Predicting Seizures and Epilepsy Milestones 4 & 5

Harvard CS 109A

November 28, 2016

Sean Keery & Shivas Jayaram

Exploring various factors that might affect epilepsy and seizure recurrence

Well since milestone 3, we've obtained more data. The EEG and drug study data we used for that milestone was very limited in it's predictive ability for general populations. The additional data was from the UK data service. It consisted of the 1958 birth sweep and three more sweeps over the next twenty-one years. It included almost 1800 variables over 19000 unique cases. We had gone to the other end of the spectrum.

However, after inspecting the data, we determined that values for many of the variables were not present. Due to time constraints, we used means and medians to fill in the absent values. Input from two ancillary surveys were merged with the original file to enhance demographic components.

After our initial review, we identified some variables which allowed us to categorize subjects with epilepsy or a history of seizures. Fitting our models based on these categorizations helped us to identify variables with predictive power. Unfortunately, we had missed some variables which should have been used in our first classification. So we excluded these from our model too. After re-fitting, there are still some questions in our minds whether we are able to effectively identify factors in recurrence. We believe that some of the factors identified through Principal Component Analysis and sklearn feature selection aren't necessarily causes of recurrence. We will continue to explore this moving forward.

Milestone 4 - Baseline Models

Data Set 1 Thall & Vail

This dataset gives a two-week seizure counts for 59 epileptics. The number of seizures was recorded for a baseline period of 8 weeks, and then patients were randomly assigned to a treatment group or a control group. Counts were then recorded for four successive two-week periods. The subject's age is the only covariate.

Data Set 2 NCDS Sweeps 0 to 3

The National Child Development Study (NCDS) originated in the Perinatal Mortality Survey, which examined social and obstetric factors associated with still birth and infant mortality among over 17,000 babies born in Britain in one week in March 1958. The study has broadened in scope to chart many aspects of the health, educational, and social development of cohort members as they passed through childhood and adolescence. We used results from the first three sweeps (1965 at age 7, 1969 at age 11, 1974 at age 16) in addition to the birth data.

Surviving members of this birth cohort have been surveyed on five more occasions in order to monitor their changing health, education, social and economic circumstances. We have excluded these datasets at this time in order to establish our baselines models.

Models

We have decided that the performance metric to evaluate prediction will be seizure recurrence.

Features extracted using PCA and SelectKBest include:

- Features extracted from the entire feature set: n1827, n604, n39, n1263, n1400, n1399, n1453, n1476, n825, n2598, n1896, n1898, dvht07, OUTCME01, OUTCME02

- Features extracted from female patients: n1region, n514, n383, n1819, n1827, n1828, n1400, n1399, n825, n1896, n1898, dvht07, dvrwt07, dvwt07, OUTCME01
- Features extracted from male patients: n400, n403, n1827, n604, n39, n92, n1400, n1399, n1476, n1548, n1551, n825, n1896, OUTCME01, OUTCME02

Linear regression provided us with about sixty percent accuracy using the features we had selected above.

LDA, QDA, Trees and others also provided the same accuracy.

Step 1: Explore and Clean Data

First task will be to read, explore, and clean the data. Our first version of the data exploration can be found here (ExploreData.ipynb). There was a lot of exploratory and debugging code, so we branched off ot this notebook with the findings we thought would be appropriate for this milestone of the project.

In [1]:
```
import numpy as np
import pandas as pd
import scipy as sp
from sklearn import preprocessing
from sklearn.cross_validation import KFold
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
as QDA
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.tree import DecisionTreeClassifier as DecisionTree
from sklearn.ensemble import RandomForestClassifier as RandomForest
from sklearn.ensemble import AdaBoostClassifier as AdaBoost
from sklearn.svm import SVC
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn import grid_search
from sklearn.decomposition import PCA
from sklearn import feature_selection as fs
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

In [2]:
```
#Load and inspect the ncds data
ncds_data = pd.read_csv('datasets/ncds0123.txt', delimiter='\t', low_mem
ory=False)
# Print shapes
print "Shape of data:", ncds_data.shape
ncds_data.head()
```

```
Shape of data: (18558, 1765)
```

Out[2]:

	ncdsid	n622	n0region	n1region	n2region	n3region	n553	n545	n520	n490	...	n1
0	N10001N	2	9	9	9	9	23	4	2	12	...	-1
1	N10002P	1	9	8	8	8	34	4	5	1	...	-1
2	N10003Q	1	4	4	4	4	34	4	10	1	...	-1
3	N10004R	2	1	1	1	1	26	4	11	1	...	-1
4	N10005S	2	10	10	10	10	25	4	1	3	...	-1

5 rows × 1765 columns

In [3]:
```
#Load and inspect the pms additions data
ncds_pms_data = pd.read_csv('datasets/ncds_pms_additionals.txt', delimit
er='\t', low_memory=False)
# Print shapes
print "Shape of data:", ncds_pms_data.shape
ncds_pms_data.head()
```

