CCPS 884
Data Mining Project
Andy Lee

TABLE OF CONTENTS

CCPS 844 Data Mining (Project) - Andy Lee

1. Select a dataset or datasets of your choice. Here are few links that can be helpful for you to select a dataset.
2. Once you have selected a dataset or datasets of your choice. After reading the datasets, check the type of different attributes/columns/features to ensure that you have appropriate types (categorical/numerical) for your columns.
3. Use visualization to understand your data
4. For exploratory analysis, apply clustering algorithms (K means/ Hierarchical clustering) to improve your understanding
5. Apply the concepts learned in Module 9 to select the features
6. Try to reduce the dimensions of the data if possible (Apply a dimensionality reduction algorithm). For step 7 use both the original data and the data that you get after applying the Step 6.
7. Divide your data in Train and Test or choose cross validation to evaluate the selected model
   - Apply all learned classification algorithms to choose which one performs best
   - Apply all learned regression algorithms to choose which one performs best

Please note that you need to get your data in appropriate format before applying a classification or regression algorithm. One of the differences is: class variable for a regression model is numeric whereas it is categorical for classification.

Clustering

In [189]:
```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[189]:

	STG	SCG	STR	LPR	PEG	UNS	grade
57	0.090	0.600	0.66	0.19	0.59	Middle	2.0
109	0.299	0.295	0.80	0.37	0.84	High	3.0
44	0.115	0.350	0.65	0.27	0.04	very_low	0.0
2	0.060	0.060	0.05	0.25	0.33	Low	1.0
31	0.150	0.295	0.75	0.65	0.24	Low	1.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [190]:
```
y = list(df['UNS'])
#y = df.grade

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]
# keeping all features
#X = df.iloc[:,0:5]
```

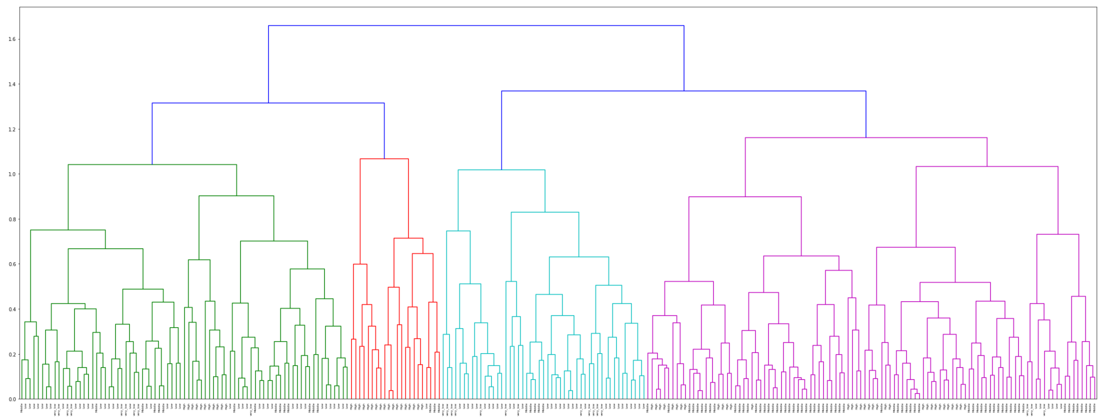
In [191]:
```
df['UNS'].value_counts()
```

Out[191]:
```
Middle      88
Low         83
High        63
very_low    24
Name: UNS, dtype: int64
```

In [192]:
```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
```

Hierarchical clustering (unnormalized)

In [193]:
```
mergings=linkage(X,method='complete')
dendrogram(mergings,labels=y,leaf_rotation=90,leaf_font_size=6)
plt.gcf().set_size_inches(40, 15)
plt.show()
```



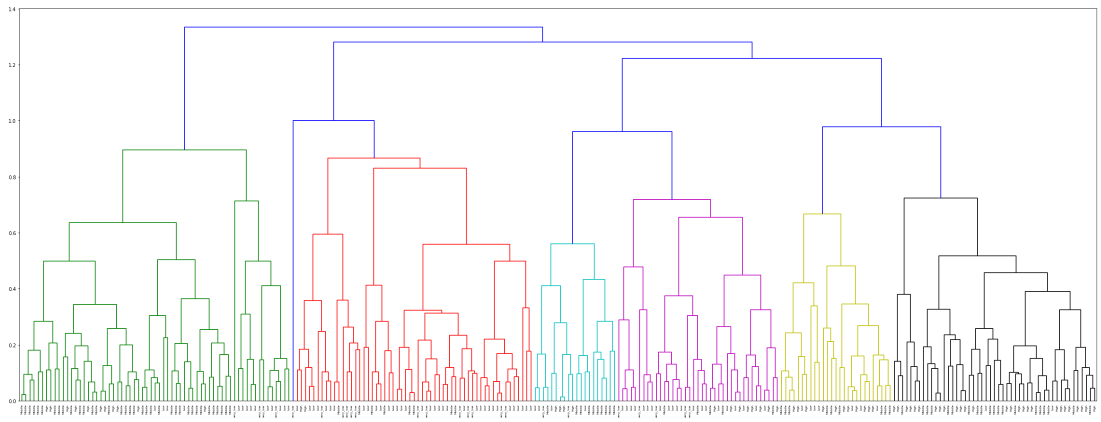
normalizing X

In [194]:
```
plt.clf

