Classification and Analysis of Brainwaves

April 30, 2017

Nicholas Paul

1 Classification and Analysis of Brainwaves

The following report will compare several sets of brainwave data in order to find which set has the largest variance and can potentially be used to determine brain states in order to control robotic devices. We also analyze several machine learning algorithms and feature selection methods in order to achieve the most reliable and robust performance.

In [44]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

pd.set_option('precision',2)

# Better latex tables
pd.set_option('display.notebook_repr_html', True)

def _repr_latex_(self):
    return "\\begin{center}\n%s\n\\end{center}" % self.to_latex()

pd.DataFrame._repr_latex_ = _repr_latex_
```

2 Datasets

The datasets chosen for the following research compare the brainwaves of a user plying a racing video game. The first set `city_race` is of the user driving a fast vehicle through a timed race. The user must avoid obstacles, drive as quickly as possible, and drive through checkpoints in order to progress. All traffic laws may be broken, the only goal is to get to the end as quickly as possible. The second dataset `city_taxi` is of the user playing the same game but this time driving a taxi around the city. The vehicle and therefore pacing is much slower, the user must try his best to obey traffic laws and not hit any obstacles.

2.0.1 City Race

```
In [45]: city_race = pd.read_csv('csv/jc2_city_race.csv', index_col=0)

         # Trim meaningful data
         city_race = city_race[(np.abs(stats.zscore(city_race)) < 2).all(axis=1)]
         city_race = city_race.reset_index(drop=True)
         city_race = city_race.loc[50:200, :]

         print(city_race.shape)

         city_race.describe()
```

```
(151, 8)
```

Out[45]:

	delta	theta	low4Alpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	1.51e+02	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	2.39e+05	72144.78	26232.77	26948.62	18158.55	12893.51	8915.42	9711.92
std	3.41e+05	95107.28	35775.62	35312.13	24515.54	16047.31	15182.13	20558.96
min	1.18e+03	435.00	328.00	283.00	108.00	125.00	28.00	21.00
25%	2.37e+04	18232.00	5994.00	8301.50	4553.50	3928.00	1834.00	1125.50
50%	6.60e+04	35254.00	14467.00	15352.00	9276.00	7178.00	3016.00	1852.00
75%	2.89e+05	78020.50	30059.50	31495.50	21522.50	14578.00	6850.50	5162.00
max	1.33e+06	532631.00	219850.00	257532.00	158932.00	96936.00	74643.00	133085.00

2.0.2 City Taxi

```
In [46]: city_taxi = pd.read_csv('csv/jc2_city_taxi.csv', index_col=0)

         # Trim meaningful data
         city_taxi = city_taxi[(np.abs(stats.zscore(city_taxi)) < 2).all(axis=1)]
         city_taxi = city_taxi.reset_index(drop=True)
         city_taxi = city_taxi.loc[50:200, :]
```

```
print(city_taxi.shape)

city_taxi.describe()
```

```
(151, 8)
```

Out[46]:

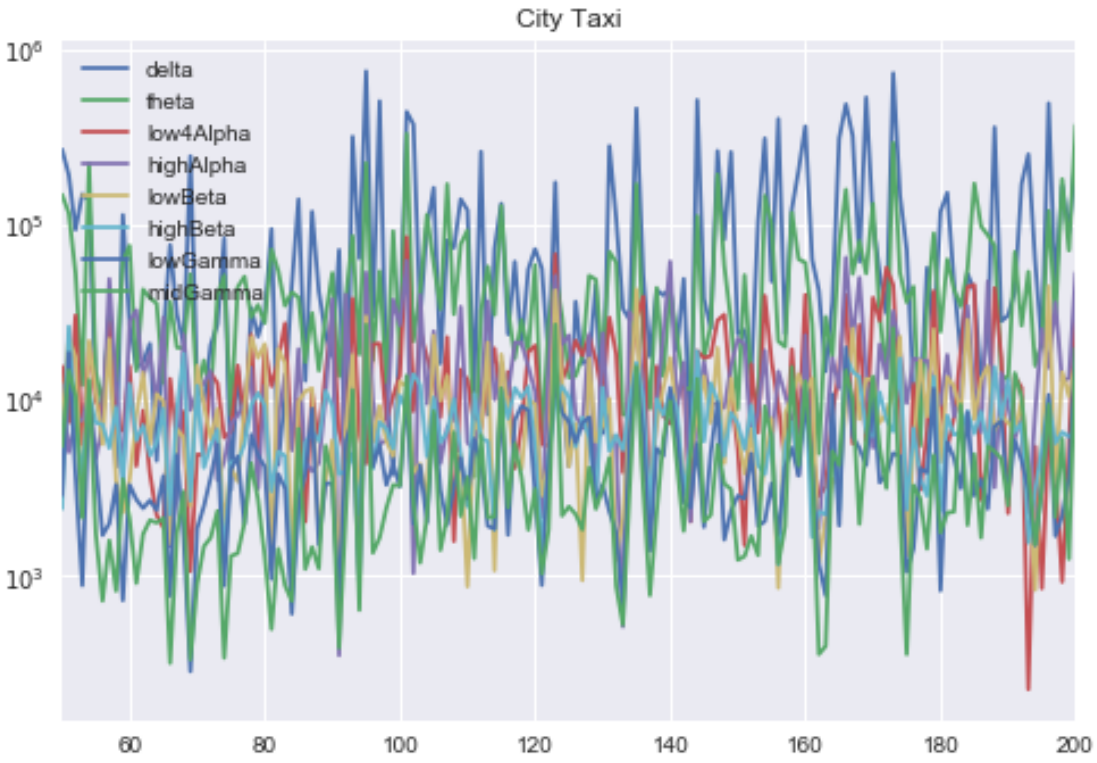
	delta	theta	low4Alpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	151.00	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	107598.15	55566.26	14914.06	15616.68	9761.29	7359.72	4688.62	4433.48
std	144554.68	59837.68	13591.58	13572.23	8006.70	4127.33	3768.55	5012.39
min	3094.00	3669.00	226.00	353.00	832.00	776.00	286.00	319.00
25%	20649.00	17823.50	5603.50	6538.50	4155.50	4798.50	2122.00	1551.50
50%	44034.00	35624.00	10610.00	10992.00	7798.00	6580.00	3632.00	2360.00
75%	127171.00	62708.00	19125.50	20812.50	13169.00	9952.00	5760.50	5122.00
max	747645.00	368453.00	85027.00	64987.00	44553.00	26302.00	19998.00	26990.00

2.1 Data Visualization

Initial plots of the data plotted on a log scaled axis. The raw data itself isn't very informative. There is no apparent distinct difference between the two datasets.

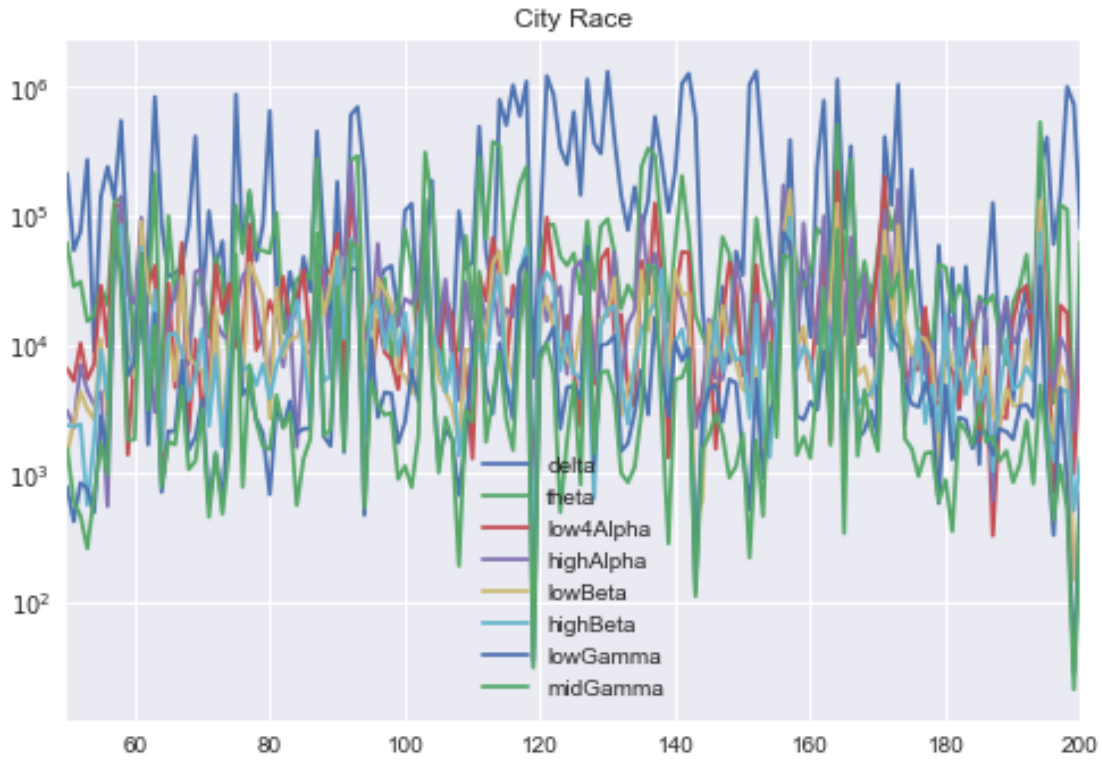
```
In [47]: plt.figure()
         city_taxi.plot()
         plt.yscale('log')
         plt.title('City Taxi')
         plt.show()
```