```
Shape of data: (16990, 54)
```

Out[3]:

	NCDSID	N622	BSTATUS	POD	BOOKING	PLANC	DIASTOL	MAXDBP	ALBECL	XR
0	N10001N	2	0	8	8	2	1	1	0	0
1	N10002P	1	0	2	0	4	4	3	0	0
2	N10003Q	1	0	8	8	2	1	3	0	0
3	N10004R	2	0	8	8	2	1	-8	-8	1
4	N10005S	2	0	8	8	2	1	3	0	1

5 rows × 54 columns

In [4]:
```
#Load and inspect the response data
ncds_response_data = pd.read_csv('datasets/ncds_response.txt', delimiter=
\t', low_memory=False)
# Print shapes
print "Shape of data:", ncds_response_data.shape
ncds_response_data.head()
```

```
Shape of data: (18558, 18)
```

Out[4]:

	NCDSID	N622	BSTATUS	COBIRTH	MULTIPNO	MULTCODE	ETHNICID	OUTCME00
0	N10001N	2	0	1	-1	-1	1	1
1	N10002P	1	0	1	-1	-1	1	1
2	N10003Q	1	0	1	-1	-1	1	1
3	N10004R	2	0	1	-1	-1	1	1
4	N10005S	2	0	2	-1	-1	5	1

Step 1.1: Understand the data:

1. Explore the data
2. Understand the predictors, what they mean in real life
3. Understand the values of each predictors
4. Join appropriate datasets

In [5]:
```
# Columns the help us identify if the patient has epilepsy
epil_columns = ["n390","n391","n392","n415", "n1842", "n1307", "n1308",
"n1309", "n1314", "n1317", "n1477", "n1478", "n1479", "n2416", "n2663",
"n2664", "n2665"
                , "n2666", "n2667", "n1893", "n1894", "n1895", "n1904",
"n1910", 'n1817', 'n1818',  'n1394', 'n1502', 'n2615', 'n2616']

def evaluate_data(df):
    # Check for range of unique values for the train data
    for i in range(df.shape[1]):
        vals = np.unique(df.iloc[:, i])
        if len(vals) < 15:
            print '(Categorical) {} unique values - {}: {}'.format(len(v
als), df.columns[i], vals)
        else:
            print '(Continuous) range of values - ', df.columns[i], ':
 {} to {}'.format(df.iloc[:, i].min(), df.iloc[:, i].max())

def evaluate_epil_columns(df):
    for column in epil_columns:
        vals = np.unique(df[column])
        if len(vals) < 15:
            print '(Categorical) {} unique values - {}: {}'.format(len(v
als), column, vals)
        else:
            print '(Continuous) range of values - ', column, ': {} to
 {}'.format(df[column].min(), df[column].max())


def columns_with_null(df):
    for column in df.columns:
        df_missing = df[df[column].isnull()]
        count = 0
        if df_missing.shape[0] > 0:
            print "Predictor " , column, " contain null values / Count =
 " ,df_missing.shape[0]
            count = count +1
    print "Total number of columns with null:",count
```

In [6]:
```
# Join datasets
ncds_merged_data = pd.merge(left=ncds_data,right=ncds_pms_data,how='lef
t',left_on='ncdsid',right_on='NCDSID')
ncds_merged_data = pd.merge(left=ncds_merged_data,right=ncds_response_da
ta,how='left',left_on='ncdsid',right_on='NCDSID')
print "Shape of data:", ncds_merged_data.shape
```

```
Shape of data: (18558, 1837)
```

In [7]:
```
# Evalute the ncds data
#evaluate_data(ncds_merged_data)
```

Step 1.2: Handle missing data:

Are there any missing values, if there are:

1. Can we impute them based on some algorithm
2. Remove or ignore them
3. Assume values based on common sense or prior knowledge

In [8]:

```
# Remove spaces from data
def convert_spaces_to_null(data):
    data = data.replace([' '],[None])
    return data

def fill_with_median(x_fill):
    x_fill = x_fill.groupby(x_fill.columns, axis = 1).transform(lambda
x: x.fillna(x.median()))
    return x_fill

def fill_with_mean(x_fill):
    x_fill = x_fill.groupby(x_fill.columns, axis = 1).transform(lambda
x: x.fillna(x.mean()))
    return x_fill

def fill_pms_columns(x_fill):
    for index, row in x_fill.iterrows():
#         # 0-3D Sex of child
#         if pd.isnull(row["N622"]):
#             x_fill.set_value(index, 'N622', -1.0)
        # Reconciled Birth Status
#         if pd.isnull(row["BSTATUS"]):
#             x_fill.set_value(index, 'BSTATUS', 0)
        # Q6:Place of Delivery
        if pd.isnull(row["POD"]):
            x_fill.set_value(index, 'POD', 5.0)
        # Q26b: Booking In place
        if pd.isnull(row["BOOKING"]):
            x_fill.set_value(index, 'BOOKING', 3.0)
        # Q21b: Place of Antenatal care
        if pd.isnull(row["PLANC"]):
            x_fill.set_value(index, 'PLANC', -8.0)
        # Q29a: Diastolic Blood Pressure
        if pd.isnull(row["DIASTOL"]):
            x_fill.set_value(index, 'DIASTOL', -2.0)
        # Q29b: Maximum Diatolic Blood Pressure
        if pd.isnull(row["MAXDBP"]):
            x_fill.set_value(index, 'MAXDBP', -2.0)
        # Q31: Albuminuria and Eclampsia
        if pd.isnull(row["ALBECL"]):
            x_fill.set_value(index, 'ALBECL', -8.0)
        # Q36: X-Ray given
        if pd.isnull(row["XRAY"]):
            x_fill.set_value(index, 'XRAY', -8.0)
        # Q37: Obstetric, pregnancy abnormality - No information
        if pd.isnull(row["ABNORM0X"]):
            x_fill.set_value(index, 'ABNORM0X', -2.0)
        # Q37: No Obstetric, pregnancy abnormality
        if pd.isnull(row["ABNORM00"]):
            x_fill.set_value(index, 'ABNORM00', -2.0)
        # Q37: Obstetric, pregnancy abnormality - Diabetes
        if pd.isnull(row["ABNORM01"]):
            x_fill.set_value(index, 'ABNORM01', -2.0)
        # Q37: Obstetric, pregnancy abnormality - Heart
        if pd.isnull(row["ABNORM02"]):
            x_fill.set_value(index, 'ABNORM02', -2.0)
        # Q37: Obstetric, pregnancy abnormality - Active TB
```

```
if pd.isnull(row["ABNORM03"]):
    x_fill.set_value(index, 'ABNORM03', -2.0)
# Q37: Obstetric, pregnancy abnormality - influenza
if pd.isnull(row["ABNORM04"]):
    x_fill.set_value(index, 'ABNORM04', -2.0)
# Q37: Obstetric, pregnancy abnormality - German Measles
if pd.isnull(row["ABNORM05"]):
    x_fill.set_value(index, 'ABNORM05', -2.0)
# Q37: Obstetric, pregnancy abnormality - Disproportion
if pd.isnull(row["ABNORM06"]):
    x_fill.set_value(index, 'ABNORM06', -2.0)
# Q37: Obstetric, pregnancy abnormality - External version
if pd.isnull(row["ABNORM07"]):
    x_fill.set_value(index, 'ABNORM07', -2.0)
# Q37: Obstetric, pregnancy abnormality - Epilepsy
if pd.isnull(row["ABNORM08"]):
    x_fill.set_value(index, 'ABNORM08', -2.0)
# Q37: Obstetric, pregnancy abnormality - Other
if pd.isnull(row["ABNORM09"]):
    x_fill.set_value(index, 'ABNORM09', -2.0)
# Q37: Bleeding in Pregnancy and before delivery
if pd.isnull(row["BLEED"]):
    x_fill.set_value(index, 'BLEED', -1.0)
# Q38a: Admission to hospital
if pd.isnull(row["AD2HOSP"]):
    x_fill.set_value(index, 'AD2HOSP', -1.0)
# Q39: Type of Labour or Delivery Admission (Hospital)
if pd.isnull(row["ADTYPE"]):
    x_fill.set_value(index, 'ADTYPE', -1.0)
# Q44: Presenting Part
if pd.isnull(row["PRESENT"]):
    x_fill.set_value(index, 'PRESENT', -1.0)
# Q49a: No drugs of this type
if pd.isnull(row["LDRUG00"]):
    x_fill.set_value(index, 'LDRUG00', -2.0)
# Q49a: Chloral, Welldorm
if pd.isnull(row["LDRUG01"]):
    x_fill.set_value(index, 'LDRUG01', -2.0)
# Q49a: Barbiturate
if pd.isnull(row["LDRUG02"]):
    x_fill.set_value(index, 'LDRUG02', -2.0)
# Q49a: Heroin
if pd.isnull(row["LDRUG03"]):
    x_fill.set_value(index, 'LDRUG03', -2.0)
# Q49a: Largactil (chlorpomazine)
if pd.isnull(row["LDRUG04"]):
    x_fill.set_value(index, 'LDRUG04', -2.0)
# Q49a: Sparine (promazine)
if pd.isnull(row["LDRUG05"]):
    x_fill.set_value(index, 'LDRUG05', -2.0)
# Q49a: Phenergan (promethazine)
if pd.isnull(row["LDRUG06"]):
    x_fill.set_value(index, 'LDRUG06', -2.0)
# Q49a: Doriden
if pd.isnull(row["LDRUG07"]):
    x_fill.set_value(index, 'LDRUG07', -2.0)
# Q49a: Oblivon
```