from sklearn.preprocessing import normalize
X_norm = normalize(X)
```

In [195]:
```
X_norm
```

Out[195]:
```
array([[ 0.        ,  0.        ,  0.        ,  0.        ],
       [ 0.08508713,  0.10635891,  0.25526138,  0.95723017],
       [ 0.1424138 ,  0.11867817,  0.59339083,  0.78327589],
       ...,
       [ 0.44497603,  0.58506108,  0.23896861,  0.63450286],
       [ 0.43100245,  0.69822397,  0.52582299,  0.22412128],
       [ 0.43352618,  0.49921196,  0.57146632,  0.4860748 ]])
```

In [196]:
```
# Hierarchical clustering (normalized)
mergings=linkage(X_norm,method='complete')
dendrogram(mergings,labels=y,leaf_rotation=90,leaf_font_size=6)
plt.gcf().set_size_inches(40, 15)
plt.show()
```



calculating distance

In [197]:
```
from scipy.cluster.hierarchy import fcluster
labels = fcluster(mergings, 1, criterion='distance')
varieties = list(df['UNS'])
df3 = pd.DataFrame({'labels': labels, 'varieties': varieties})

ct = pd.crosstab(df3.iloc[:,0], df3['varieties'])
df3.count()
```

Out[197]:
```
labels       258
varieties    258
dtype: int64
```

In [198]:
```
df3.sample(5)
```

Out[198]:

	labels	varieties
114	4	Middle
111	1	Low
22	4	Middle
156	1	Middle
208	3	very_low

In [199]:
```
ct
```

Out[199]:

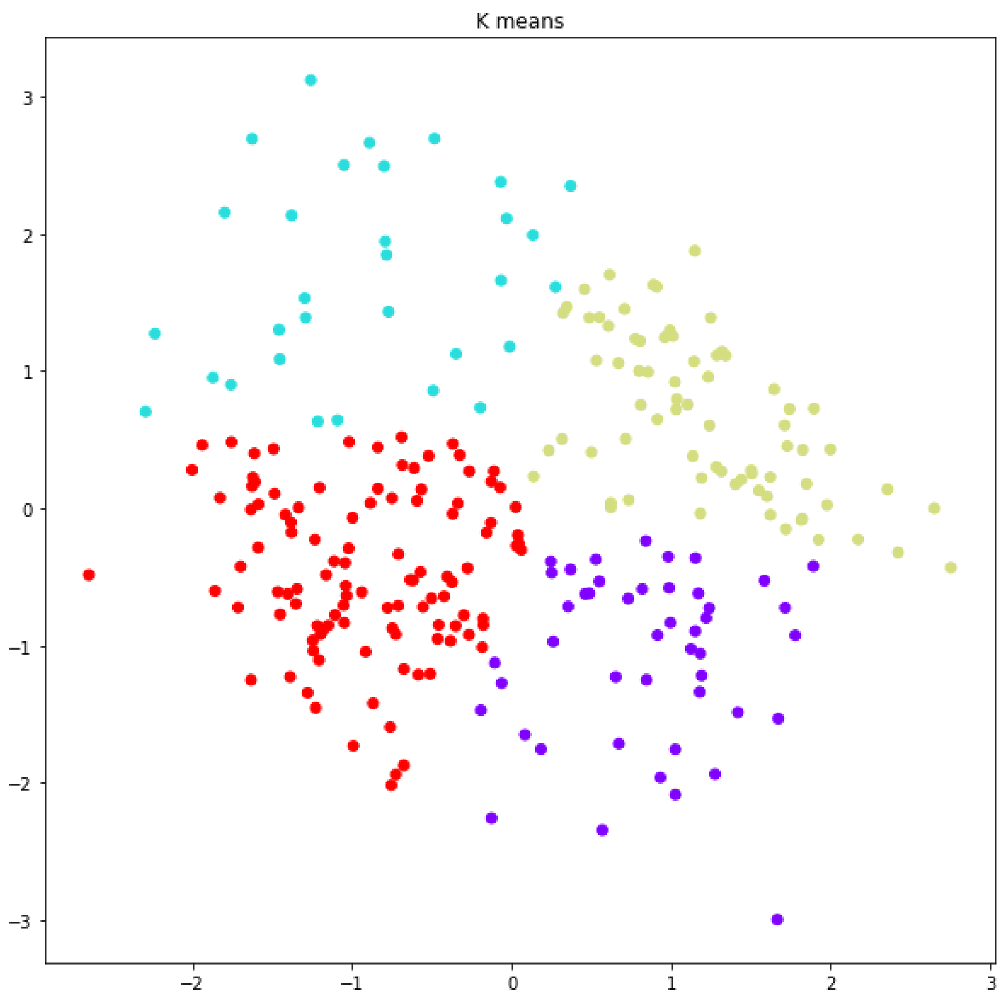
varieties	High	Low	Middle	very_low
labels				
1	11	14	36	4
2	2	35	8	13
3	12	25	15	7
4	38	9	29	0

K means

In [200]:
```
# on goal_results - pairing study time and exam performance for goal objects
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_sc = sc.fit_transform(X)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_sc)

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X_pca)
```

Out[200]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [201]:
```
plt.gcf().set_size_inches(10, 10)
plt.scatter(X_pca[:,0],X_pca[:,1],c=kmeans.labels_,cmap='rainbow')
plt.title('K means')
plt.show()
```



MLP

In [30]:
```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[30]:

	STG	SCG	STR	LPR	PEG	UNS	grade
195	0.550	0.100	0.27	0.25	0.29	Low	1.0
227	0.580	0.348	0.06	0.29	0.31	Low	1.0
95	0.255	0.305	0.86	0.62	0.15	Low	1.0
192	0.370	0.600	0.77	0.40	0.50	Middle	2.0
85	0.248	0.300	0.31	0.20	0.03	very_low	0.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [31]:
```
y = list(df['UNS'])
y_grade = df.grade

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()
```

Out[31]:

	STG	STR	LPR	PEG
211	0.8	0.06	0.31	0.51

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
state=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

MLP

In [33]:
```
from sklearn.neural_network import MLPClassifier

MLP_clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,5), rando
m_state=1)

MLP_clf = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(7,2), random_
state=1)
MLP_clf_pca = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(7,2), ran
dom_state=1)

MLP_clf.fit(X_train,y_train)
#y_pred=clf.predict(X_test)

#print("MLP accuracy :",metrics.accuracy_score(y_test, y_pred))
```

Out[33]:
```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(7, 2), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

In [34]:
```
print("MLP accuracy: ",MLP_clf.score(X_test,y_test))
```

```
MLP accuracy:  0.93023255814
```

PCA + MLP approach

In [35]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

MLP_clf_pca.fit(X_train_pca, y_train)
print("MLP (PCA transformed) accuracy: ", MLP_clf_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy:  0.604651162791
```