```
<matplotlib.figure.Figure at 0x7f0dec11c470>
```




```
In [48]: plt.figure()
         city_race.plot()
         plt.yscale('log')
         plt.title('City Race')
         plt.show()
```

```
<matplotlib.figure.Figure at 0x7f0dec080080>
```




```
In [49]: print(city_taxi.values.shape, city_race.values.shape)
```

```
(151, 8) (151, 8)
```

2.2 Set up arrays for training the classifier

```
In [50]: def combine_train_test(data1, data2, test_size=0.3):
             """
             Combine two dataframes into a training and testing data
             """

             # Get value arrays from the dataframes
             X_1 = data1.values
             X_2 = data2.values

             # classes: 1 = all_about_us, 2 = people_are_awesome

             y_1 = np.empty(len(X_1), dtype='int')
             y_1.fill(1)

             y_2 = np.empty(len(X_2), dtype='int')
             y_2.fill(2)
```

```
            # Combine the data
            # 2d, use vstack
            X_combined = np.vstack( (X_1, X_2))

            # 1d use concatenate
            y_combined = np.concatenate( (y_1, y_2))

            return train_test_split(X_combined, y_combined, test_size=test_size)
```

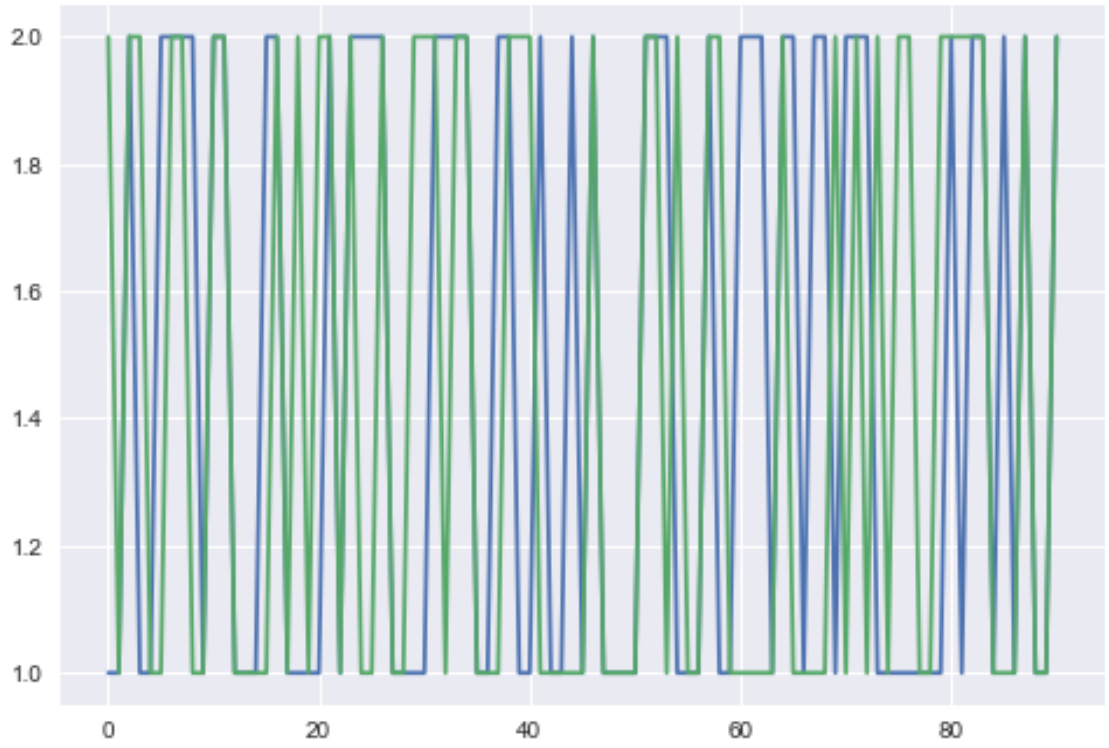
```
In [51]: X_train_rc, X_test_rc, y_train_rc, y_test_rc = combine_train_test(city_tax