```
    if pd.isnull(row["LDRUG08"]):
        x_fill.set_value(index, 'LDRUG08', -2.0)
    # Q49a: Other
    if pd.isnull(row["LDRUG09"]):
        x_fill.set_value(index, 'LDRUG09', -2.0)
    # Q50: Anaesthetic
    if pd.isnull(row["ATHETIC"]):
        x_fill.set_value(index, 'ATHETIC', -2.0)
    # Q55: Resuscitation
    if pd.isnull(row["RESUS"]):
        x_fill.set_value(index, 'RESUS', -2.0)
    # Q56: Drugs to baby (None)
    if pd.isnull(row["DTB1"]):
        x_fill.set_value(index, 'DTB1', -2.0)
    # Q56: Drugs to baby (Coranine)
    if pd.isnull(row["DTB2"]):
        x_fill.set_value(index, 'DTB2', -2.0)
    # Q56: Drugs to baby (Lobeline)
    if pd.isnull(row["DTB3"]):
        x_fill.set_value(index, 'DTB3', -2.0)
    # Q56: Drugs to baby (Sedatives)
    if pd.isnull(row["DTB4"]):
        x_fill.set_value(index, 'DTB4', -2.0)
    # Q56: Drugs to baby (Antagonists, nalorphine, levalorfan)

    if pd.isnull(row["DTB5"]):
        x_fill.set_value(index, 'DTB5', -2.0)
    # Q56: Drugs to baby (Synkavit, Vikastab)
    if pd.isnull(row["DTB6"]):
        x_fill.set_value(index, 'DTB6', -2.0)
    # Q56: Drugs to baby (Sulphonamides)
    if pd.isnull(row["DTB7"]):
        x_fill.set_value(index, 'DTB7', -2.0)
    # Q56: Drugs to baby (Penicilin)
    if pd.isnull(row["DTB8"]):
        x_fill.set_value(index, 'DTB8', -2.0)
    # Q56: Drugs to baby (Streptomycin)
    if pd.isnull(row["DTB9"]):
        x_fill.set_value(index, 'DTB9', -2.0)
    # Q56: Drugs to baby (Other antibiotics)
    if pd.isnull(row["DTB10"]):
        x_fill.set_value(index, 'DTB10', -2.0)
    # Q59: Baby's Illness
    if pd.isnull(row["ILLNESS"]):
        x_fill.set_value(index, 'ILLNESS', -1.0)
    # Q61: Month of Death
    if pd.isnull(row["MOD"]):
        x_fill.set_value(index, 'MOD', 0.0)
    # Q61: Time of death
    if pd.isnull(row["TOD"]):
        x_fill.set_value(index, 'TOD', -1.0)
    # Q61: Age at Death
    if pd.isnull(row["AAD"]):
        x_fill.set_value(index, 'AAD', -1.0)
    # Q61: Still Birth or Neo-natal Death (Dervied)
    if pd.isnull(row["SBNND"]):
        x_fill.set_value(index, 'SBNND', -1.0)
```

```
        # Placental Weight
        if pd.isnull(row["PLCWGT"]):
            x_fill.set_value(index, 'PLCWGT', -2.0)
        # Time of death for still births and neonatal deaths (Table 62)

        if pd.isnull(row["TABLE62"]):
            x_fill.set_value(index, 'TABLE62', -1.0)
    return x_fill
```

In [9]:
```
# Make a copy
ncds_data_clean = ncds_merged_data.copy()

# drop the ID columns
ncds_data_clean =  ncds_data_clean.drop(["ncdsid","NCDSID_x","NCDSID_y"],
s=1)
# Drop other duplicate columns
ncds_data_clean =  ncds_data_clean.drop(["N622_x","BSTATUS_x"],axis=1)


# Convert spaces in the data to nulls
ncds_data_clean = convert_spaces_to_null(ncds_data_clean)

# Convert all columns to float
for column in ncds_data_clean.columns:
    ncds_data_clean[column] = ncds_data_clean[column].astype(float)