In [57]:
```
## KNN multi-label
```

In [58]:
```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[58]:

	STG	SCG	STR	LPR	PEG	UNS	grade
115	0.285	0.640	0.18	0.61	0.45	Middle	2.0
158	0.465	0.258	0.73	0.18	0.59	Middle	2.0
131	0.400	0.180	0.26	0.26	0.67	Middle	2.0
186	0.495	0.820	0.67	0.01	0.93	High	3.0
254	0.780	0.610	0.71	0.19	0.60	Middle	2.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [59]:
```
y = list(df['UNS'])
y_grade = df.grade

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()
```

Out[59]:

	STG	STR	LPR	PEG
173	0.4	0.58	0.75	0.16

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
state=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

KNN

In [61]:
```
# Classifier implementing the k-nearest neighbors vote

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
print("KNN Classfier",knn.score(X_test,y_test))
```

```
KNN Classfier 0.953488372093
```

In [62]:
```
from sklearn.metrics import confusion_matrix
label = ['very_low', 'Low','Middle','High']

print(confusion_matrix(y_test, knn.predict(X_test), label))
```

```
[[ 5  2  0  0]
 [ 0 32  0  0]
 [ 0  1 27  0]
 [ 0  0  1 18]]
```

KNN (PCA transformed)

In [63]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_pca.fit(X_train_pca, y_train)
print("MLP (PCA transformed) accuracy: ", knn_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy:  0.523255813953
```

In [64]:
```
pd.Series(y_test).value_counts().head(1)/len(y_test)
```

Out[64]:
```
Low    0.372093
dtype: float64
```

KNN - one-label vs rest with confusion matrix

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [130]:

```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.loc[df.UNS == 'very_low','vlow'] = 1
df.loc[df.UNS == 'Low','low'] = 1
df.loc[df.UNS == 'Middle','mid'] = 1
df.loc[df.UNS == 'High','high'] = 1

df.fillna(0, inplace=True)
df.sample(5)
```

Out[130]:

	STG	SCG	STR	LPR	PEG	UNS	grade	vlow	low	mid	high
239	0.520	0.44	0.82	0.30	0.52	Middle	2.0	0.0	0.0	1.0	0.0
254	0.780	0.61	0.71	0.19	0.60	Middle	2.0	0.0	0.0	1.0	0.0
87	0.270	0.31	0.32	0.41	0.28	Low	1.0	0.0	1.0	0.0	0.0
190	0.445	0.70	0.82	0.16	0.64	Middle	2.0	0.0	0.0	1.0	0.0
256	0.500	0.75	0.81	0.61	0.26	Middle	2.0	0.0	0.0	1.0	0.0

In [131]:
```
# defining different labels
y1 = df.vlow
y2 = df.low
y3 = df.mid
y4 = df.high

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()
```

Out[131]:

	STG	STR	LPR	PEG
20	0.12	0.2	0.78	0.2

In [132]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y1, test_size=0.33, random_st
ate=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

KNN one label

In [133]:
```
# Classifier implementing the k-nearest neighbors vote

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)

print("KNN Classfier (y1)",knn.score(X_test,y_test))
```

```
KNN Classfier (y1) 0.976744186047
```

In [134]:
```
from sklearn.metrics import confusion_matrix
label = ['very_low', 'Low','Middle','High']

confusion = confusion_matrix(y_test, y_pred)
print(confusion)

TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

accuracy = (TP+TN)/(TP+TN+FP+FN)
sensitivity = TP/(FN+TP)
specificity = TN/(TN+FP)
false_p = FP/(FP+TN)
precision = TP/(FP+TP)

print("\nAccuracy: %f\nSensitivty: %f\nSpecificity: %f\nFalse Positive: %f\nPrecisio
n: %f" % (accuracy,sensitivity,specificity,false_p,precision))
```

```
[[79  0]
 [ 2  5]]

Accuracy: 0.976744
Sensitivty: 0.714286
Specificity: 1.000000
False Positive: 0.000000
Precision: 1.000000
```

KNN (PCA transformed)

In [135]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_pca.fit(X_train_pca, y_train)
y_pred=knn_pca.predict(X_test_pca)

print("MLP (PCA transformed) accuracy (y1): ", knn_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy (y1):  0.918604651163
```

In [136]:
```
from sklearn.metrics import confusion_matrix
label = ['very_low', 'Low','Middle','High']

confusion_matrix(y_test, y_pred)
print(confusion)

TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

accuracy = (TP+TN)/(TP+TN+FP+FN)
sensitivity = TP/(FN+TP)
specificity = TN/(TN+FP)
false_p = FP/(FP+TN)
precision = TP/(FP+TP)

print("\nAccuracy: %f\nSensitivty: %f\nSpecificity: %f\nFalse Positive: %f\nPrecisio
n: %f" % (accuracy,sensitivity,specificity,false_p,precision))
```

```
[[79  0]
 [ 2  5]]

Accuracy: 0.976744
Sensitivty: 0.714286
Specificity: 1.000000
False Positive: 0.000000
Precision: 1.000000
```

against y2. Grade = low

In [137]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y2, test_size=0.33, random_st
ate=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

# KNN
# Classifier implementing the k-nearest neighbors vote

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)

print("KNN Classfier (y2)",knn.score(X_test,y_test))
```

```
KNN Classfier (y2) 0.953488372093
```

KNN (PCA transformed)

In [138]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_pca.fit(X_train_pca, y_train)
y_pred=knn_pca.predict(X_test_pca)

print("MLP (PCA transformed) accuracy (y2): ", knn_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy (y2):  0.790697674419
```

against y3. Grade = Mid

In [139]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y3, test_size=0.33, random_st
ate=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

# KNN
# Classifier implementing the k-nearest neighbors vote

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)

print("KNN Classfier (y3)",knn.score(X_test,y_test))
```

```
KNN Classfier (y3) 0.976744186047
```

KNN (PCA transformed)

In [140]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_pca.fit(X_train_pca, y_train)
y_pred=knn_pca.predict(X_test_pca)

print("MLP (PCA transformed) accuracy (y3): ", knn_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy (y3):  0.639534883721
```

against y4. Grade = High

In [141]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y4, test_size=0.33, random_st
ate=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

# KNN
# Classifier implementing the k-nearest neighbors vote

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)

print("KNN Classfier (y4)",knn.score(X_test,y_test))
```

```
KNN Classfier (y4) 0.988372093023
```

KNN (PCA transformed)

In [142]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_pca.fit(X_train_pca, y_train)
y_pred=knn_pca.predict(X_test_pca)

print("MLP (PCA transformed) accuracy (y4): ", knn_pca.score(X_test_pca, y_test))
```

```
MLP (PCA transformed) accuracy (y4):  0.697674418605
```

KNN probability - classification threshold

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [125]:

```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.loc[df.UNS == 'very_low','vlow'] = 1
df.loc[df.UNS == 'Low','low'] = 1
df.loc[df.UNS == 'Middle','mid'] = 1
df.loc[df.UNS == 'High','high'] = 1

df.fillna(0, inplace=True)
df.sample(5)
```

Out[125]:

	STG	SCG	STR	LPR	PEG	UNS	grade	vlow	low	mid	high
191	0.420	0.700	0.72	0.30	0.80	High	3.0	0.0	0.0	0.0	1.0
165	0.400	0.330	0.12	0.30	0.90	High	3.0	0.0	0.0	0.0	1.0
170	0.420	0.360	0.63	0.04	0.25	Low	1.0	0.0	1.0	0.0	0.0
25	0.090	0.300	0.68	0.18	0.85	High	3.0	0.0	0.0	0.0	1.0
157	0.495	0.276	0.58	0.77	0.83	High	3.0	0.0	0.0	0.0	1.0

In [126]:
```
# defining different labels
y =df.grade
y1 = df.vlow
y2 = df.low
y3 = df.mid
y4 = df.high

