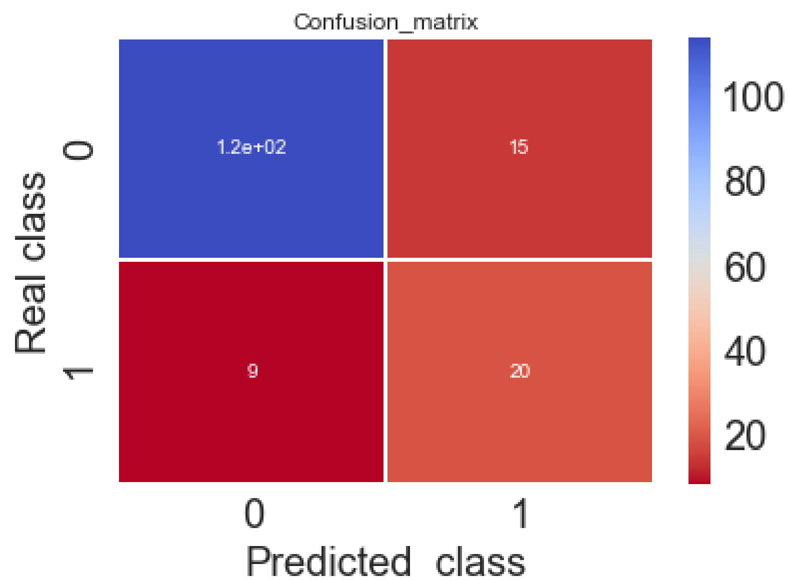
In [42]:
```
from sklearn.model_selection import cross_val_score
#linear_svc = LinearSVC()
linear_svc = LinearSVC()
#linear_svc = LinearSVC(cl)



scores = cross_val_score(linear_svc, trans_X_train, Y_train, cv=2)
print("score mean value: ",scores.mean())
print("std of scores: ",scores.std())
```

```
score mean value:  0.765533275211
std of scores:  0.0128451031677
```

Initial Scores are not so great!

In [43]:
```
linear_svc = LinearSVC()
linear_svc.fit(trans_X_train, Y_train)
Y_pred = linear_svc.predict(trans_X_test)
acc_linear_svc = round(linear_svc.score(trans_X_train, Y_train) * 100,
2)
cnf_matrix=confusion_matrix(Y_test,Y_pred)
print("score for training data",acc_linear_svc)
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_class")
plt.ylabel("Real class")
plt.show()
print("\n----------Classification Report------------------------------------
----")
print(classification_report(Y_test,Y_pred))
acc_linear_svc = round(linear_svc.score(trans_X_test, Y_test) * 100, 2)
print("score for test data",acc_linear_svc)
```

```
score for training data 87.6

/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python
3.4/site-packages/matplotlib/collections.py:590: FutureWarning: element
wise comparison failed; returning scalar instead, but in the future wil
l perform elementwise comparison
  if self._edgecolors == str('face'):
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

          0       0.93      0.88      0.91       130
          1       0.57      0.69      0.62        29

avg / total       0.86      0.85      0.85       159

score for test data 84.91
```

Notice the scores for the training set= 87.There is no overfitting.

Let's perfomed so hyperdimensional tuning to improve the model a bit

In [364]:
```
linear_svc = LinearSVC(class_weight={1: 0.01,0:0.1},C=0.1,tol=1e-6,penal
ty='l2')
#linear_svc = LinearSVC(class_weight={1: 0.01,0:0.06},C=0.1,tol=1e-6)
linear_svc.fit(trans_X_train, Y_train)
Y_pred = linear_svc.predict(trans_X_test)
acc_linear_svc = round(linear_svc.score(trans_X_test, Y_test) * 100, 2)
cnf_matrix=confusion_matrix(Y_test,Y_pred)
print("score for test data",acc_linear_svc)
#sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
#plt.title("Confusion_matrix")
#plt.xlabel("Predicted_class")
#plt.ylabel("Real class")
#plt.show()
print("\n----------Classification Report------------------------------------
----")
print(classification_report(Y_test,Y_pred))
```

```
90.57

----------Classification Report------------------------------------
             precision    recall  f1-score   support

          0       0.90      0.99      0.95       130
          1       0.94      0.52      0.67        29

avg / total       0.91      0.91      0.89       159
```

In [ ]:
```

```

! Great precision = 0.94 and mid-level recall for class 1. The prediction for class 0 is good as expected! Let's do some cross-validation

from sklearn.model_selection import cross_val_score #linear_svc = LinearSVC() linear_svc = LinearSVC(class_weight={1: 0.01,0:0.1},C=0.1,tol=1e-6) #linear_svc = LinearSVC(cl) scores = cross_val_score(linear_svc, trans_X_train, Y_train, cv=3) print("score mean value: ",scores.mean()) print("std of scores: ",scores.std())

In [45]:
```
from sklearn.model_selection import cross_val_score
#linear_svc = LinearSVC()
linear_svc = LinearSVC(class_weight={1: 0.01,0:0.1},C=0.1,tol=1e-6)
#linear_svc = LinearSVC(cl)



scores = cross_val_score(linear_svc, trans_X_train, Y_train, cv=3)
print("score mean value: ",scores.mean())
print("std of scores: ",scores.std())
```

```
score mean value:  0.868075880759
std of scores:  0.030560263189
```

Scores look great and close to those of test data scores! One could expect a robust performance of the SVM estimator.

Summary Remarks

Initially we found relevant features to create Random Forrest that does well in precision ~ 80% for the response class. Cross-validation shows that the model will performed consistently. After some PCA decomposition, further reduction in dimensionality allowed to create a SVM model that can improve the precision to > 90% for class 1 while maintaining a mid-level recall and good precision/recall for class 0 after some hyperdimensional tuning. I suspect that after some systematic hyperdimensional tuning one could improved the recall at the expense of precision for response =1 class, if desired.

In [ ]:

```

```

In [ ]:

```

```