# Impute missing data from joined columns using default values
ncds_data_clean = fill_pms_columns(ncds_data_clean)

# Impute missing data with median values
ncds_data_clean = fill_with_median(ncds_data_clean)

# Impute missing data with mean values - there are some columns we canno
t impute with median
ncds_data_clean = fill_with_mean(ncds_data_clean)
```

In [10]:
```
# Get the columns which have null data
columns_with_null(ncds_data_clean)
```

```
Total number of columns with null: 0
```

Step 1.3: Identify Epilepsy Records:

In our data a patient is assumed to be epileptic is one or more conditions are satisified in the dataset. We need to check all the conditions in the data and determine if the patient is epileptic.

In [11]:

```
# Identify if patient has epilepsy
ncds_data_clean["epileptic"] = 0
for index, row in ncds_data_clean.iterrows():
    # 1M Reason for Special Education MC1:3
    if row["n390"] == 10.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 1M Reason for Special Education MC2:3
    if row["n391"] == 10.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 1M Reason for Special Education MC2:3
    if row["n392"] == 10.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 1M Epileptic condition
    if row["n415"] >= 3.0 :
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 12D Epilepsy identification
    if row["n1842"] == 5.0 :
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Has child had epilepsy attacks-MC 1:3
    if (row["n1307"] >= 1.0 and row["n1307"] <= 5.0):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Has child had epilepsy attacks-MC 2:3
    if (row["n1308"] >= 1.0 and row["n1308"] <= 5.0):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Has child had epilepsy attacks-MC 3:3
    if (row["n1309"] >= 1.0 and row["n1309"] <= 5.0):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Age at most recent epilepsy attack
    if (row["n1314"] >= 0.0):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Age at 1st epilepsy attack
    if (row["n1317"] >= 0.0):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2M Reason for special education - MC1:3
    if row["n1477"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2M Reason for special education - MC2:3
    if row["n1478"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2M Reason for special education - MC3:3
    if row["n1479"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Type hcap for which will require help
    if row["n2416"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Nature of child-s disability-MC 1:5
    if row["n2663"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Nature of child-s disability-MC 2:5
    if row["n2664"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Nature of child-s disability-MC 3:5
    if row["n2665"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Nature of child-s disability-MC 4:5
    if row["n2666"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
```

```
    # 3P Nature of child-s disability-MC 5:5
    if row["n2667"] == 7.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Category of child's handicap MC1:3
    if row["n1893"] == 8.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Category of child's handicap MC2:3
    if row["n1894"] == 8.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Category of child's handicap MC3:3
    if row["n1895"] == 8.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Reason for hosp admiss last 12 mnths
    if row["n1904"] == 17.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Reason hosp outpatient last yr
    if row["n1910"] == 17.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3M Epilepsy
#     if row["n2032"] >= 1.0:
#         ncds_data_clean.set_value(index, 'epileptic', 1)

    # New columns

    # 1D Defects found in NCDS1 sample-MC 1:4
    if row["n1817"] > 0.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 1D Defects found in NCDS1 sample-MC 2:4
    if row["n1818"] > 0.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 1D Defects found in NCDS1 sample-MC 3:4
    if row["n1819"] > 0.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2P Ever seen specialist-convulsions,fits
    if row["n1394"] == 5.0:
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 2M Has child ever had convulsions
    if any(row["n1502"] == s for s in [2.0,3.0,4.0]):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P When convulsions,fits 1st occured
    if any(row["n2615"] == s for s in [1.0,2.0,3.0,4.0,5.0,6.0]):
        ncds_data_clean.set_value(index, 'epileptic', 1)
    # 3P Convulsions-most recent occurrence
    if any(row["n2616"] == s for s in [1.0,2.0,3.0,4.0,5.0,6.0,7.0]):
        ncds_data_clean.set_value(index, 'epileptic', 1)
```

In [12]:
```
print "Number of rows with epilepsy",ncds_data_clean[ncds_data_clean["epileptic"] == 1].shape[0]
```

```
Number of rows with epilepsy 1609
```

In [13]:
```
# Remove the epilepsy columns from the data
ncds_data_no_indicators=ncds_data_clean.copy()
ncds_data_no_indicators.drop(epil_columns,inplace=True,axis=1)
print "Shape of dataset: " , ncds_data_no_indicators.shape
```

```
Shape of dataset:  (18558, 1803)
```

Step 1.4: Split data into train and test:

Split our dataset into train and test and analyze the splits. We can explore and verify the matrix of classes to check if our data is balanced. If the class is Imbalanced we will need to do any of the following:

1. Over sample
2. Under sample
3. Over weight
4. Adjust class weights in model

In [14]:
```
x = ncds_data_no_indicators.values[:, :-1]
y = ncds_data_no_indicators.values[:, -1]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
 random_state=42)