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()
```

Out[126]:

	STG	STR	LPR	PEG
12	0.1	0.52	0.78	0.34

In [127]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y4, test_size=0.33, random_st
ate=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

KNN

Classifier implementing the k-nearest neighbors vote

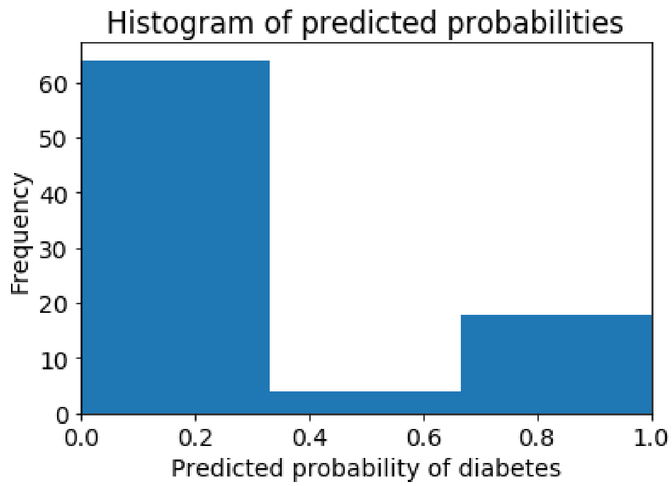
In [128]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn_pca = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)

print("KNN Classfier (y1)",knn.score(X_test,y_test))
```

```
KNN Classfier (y1) 0.988372093023
```

In [129]:
```
y_pred_prob = knn.predict_proba(X_test)[:,1]
```

In [130]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 14
# histogram of predicted probabilities
plt.hist(y_pred_prob, bins=3)
plt.xlim(0, 1)
plt.title('Histogram of predicted probabilities')
plt.xlabel('Predicted probability of diabetes')
plt.ylabel('Frequency')
```

Out[130]: Text(0,0.5,'Frequency')



In [131]:
```
from sklearn.preprocessing import binarize
y_pred_class = binarize([y_pred_prob], 0.3)

from sklearn import metrics
print(metrics.confusion_matrix(y_test, np.reshape(y_pred_class,(-1,1))))
```

```
[[64  3]
 [ 0 19]]
```

null accuracy

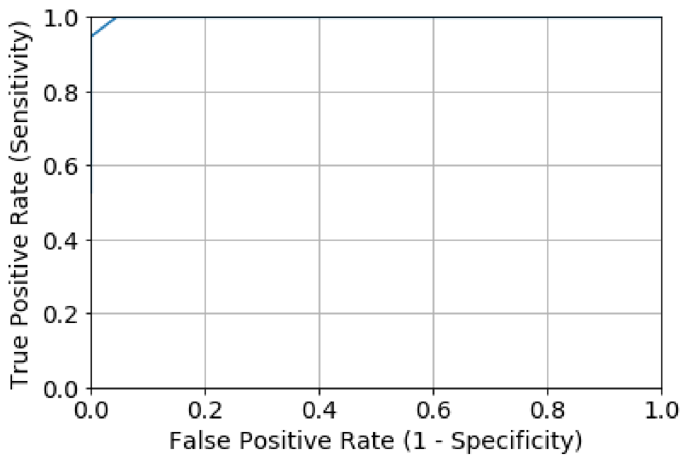
In [132]:
```
max(y_test.mean(), 1-y_test.mean())
```

Out[132]: 0.7790697674418605

In [133]:
```
y_test.value_counts().head(1)/len(y_test)
```

Out[133]:
```
0.0    0.77907
Name: high, dtype: float64
```

In [134]:
```
# IMPORTANT: first argument is true values, second argument is predicted probabiliti
es
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



In [11]:
```
## SVC
```

In [12]:
```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[12]:

	STG	SCG	STR	LPR	PEG	UNS	grade
15	0.12	0.12	0.75	0.35	0.80	High	3.0
25	0.09	0.30	0.68	0.18	0.85	High	3.0
45	0.17	0.36	0.80	0.14	0.66	Middle	2.0
9	0.00	0.00	0.50	0.20	0.85	High	3.0
147	0.33	0.27	0.20	0.33	0.10	very_low	0.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [13]:
```
y = list(df['UNS'])
y_grade = df.grade

# feature selection, dropping SCG
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()
```

Out[13]:

	STG	STR	LPR	PEG
92	0.251	0.57	0.6	0.09

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
state=0)
sc = StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

SVC approach

In [15]:
```
from sklearn.svm import SVC

svc = SVC()
svc_pca = SVC()

svc.fit(X_train, y_train)
print("PCA accuracy: ", svc.score(X_test, y_test))
```

```
PCA accuracy:  0.906976744186
```

PCA + SVC approach

In [16]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

svc_pca.fit(X_train_pca, y_train)
print("PCA + SVC accuracy: ", svc_pca.score(X_test_pca, y_test))
```

```
PCA + SVC accuracy:  0.558139534884
```