         # Standard Scalar
         # Map the data onto a normal curve
         stdsc = StandardScaler()
         X_train_rc_std = stdsc.fit_transform(X_train_rc)
         X_test_rc_std = stdsc.fit_transform(X_test_rc)
```

```
/home/nick/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:429: D
  warnings.warn(msg, _DataConversionWarning)
```

2.3 The Random Forest Classifier

```
In [52]: rf = RandomForestClassifier(criterion='entropy', n_estimators=1000, n_jobs
         rf.fit(X_train_rc_std, y_train_rc)
         y_pred_rc = rf.predict(X_test_rc_std)
```

```
In [53]: plt.figure()

         domain = np.arange(len(y_pred_rc))

         # plot the predictions along-side the actual values
         plt.plot(domain, y_test_rc, domain, y_pred_rc)

         plt.show()
```




```
In [54]: # Accuracy score of the classifier
         accuracy_score(y_pred_rc, y_test_rc)
```

```
Out[54]: 0.61538461538461542
```

3 Dimensionality Reduction (via Feature Selection)

After running several classifiers on the data, we can see that it is far too noisy to infer any meaningful results. In this section we will take several approaches to reduce the dimensionality of the data.

3.1 Analyzing Feature Importance

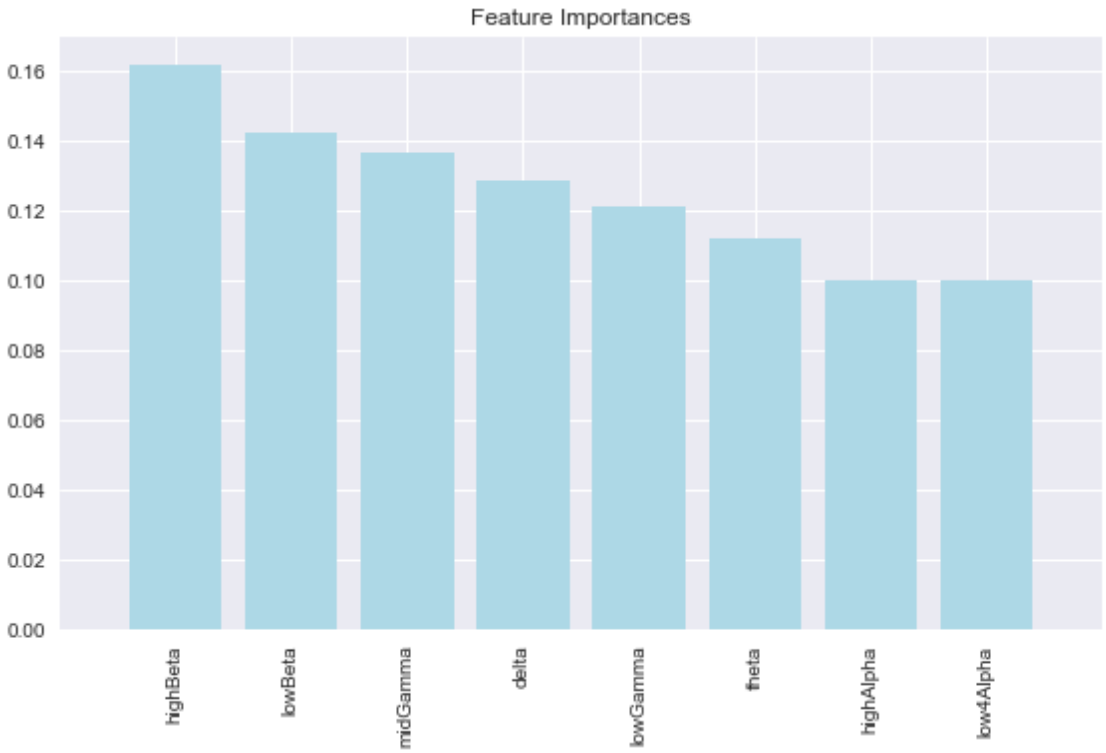
We use the random forest classifier to analyze the importance of each of the features. We can see that the importance of all features, I_f, is below the threshold $I_f < 0.2$. This means that the data is quite similar and each of the datapoints on their own are not largely significant.

```
In [55]: feat_labels = city_race.columns[0:]
         forest = RandomForestClassifier(n_estimators=1000,
                                         random_state=0,
                                         n_jobs=-1)

         forest.fit(X_train_rc, y_train_rc)
```

```
        importances = forest.feature_importances_
        indices = np.argsort(importances)[::-1]
        for f in range(X_train_rc.shape[1]):
            print("%2d) %-*s %f" % (f + 1, 30,
                                    feat_labels[indices[f]],
                                    importances[indices[f]]))
```