#Print some useful info for our test, train sets
print 'Train data: ', x_train.shape
print 'Test data: ', x_test.shape
print 'Train class 0: {}, train class 1: {}'.format(len(y_train[y_train == 0]), len(y_train[y_train == 1]))
print 'Test class 0: {}, test class 1: {}'.format(len(y_test[y_test == 0]), len(y_test[y_test == 1]))
```

```
Train data:  (11134, 1802)
Test data:  (7424, 1802)
Train class 0: 10159, train class 1: 975
Test class 0: 6790, test class 1: 634
```

Step 2: Feature Selection

From the merged datasets we can see we have over 1800 features. Going through the 1800 would be a very time consuming task so let us apply some algorithims to find the best features that we can use to build the model. In our exploration phase we did use PCA to find a subset of components but chose not to use those components in our base models. The exploration phase can be seen here (ExploreData.ipynb). However we may chose to use PCA during model tuning and evaluating model performance phase.

In [15]:
```
# Best features
num_of_features = 15
features = fs.SelectKBest(fs.f_regression, k=num_of_features) #k is numb
er of features.
features.fit(x_train, y_train)

selected_features = features.get_support()
print "Selected Features:"
selected_features_columns =  ncds_data_no_indicators.columns[selected_fe
atures].values
print selected_features_columns
```

```
Selected Features:
['n1827' 'n604' 'n39' 'n1263' 'n1400' 'n1399' 'n1453' 'n1476' 'n825'
 'n2598' 'n1896' 'n1898' 'dvht07' 'OUTCME01' 'OUTCME02']
```

```
//anaconda/envs/py27/lib/python2.7/site-packages/pandas/indexes/base.p
y:1275: VisibleDeprecationWarning: boolean index did not match indexed
 array along dimension 0; dimension is 1803 but corresponding boolean d
imension is 1802
  result = getitem(key)
```

Step 3: Build Base Models

In [16]:
```
# Function for computing the accuracy a given model on the entire test s
et,
# the accuracy on class 0 in the test set
# and the accuracy on class 1
score = lambda model, x_test, y_test: pd.Series([model.score(x_test, y_t
est),
                                                 model.score(x_test[y_te
st==0], y_test[y_test==0]),
                                                 model.score(x_test[y_te
st==1], y_test[y_test==1])],
                                                index=['overall accurac
y', 'accuracy on class 0', 'accuracy on class 1'])
```

In [17]:
```
# Split data for selected features only
#x = ncds_data_no_indicators.values[:, selected_features]
x = ncds_data_no_indicators[selected_features_columns].values[:,:]
y = ncds_data_no_indicators.values[:, -1]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
 random_state=42)

#Print some useful info for our test, train sets
print 'Train data: ', x_train.shape
print 'Test data: ', x_test.shape
print 'Train class 0: {}, train class 1: {}'.format(len(y_train[y_train
== 0]), len(y_train[y_train == 1]))
print 'Test class 0: {}, test class 1: {}'.format(len(y_test[y_test ==
0]), len(y_test[y_test == 1]))
```

```
Train data:  (11134, 15)
Test data:  (7424, 15)
Train class 0: 10159, train class 1: 975
Test class 0: 6790, test class 1: 634
```

Step 3.1: Linear Regression:

In [18]:
```
# Linear regression
linear = LinearRegression()
linear.fit(x_train, y_train)
linear_scores = score(linear, x_test, y_test)
print "Linear regression:"
print linear_scores
```

```
Linear regression:
overall accuracy       0.050431
accuracy on class 0    0.000000
accuracy on class 1    0.000000
dtype: float64
```

Step 3.2: Logistic Regression:

In [19]:
```
# Unweighted logistic regression
unweighted_logistic = LogisticRegression()
unweighted_logistic.fit(x_train, y_train)
unweighted_log_scores = score(unweighted_logistic, x_test, y_test)

# Weighted logistic regression
weighted_logistic = LogisticRegression(class_weight='balanced')
weighted_logistic.fit(x_train, y_train)
weighted_log_scores = score(weighted_logistic, x_test, y_test)

print "Logistic regression (Unweighted):"
print unweighted_log_scores
print "Logistic regression (Weighted):"
print weighted_log_scores
```