In [17]:
```
p=svc.predict(X_test)
```

In [18]:
```
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print("SVM classification accuracy :",metrics.accuracy_score(y_test, p))
```

```
SVM classification accuracy : 0.906976744186
```

Linear Regression

In [54]:

```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[54]:

	STG	SCG	STR	LPR	PEG	UNS	grade
134	0.400	0.12	0.41	0.10	0.65	Middle	2.0
89	0.290	0.30	0.52	0.09	0.67	Middle	2.0
141	0.420	0.15	0.66	0.78	0.40	Middle	2.0
20	0.120	0.28	0.20	0.78	0.20	Low	1.0
78	0.245	0.10	0.71	0.26	0.20	very_low	0.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [55]:
```
#y = list(df['UNS'])
y = df.grade

# feature selection, dropping SCG
#X = df.drop(columns=['SCG']).iloc[:,0:4]
# keeping all features
X = df.iloc[:,0:5]

X.sample()
```

Out[55]:

	STG	SCG	STR	LPR	PEG
197	0.73	0.2	0.07	0.72	0.26

In [56]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
te=0)

# separating transformed
sc = StandardScaler()
X_train_sc=sc.fit_transform(X_train)
X_test_sc=sc.transform(X_test)
```

Linear regression

In [57]:
```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg_pca = LinearRegression()

reg.fit(X_train, y_train)
reg_y_pred=reg.predict(X_test)
print("Linear Regression accuracy: ", reg.score(X_test, y_test))
```

```
Linear Regression accuracy:  0.935041373654
```

In [58]:
```
from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred)))
```

```
MAE:  0.170240265657
MSE:  0.0532685328268
RMSE:  0.230799767822
```

Linear regression + SVC approach

In [59]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_sc)
X_test_pca = pca.transform(X_test_sc)

reg_pca.fit(X_train_pca, y_train)
reg_y_pred_pca = reg_pca.predict(X_test_pca)
print("Linear Regression (with PCA) accuracy: ", reg_pca.score(X_test_pca, y_test))
```

```
Linear Regression (with PCA) accuracy:  0.485295947626
```

In [60]:

```
from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred_pca))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred_pca))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred_pca)))
```

```
MAE:  0.541667047803
MSE:  0.422076808768
RMSE:  0.649674386726
```

Feature select. Dropping SCG to see whether RMSE improves

In [61]:
```
# feature selection, dropping SCG to see whether RMSE improves
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
te=0)

# separating transformed
sc = StandardScaler()
X_train_sc=sc.fit_transform(X_train)
X_test_sc=sc.transform(X_test)

# Linear regression
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg_pca = LinearRegression()

reg.fit(X_train, y_train)
reg_y_pred=reg.predict(X_test)
print("Linear Regression accuracy: ", reg.score(X_test, y_test))

from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred)))

# Linear regression + SVC approach
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_sc)
X_test_pca = pca.transform(X_test_sc)

reg_pca.fit(X_train_pca, y_train)
reg_y_pred_pca = reg_pca.predict(X_test_pca)
print("Linear Regression (with PCA) accuracy: ", reg_pca.score(X_test_pca, y_test))

from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred_pca))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred_pca))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred_pca)))
```

```
Linear Regression accuracy:  0.932686320712
MAE:  0.169140096604
MSE:  0.0551997653979
RMSE:  0.234946303223
Linear Regression (with PCA) accuracy:  0.477349803784
MAE:  0.546350442153
MSE:  0.428592947546
RMSE:  0.65467010589
```

Linear Regression has a slight decrease, while Linear Regression with PCA shows a slight increase.

Logistic Regression

In [19]:

```
% matplotlib inline

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)

df = pd.read_csv('User_Knowledge.csv')

df.loc[df.UNS == 'very_low','grade'] = 0
df.loc[df.UNS == 'Low','grade'] = 1
df.loc[df.UNS == 'Middle','grade'] = 2
df.loc[df.UNS == 'High','grade'] = 3

df.sample(5)
```

Out[19]:

	STG	SCG	STR	LPR	PEG	UNS	grade
191	0.42	0.700	0.72	0.30	0.80	High	3.0
220	0.61	0.258	0.56	0.62	0.24	Low	1.0
136	0.38	0.100	0.40	0.48	0.26	Low	1.0
60	0.09	0.610	0.53	0.75	0.01	Low	1.0
208	0.55	0.170	0.71	0.48	0.11	very_low	0.0

Attribute Information

- STG (The degree of study time for goal object materails), (input value)
- SCG (The degree of repetition number of user for goal object materails) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
  - Very Low: 50
  - Low:129
  - Middle: 122
  - High 130

In [20]:
```
#y = list(df['UNS'])
y = df.grade

# feature selection, dropping SCG
#X = df.drop(columns=['SCG']).iloc[:,0:4]
# keeping all features
X = df.iloc[:,0:5]

X.sample()
```

Out[20]:

	STG	SCG	STR	LPR	PEG
184	0.38	0.59	0.31	0.62	0.2

In [21]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
te=0)

# separating transformed
sc = StandardScaler()
X_train_sc=sc.fit_transform(X_train)
X_test_sc=sc.transform(X_test)
```

Linear regression

In [22]:
```
from sklearn.linear_model import LogisticRegression

reg = LogisticRegression()
reg_pca = LogisticRegression()

reg.fit(X_train, y_train)
reg_y_pred=reg.predict(X_test)
print("Logistic Regression accuracy: ", reg.score(X_test, y_test))
```

```
Logistic Regression accuracy:  0.720930232558
```

In [23]:
```
from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred)))
```

```
MAE:  0.279069767442
MSE:  0.279069767442
RMSE:  0.528270543795
```

Linear regression + SVC approach

In [24]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_sc)
X_test_pca = pca.transform(X_test_sc)

reg_pca.fit(X_train_pca, y_train)
reg_y_pred_pca = reg_pca.predict(X_test_pca)
print("Logistic Regression (with PCA) accuracy: ", reg_pca.score(X_test_pca, y_test
))
```

```
Logistic Regression (with PCA) accuracy:  0.546511627907
```

MAE, MSE, RMSE

In [25]:
```
from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred_pca))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred_pca))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred_pca)))
```

```
MAE:  0.476744186047
MSE:  0.523255813953
RMSE:  0.723364233256
```

Feature selection. Dropping SCG to see whether RMSE improves

In [26]:

```
# feature selection, dropping SCG to see whether RMSE improves
X = df.drop(columns=['SCG']).iloc[:,0:4]

X.sample()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
te=0)

# separating transformed
sc = StandardScaler()
X_train_sc=sc.fit_transform(X_train)
X_test_sc=sc.transform(X_test)

# Linear regression
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg_pca = LinearRegression()

reg.fit(X_train, y_train)
reg_y_pred=reg.predict(X_test)
print("Logistic Regression accuracy: ", reg.score(X_test, y_test))

from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred)))

# Linear regression + SVC approach
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_sc)
X_test_pca = pca.transform(X_test_sc)

reg_pca.fit(X_train_pca, y_train)
reg_y_pred_pca = reg_pca.predict(X_test_pca)
print("Logistic Regression (with PCA) accuracy: ", reg_pca.score(X_test_pca, y_test
))

from sklearn import metrics
print("MAE: ", metrics.mean_absolute_error(y_test, reg_y_pred_pca))
print("MSE: ", metrics.mean_squared_error(y_test, reg_y_pred_pca))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, reg_y_pred_pca)))
```

```
Logistic Regression accuracy:  0.932686320712
MAE:  0.169140096604
MSE:  0.0551997653979
RMSE:  0.234946303223
Logistic Regression (with PCA) accuracy:  0.477349803784
MAE:  0.546350442153
MSE:  0.428592947546
RMSE:  0.65467010589
```

Logistic Regression with 4 columns increases significantly. However Logistic Regression with PCA decreases.