```
 1) highBeta                       0.161527
 2) lowBeta                        0.141914
 3) midGamma                       0.136287
 4) delta                          0.128246
 5) lowGamma                       0.120948
 6) theta                          0.111869
 7) highAlpha                      0.099610
 8) low4Alpha                      0.099599
```

In [56]:
```
plt.figure()
plt.title('Feature Importances')
plt.bar(range(X_train_rc.shape[1]),
        importances[indices],
        color='lightblue',
        align='center')
plt.xticks(range(X_train_rc.shape[1]),
           feat_labels[indices], rotation=90)
plt.xlim([-1, X_train_rc.shape[1]])
plt.tight_layout()
plt.show()
```



3.2 Sequential Backward Selection

Next, we implement Sequential Backwards Selection (SBS) to determine the top k most important features.

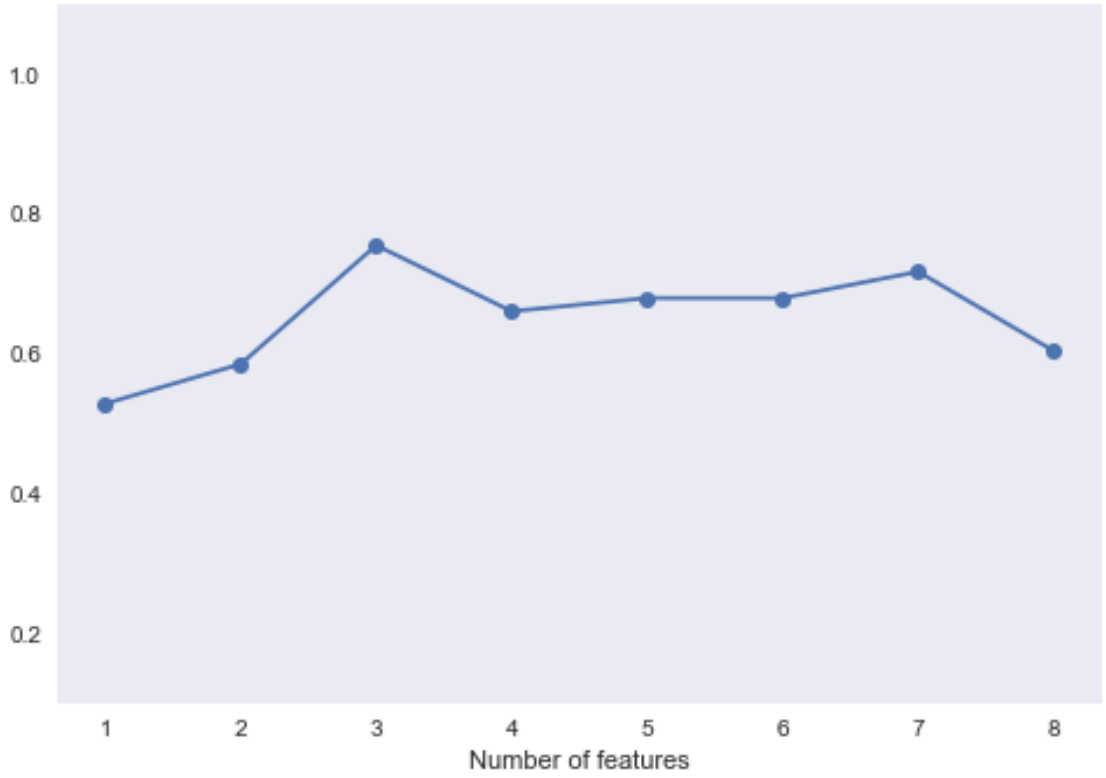
```
In [57]: import numpy as np
         from sklearn.base import clone
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from itertools import combinations

         class SBS():
             def __init__(self, estimator, k_features,
                          scoring=accuracy_score,
                          test_size=0.25, random_state=1):
                 self.scoring = scoring
                 self.estimator = clone(estimator)
                 self.k_features = k_features
                 self.test_size = test_size
                 self.random_state = random_state

             def fit(self, X, y):
                 X_train, X_test, y_train, y_test = \
```

```
            train_test_split(X, y, test_size=self.test_size,
                             random_state=self.random_state)

        dim = X_train.shape[1]
        self.indices_ = tuple(range(dim))
        self.subsets_ = [self.indices_]
        score = self._calc_score(X_train, y_train,
                                 X_test, y_test, self.indices_)

        self.scores_ = [score]

        while dim > self.k_features:
            scores = []
            subsets = []

            for p in combinations(self.indices_, r=dim-1):
                score = self._calc_score(X_train, y_train,
                                         X_test, y_test, p)
                scores.append(score)
                subsets.append(p)

            best = np.argmax(scores)
            self.indices_ = subsets[best]
            self.subsets_.append(self.indices_)
            dim -= 1

            self.scores_.append(scores[best])

        self.k_score_ = self.scores_[-1]

        return self

    def transform(self, X):
        return X[:, self.indices_]

    def _calc_score(self, X_train, y_train,
                    X_test, y_test, indices):
        self.estimator.fit(X_train[:, indices], y_train)
        y_pred = self.estimator.predict(X_test[:, indices])
        score = self.scoring(y_test, y_pred)
        return score
```

We run a KNN classifier on the datasets D_k where D_k is the dataset with all but the top k features removed. We can see that the variance is quite low indicating that feature selection is not a good choice for dimensionality reduction.

```
In [58]: knn = KNeighborsClassifier(n_neighbors = 3)
         sbs = SBS(knn, k_features=1)
```

```
sbs.fit(X_train_rc_std, y_train_rc)
k_feat = [len(k) for k in sbs.subsets_]
plt.plot(k_feat, sbs.scores_, marker='o')
plt.ylim([0.1, 1.1])
plt.xlabel('Accuracy')
plt.xlabel('Number of features')
plt.grid()
plt.show()
```



In [59]:
```
# Which 5 features yeilded good performance on the dataset?
k5 = list(sbs.subsets_[3])
print(city_race.columns[0:][k5])
```

```
Index(['delta', 'theta', 'highAlpha', 'lowBeta', 'lowGamma'], dtype='object')
```

In [60]:
```
# Evaluate the performance of the KNN on the original set
knn.fit(X_train_rc_std, y_train_rc)
print('Training accuracy: ', knn.score(X_train_rc_std, y_train_rc))
print('Test accuracy: ', knn.score(X_test_rc_std, y_test_rc))
```

```
Training accuracy:  0.867298578199
Test accuracy:  0.582417582418
```

```
In [61]: # Evaluate the performance of the KNN on the selected feature set
         knn.fit(X_train_rc_std[:, k5], y_train_rc)
         print('Training accuracy: ', knn.score(X_train_rc_std[:, k5], y_train_rc))
         print('Test accuracy: ', knn.score(X_test_rc_std[:, k5], y_test_rc))

Training accuracy:  0.78672985782
Test accuracy:  0.582417582418
```

4 Dimensionality Reduction (via Feature Extraction)

Feature selection did not yield significant results for training a classifier. Here we will use principal component analysis (PCA) to extract meaningful features from the data.

First we define a function that will allow us to visualize the performance of a classifier.

```
In [62]: from matplotlib.colors import ListedColormap

         def versiontuple(v):
             return tuple(map(int, (v.split("."))))


         def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02

             # setup marker generator and color map
             markers = ('s', 'x', 'o', '^', 'v')
             colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
             cmap = ListedColormap(colors[:len(np.unique(y))])

             # plot the decision surface
             x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
             x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
             xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                                    np.arange(x2_min, x2_max, resolution))
             Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
             Z = Z.reshape(xx1.shape)
             plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
             plt.xlim(xx1.min(), xx1.max())
             plt.ylim(xx2.min(), xx2.max())

             for idx, cl in enumerate(np.unique(y)):
                 plt.scatter(x=X[y == cl, 0],
                             y=X[y == cl, 1],
                             alpha=0.6,
                             c=cmap(idx),
                             edgecolor='black',
                             marker=markers[idx],
                             label=cl)
```

```
            # highlight test samples
            if test_idx:
                # plot all samples
                if not versiontuple(np.__version__) >= versiontuple('1.9.0'):
                    X_test, y_test = X[list(test_idx), :], y[list(test_idx)]
                    warnings.warn('Please update to NumPy 1.9.0 or newer')
                else:
                    X_test, y_test = X[test_idx, :], y[test_idx]

                plt.scatter(X_test[:, 0],
                            X_test[:, 1],
                            c='',
                            alpha=1.0,
                            edgecolor='black',
                            linewidths=1,
                            marker='o',
                            s=55, label='test set')
```

4.1 Principal Component Analysis

We reduce the dimension of the data to $k = 2$ and perform additional analysis. We also define a function `plot_compare_test_train()` to plot and test the performance of the classifier on the PCA data.