```
Logistic regression (Unweighted):
overall accuracy       0.915948
accuracy on class 0    0.997791
accuracy on class 1    0.039432
dtype: float64
Logistic regression (Weighted):
overall accuracy       0.623788
accuracy on class 0    0.625626
accuracy on class 1    0.604101
dtype: float64
```

Step 3.3: Linear Discriminant Analysis:

In [20]:
```
# LDA
lda = LDA()
lda.fit(x_train, y_train)
lda_scores = score(lda, x_test, y_test)

print "LDA:"
print lda_scores
```

```
LDA:
overall accuracy       0.912177
accuracy on class 0    0.988954
accuracy on class 1    0.089905
dtype: float64
```

Step 3.4: Other Models:

Let us build some more baseline models using other techniques and compare to the ones above

In [21]:
```
# KNN
knn = KNN()
knn.fit(x_train, y_train)
knn_scores = score(knn, x_test, y_test)

#QDA
qda = QDA()
qda.fit(x_train, y_train)
qda_scores = score(qda, x_test, y_test)

#Decision Tree
tree = DecisionTree()
tree.fit(x_train, y_train)
tree_scores = score(tree, x_test, y_test)

#Random Forest
rf = RandomForest(class_weight='balanced')
rf.fit(x_train, y_train)
rf_scores = score(rf, x_test, y_test)

# SVC
svc = SVC(probability=True,class_weight='balanced')
svc.fit(x_train, y_train)
svc_scores = score(svc, x_test, y_test)

#Score Dataframe
score_df = pd.DataFrame({'linear':linear_scores,
                         'unweighted logistic': unweighted_log_scores,
                         'weighted logistic': weighted_log_scores,
                         'lda': lda_scores,
                         'qda': qda_scores,
                        'knn': knn_scores,
                         'tree': tree_scores,
                         'rf': rf_scores,'svc':svc_scores})
score_df
```

Out[21]:

	knn	lda	linear	qda	rf	svc	tree	unweigh logistic
overall accuracy	0.909752	0.912177	0.050431	0.851832	0.799300	0.715383	0.895609	0.91594
accuracy on class 0	0.988218	0.988954	0.000000	0.911487	0.856701	0.737113	0.964654	0.99779
accuracy on class 1	0.069401	0.089905	0.000000	0.212934	0.184543	0.482650	0.156151	0.03943

Step 4: Feature Selection by Demographic:

Let's group the data by demographics and perform a feature selection on each individual group

1. Sex
2. Country of birth
3. Ethnic group

In [22]:
```
def find_features_by_group(column,list_of_group):
    for i in list_of_group:
        ncds_data_no_indicators_by_group = ncds_data_no_indicators[ncds_
data_no_indicators[column] == i]
        if ncds_data_no_indicators_by_group.shape[0] > 50:
            x = ncds_data_no_indicators_by_group.values[:, :-1]
            y = ncds_data_no_indicators_by_group.values[:, -1]
            x_train, x_test, y_train, y_test = train_test_split(x, y, te
st_size=0.4, random_state=42)

            features = fs.SelectKBest(fs.f_regression,
k=num_of_features) #k is number of features.
            features.fit(x_train, y_train)

            selected_features = features.get_support()
            print "Selected Features for ["+ str(i) +"]:"
            print ncds_data_no_indicators_by_group.columns[selected_feat
ures].values
```

In [23]:
```
### Find factors effecting seizures across different sex
list_of_sex = list(ncds_data_no_indicators['n622'].unique())
print list_of_sex

find_features_by_group('n622',list_of_sex)
```

```
[2.0, 1.0, -1.0]
Selected Features for [2.0]:
['n1region' 'n514' 'n383' 'n1819' 'n1827' 'n1828' 'n1400' 'n1399' 'n82
5'
 'n1896' 'n1898' 'dvht07' 'dvrwt07' 'dvwt07' 'OUTCME01']
Selected Features for [1.0]:
['n400' 'n403' 'n1827' 'n604' 'n39' 'n92' 'n1400' 'n1399' 'n1476' 'n154
8'
 'n1551' 'n825' 'n1896' 'OUTCME01' 'OUTCME02']
```

In [24]:
```
### Find factors effecting seizures based on country of birth
list_of_cob = list(ncds_data_no_indicators['COBIRTH'].unique())
print list_of_cob