```
In [63]: from sklearn.decomposition import PCA

         # Principal component analysis
         pca = PCA(n_components=2)
         X_train_rc_pca = pca.fit_transform(X_train_rc_std)
         X_test_rc_pca = pca.fit_transform(X_test_rc_std)
```

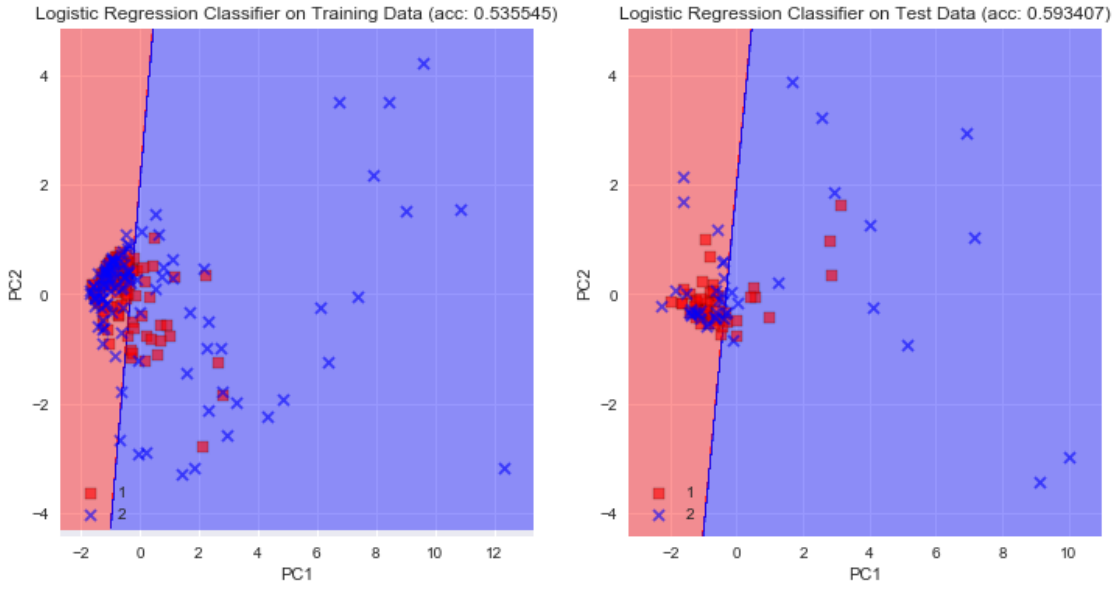
```
In [64]: def plot_compare_test_train(X_train, X_test, y_train, y_test, clsf, cname)

             clsf.fit(X_train, y_train)

             train_accuracy = clsf.score(X_train, y_train)
             test_accuracy = clsf.score(X_test, y_test)

             plt.figure(figsize=(12,6))

             ax0 = plt.subplot(121) #nrows, ncols, plot_number
             plot_decision_regions(X_train, y_train, classifier=clsf)
             plt.xlabel('PC1')
             plt.ylabel('PC2')
             plt.legend(loc='lower left')
             plt.title("%s Classifier on Training Data (acc: %f)" % (cname, train_a

             plt.subplot(122, sharey=ax0) #nrows, ncols, plot_number
```

```
    plot_decision_regions(X_test, y_test, classifier=clsf)
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.legend(loc='lower left')
    plt.title("%s Classifier on Test Data (acc: %f)" % (cname, test_accura

    plt.show()
```

4.1.1 Logistic Regression

The figure below indicates that the data is not linearly separable and therefore a simple logistic regression classifier will not be sufficient in classifying the data.

```
In [65]: lr = LogisticRegression()

         plot_compare_test_train(X_train_rc_pca, X_test_rc_pca, y_train_rc, y_test_
```

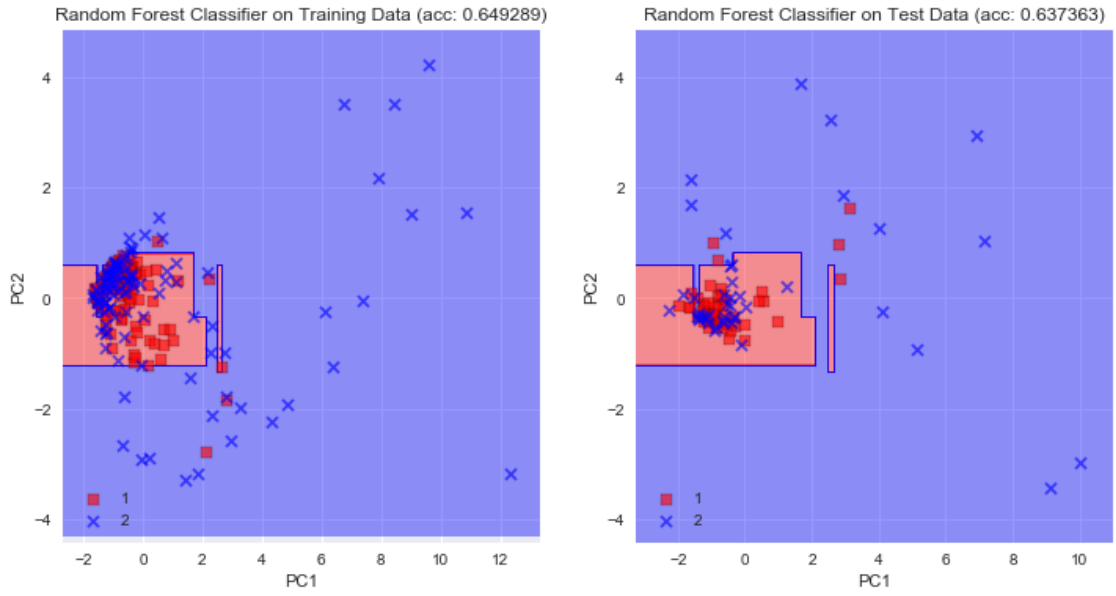


4.1.2 RandomForest

Since the data is very noisy, we must carefully tune the parameters to avoid overfitting or underfitting the data.

We can see, that even with a low number of trees (6) and a low max tree depth (3) we have significant overfitting of the data. The training set has an accuracy of approximately 70% and the test accuracy is at about 60%.

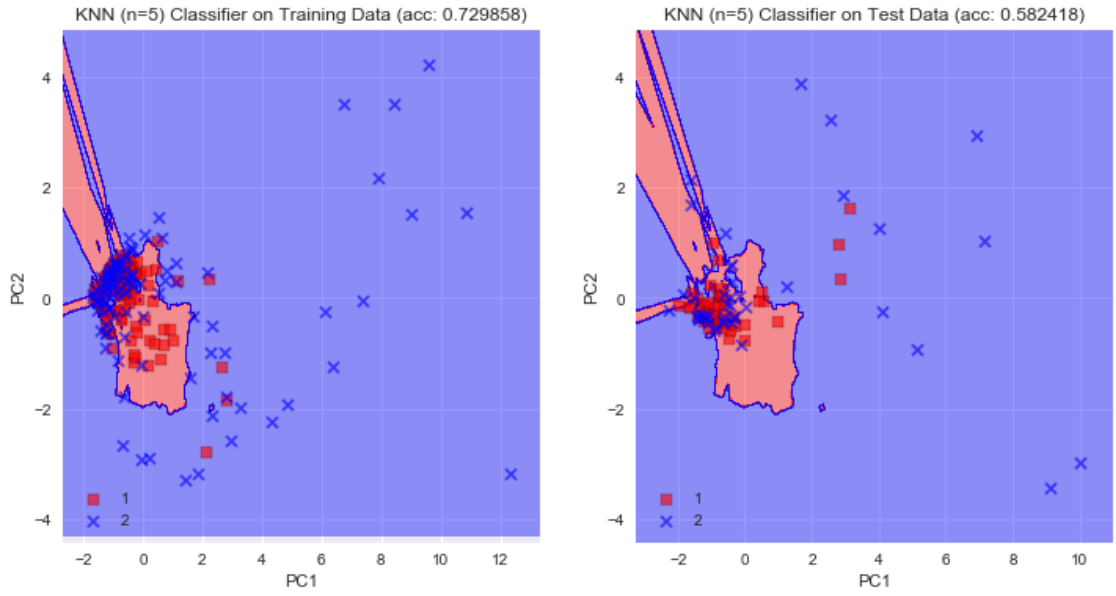
```
In [66]: rf = RandomForestClassifier(criterion='entropy', n_estimators=6, max_depth

         plot_compare_test_train(X_train_rc_pca, X_test_rc_pca, y_train_rc, y_test_
```



4.1.3 K Nearest Neighbors

The **K Nearest Neighbors Classifier** performs similar to the random forest.

```
In [67]: knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')

         plot_compare_test_train(X_train_rc_pca, X_test_rc_pca, y_train_rc, y_test_
```



5 Additional Datasets

5.1 Set 1: Sad vs. Happy

5.1.1 People Are Awesome (happy)

A recording of the user listening to a "people are awesome" YouTube video. The video is fast paced and exciting and depicts people achieving many great feats.

```
In [68]: people_are_awesome = pd.read_csv('people_are_awesome.csv', index_col=0)

         # Trim meaningful data
         people_are_awesome = people_are_awesome[(np.abs(stats.zscore(people_are_aw
         people_are_awesome = people_are_awesome.reset_index(drop=True)
         people_are_awesome = people_are_awesome.loc[50:200, :]

         print(people_are_awesome.shape)

         people_are_awesome.describe()
```

```
(151, 8)
```

Out[68]:

	delta	theta	lowAlpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	151.00	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	59475.69	32396.41	13532.06	14911.13	9435.95	7592.79	4252.99	2743.69
std	97857.58	23429.53	11381.82	9958.77	6054.98	3796.78	2585.25	1599.28
min	1982.00	3852.00	455.00	1601.00	1432.00	658.00	455.00	194.00
25%	13574.00	15950.50	5298.50	7043.00	5022.50	5257.50	2560.50	1681.50
50%	26538.00	24582.00	10970.00	12966.00	8262.00	7218.00	3760.00	2385.00
75%	56199.50	43495.50	17609.50	19431.50	12767.00	9399.50	5415.00	3475.00
max	605903.00	135384.00	69335.00	59943.00	32732.00	23261.00	12606.00	10406.00

5.1.2 All About Us (sad)

A recording of the user watching the music video for *All About Us* by He Is We. The music video is generally regarded to be sad.

```
In [69]: all_about_us = pd.read_csv('all_about_us.csv', index_col=0)

         # Trim meaningful data
         # all_about_us = all_about_us[(np.abs(stats.zscore(all_about_us)) < 3).all
         all_about_us = all_about_us.reset_index(drop=True)
         all_about_us = all_about_us.loc[50:200, :]