find_features_by_group('COBIRTH',list_of_cob)
```

```
[1.0, 2.0, 9.0, 3.0, 4.0]
Selected Features for [1.0]:
['n514' 'n1827' 'n604' 'n1400' 'n1476' 'n825' 'n857' 'n2598' 'n1896'
 'n1897' 'n1898' 'dvht07' 'OUTCME01' 'OUTCME02' 'OUTCME03']
Selected Features for [2.0]:
['n1819' 'n604' 'n1400' 'n1239' 'n1543' 'n1537' 'n1551' 'n825' 'n2572'
 'n2573' 'n1896' 'n1898' 'n1964' 'n1966' 'n2154']
Selected Features for [9.0]:
['n95' 'n183' 'n192' 'n559' 'n459' 'n1400' 'n1221' 'n1560' 'n2406' 'n26
30'
 'n1907' 'n2241' 'n2242' 'n2289' 'n2894']
Selected Features for [3.0]:
['n1827' 'n481' 'n1263' 'n1400' 'n1548' 'n1551' 'n1591' 'n1635' 'n2559'
 'n2564' 'n2596' 'n2598' 'n1940' 'n1966' 'n15']
Selected Features for [4.0]:
['n481' 'n482' 'n1301' 'n1364' 'n1449' 'n2539' 'n2599' 'n2610' 'n2611'
 'n1964' 'n2158' 'n2185' 'n2252' 'n2327' 'n2331']
```

In [25]:
```
### Find factors effecting seizures based on ethinic group
list_of_eg = list(ncds_data_no_indicators['ETHNICID'].unique())
print list_of_eg

find_features_by_group('ETHNICID',list_of_eg)
```

```
[1.0, 5.0, 6.0, 3.0, 2.0, 4.0]
Selected Features for [1.0]:
['n514' 'n1827' 'n604' 'n1400' 'n1476' 'n825' 'n2598' 'n1896' 'n1897'
 'n1898' 'n2102' 'dvht07' 'OUTCME01' 'OUTCME02' 'OUTCME03']
Selected Features for [5.0]:
['n500' 'n506' 'n179' 'n346' 'n558' 'n27' 'n454' 'n463' 'n472' 'n1266'
 'n1548' 'DIASTOL' 'PRESENT' 'DTB6' 'ILLNESS']
Selected Features for [6.0]:
['n545' 'n506' 'n532' 'n1338' 'n1406' 'n1447' 'n1454' 'n887' 'n2618'
 'n2621' 'n2622' 'n1935' 'n2239' 'n2241' 'n2242']
Selected Features for [3.0]:
['n542' 'n114' 'n127' 'n131' 'n147' 'n368' 'n380' 'n404' 'n405' 'n409'
 'n1290' 'n1414' 'n855' 'n2294' 'n2295']
Selected Features for [2.0]:
['n545' 'n492' 'n494' 'n236' 'n1844' 'n1866' 'n1128' 'n1167' 'n1681'
 'n1172' 'n857' 'n2380' 'n2437' 'n2592' 'n2257']
```

Step 5: Performance Metric

Our performance metric will be to build a better model than the results from the base models we have used.

In [26]:
```
score_df
```

Out[26]:

	knn	lda	linear	qda	rf	svc	tree	unweigh logistic
overall accuracy	0.909752	0.912177	0.050431	0.851832	0.799300	0.715383	0.895609	0.91594
accuracy on class 0	0.988218	0.988954	0.000000	0.911487	0.856701	0.737113	0.964654	0.99779
accuracy on class 1	0.069401	0.089905	0.000000	0.212934	0.184543	0.482650	0.156151	0.03943

Milestone 5 - Proposal

Propose methodologies and ideas to be implemented, tested and interpreted for your final project. Provide a roughly 1 page outline of what approaches, models, etc. you would like to use for the final project results.

For the final project, we need to work on the following.

Data Management

We will supplement the process where we used means and medians to compute missing values. K-Nearest Neighbor will be used to estimate values for blanks,'none' and 'NA'. As KNN is compute intensive it will be used for our predictive features only.

We will also separate some of the categorical values which have been encoded as numerical. This will allow us to better evaluate their effect on our models. In addition, by recoding them, we can use linear discriminant analysis of the textual data to better categorize factors affecting seizure occurrence into certain groups and with demographics.

Feature Selection

Cross-validation will be used to get a more accurate feature set. The methods we used were valid, but may be biased as we only used a single sample to compute them. Further comparison of ensemble selectors will help us pinpoint the best cluster.

Sweeps will be separated from each other so we can better evaluate correlation between predictors. It appears that some of our features aren't clearly predictors. In the case of hospital stays, seizures may be the cause instead of the effect.

Models

We can optimize all our models with enhanced cross-validation. Our first pass will leverage KFold for train and test splits. Another approach we have discussed is to take the per-sweep data and build separate models, then combine them to have a model which can be extended to non-adolescent subjects.

Our linear prediction model can be improved by further distillation of our features as discussed above. Further gains should be had through better imputation of missing values using KNN. Our model using logistic regression for categorization can be improved by the recoding above, and should also benefit from feature scrubbing.

Analysis

Here we can look back at our dataset and see how the models are performing. We can validate if the model prediction is actually doing a good job or not. Here we will see how we can handle false possitves. Our model tuning step will focus on improving the accuracy on both class 0 and class 1 predictions and thus reducing false posstives.

Visualization and presentation