         print(all_about_us.shape)
```

```
all_about_us.describe()
```

```
(151, 8)
```

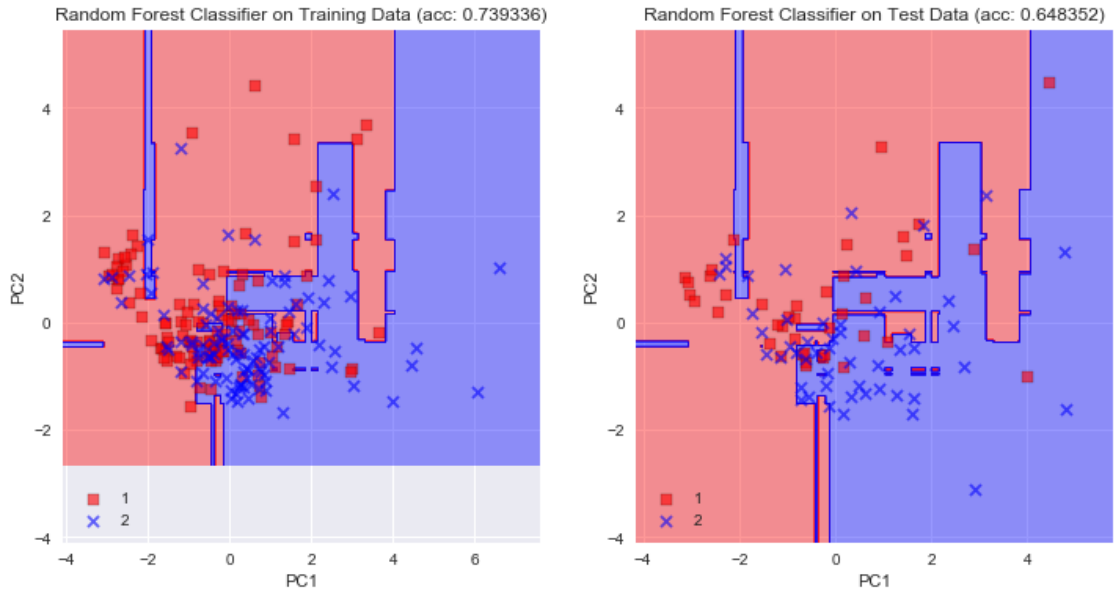
Out[69]:

	delta	theta	lowAlpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	1.51e+02	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	8.88e+04	32771.99	12670.05	11903.36	7257.78	6342.58	3106.07	1944.47
std	1.90e+05	32398.40	10887.94	10353.36	5424.50	3565.89	1946.14	1254.21
min	3.12e+03	2753.00	604.00	454.00	696.00	645.00	227.00	94.00
25%	1.59e+04	13762.50	4250.00	4417.50	3282.50	3812.00	1842.50	1079.50
50%	3.06e+04	23413.00	8992.00	9188.00	6074.00	6011.00	2934.00	1844.00
75%	7.16e+04	37112.00	18449.50	16096.00	9056.00	8693.50	3859.50	2553.50
max	1.47e+06	220409.00	53691.00	57161.00	31155.00	21644.00	13856.00	8555.00

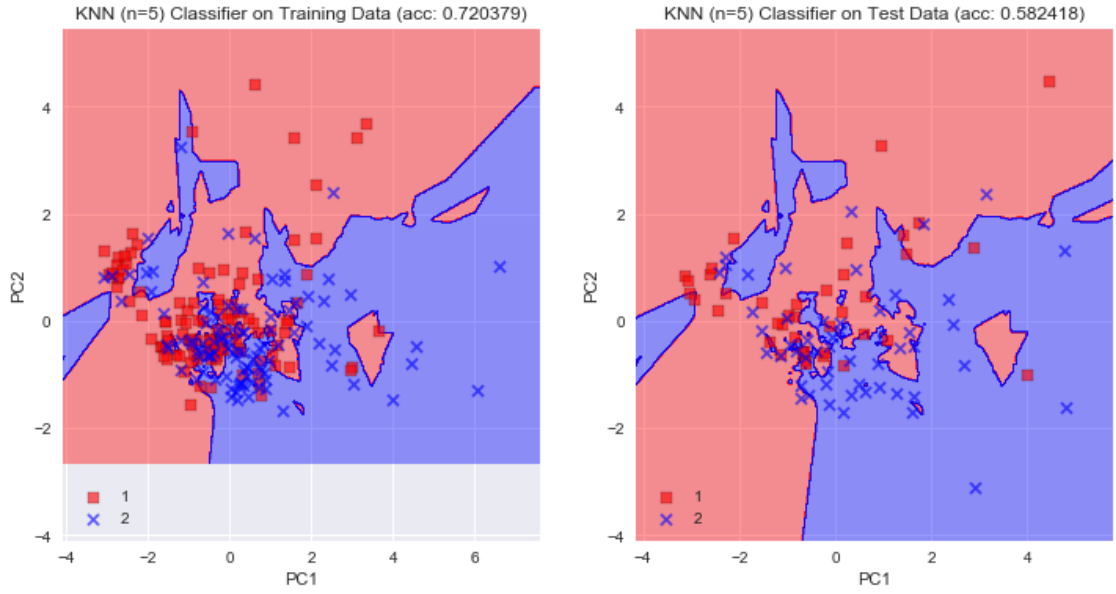
In [70]:
```
X_train_hs, X_test_hs, y_train_hs, y_test_hs = combine_train_test(all_abou

# Standard Scalar
# Map the data onto a normal curve
stdsc = StandardScaler()
X_train_hs_std = stdsc.fit_transform(X_train_hs)
X_test_hs_std = stdsc.fit_transform(X_test_hs)

# Principal component analysis
pca = PCA(n_components=2)
X_train_hs_pca = pca.fit_transform(X_train_hs_std)
X_test_hs_pca = pca.fit_transform(X_test_hs_std)
```

```
/home/nick/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:429: D
  warnings.warn(msg, _DataConversionWarning)
```

In [71]:
```
rf = RandomForestClassifier(criterion='entropy', n_estimators=10, max_dept

plot_compare_test_train(X_train_hs_pca, X_test_hs_pca, y_train_hs, y_test_
```




```
In [72]: knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')

         plot_compare_test_train(X_train_hs_pca, X_test_hs_pca, y_train_hs, y_test_
```



5.2 Set 2: Thinking Analytically vs. Erratically

5.2.1 Rubik's Cube (Analytic)

The following dataset is of the user attempting to solve a rubik's cube.

```
In [73]: rubiks = pd.read_csv('csv/rubiks_cube.csv', index_col=0)

         # Trim meaningful data
         rubiks = rubiks[(np.abs(stats.zscore(rubiks)) < 3).all(axis=1)]
         rubiks = rubiks.reset_index(drop=True)
         rubiks = rubiks.loc[50:200, :]

         print(rubiks.shape)

         rubiks.describe()
```

```
(151, 8)
```

Out[73]:

	delta	theta	low4Alpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	1.51e+02	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	4.40e+05	83935.82	21442.61	21482.09	15652.50	12986.66	5960.49	2914.17
std	4.88e+05	70169.17	21716.45	17924.05	13298.85	11538.01	5070.52	2299.99
min	2.10e+01	4.00	2.00	1.00	0.00	2.00	1.00	0.00
25%	8.62e+04	32127.00	7091.00	8167.00	6567.00	4717.50	2352.00	1213.00
50%	1.92e+05	66472.00	15031.00	16731.00	13066.00	10073.00	4637.00	2390.00
75%	6.50e+05	115449.50	26271.50	29704.00	20875.00	18539.50	8285.00	3669.00
max	2.12e+06	327533.00	115771.00	85071.00	73770.00	74907.00	30429.00	12247.00

5.3 Spelunky (Erratic)

The following dataset is a recording of me playing the video game *Spelunky*. The game requires split-second decisions, fast reactions, and little puzzle solving.

```
In [74]: spelunky = pd.read_csv('csv/spelunky.csv', index_col=0)

         # Trim meaningful data
         spelunky = spelunky[(np.abs(stats.zscore(spelunky)) < 3).all(axis=1)]
         spelunky = spelunky.reset_index(drop=True)
         spelunky = spelunky.loc[50:200, :]

         print(spelunky.shape)

         spelunky.describe()
```

```
(151, 8)
```

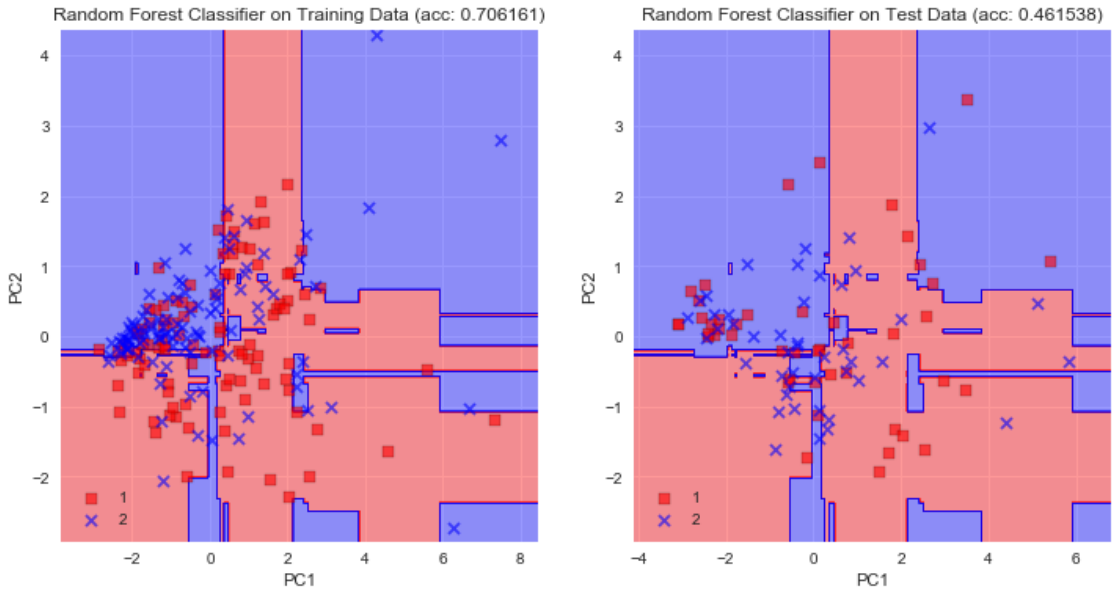
Out[74]:

	delta	theta	low4Alpha	highAlpha	lowBeta	highBeta	lowGamma	midGamma
count	1.51e+02	151.00	151.00	151.00	151.00	151.00	151.00	151.00
mean	2.87e+05	73259.93	19483.08	20208.11	13986.67	10636.50	5112.46	3685.46
std	3.73e+05	76602.32	19428.26	16281.35	15325.82	9226.43	4872.81	4963.15
min	2.75e+03	2988.00	145.00	871.00	553.00	607.00	125.00	116.00
25%	5.40e+04	25668.50	7171.50	7246.50	4648.50	4532.00	1953.50	1247.00
50%	1.15e+05	45936.00	13268.00	15106.00	9833.00	8311.00	4057.00	2381.00
75%	3.32e+05	95157.00	25857.00	27075.00	17186.00	13999.00	6804.50	4397.50
max	1.70e+06	480101.00	137553.00	68665.00	117918.00	58432.00	34582.00	50012.00

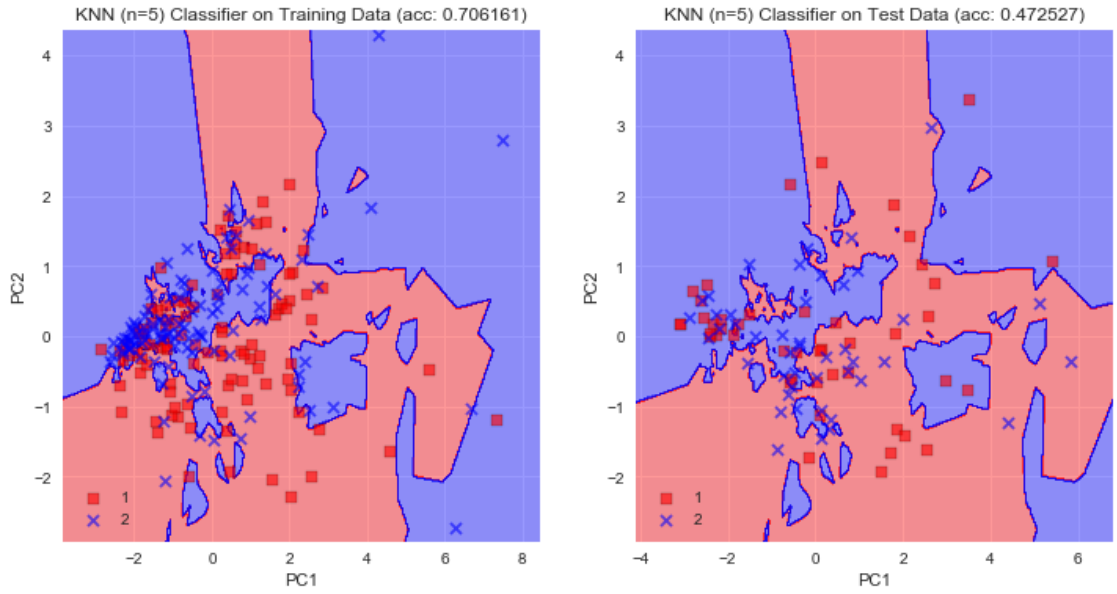
In [75]:
```
X_train_ir, X_test_ir, y_train_ir, y_test_ir = combine_train_test(rubiks,

        # Standard Scalar
        # Map the data onto a normal curve
        stdsc = StandardScaler()
        X_train_ir_std = stdsc.fit_transform(X_train_ir)
        X_test_ir_std = stdsc.fit_transform(X_test_ir)

        # Principal component analysis
        pca = PCA(n_components=2)
        X_train_ir_pca = pca.fit_transform(X_train_ir_std)
        X_test_ir_pca = pca.fit_transform(X_test_ir_std)
```

```
/home/nick/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:429: D
  warnings.warn(msg, _DataConversionWarning)
```

In [76]:
```
rf = RandomForestClassifier(criterion='entropy', n_estimators=10, max_dept

        plot_compare_test_train(X_train_ir_pca, X_test_ir_pca, y_train_ir, y_test_
```



```
In [77]: knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')

         plot_compare_test_train(X_train_ir_pca, X_test_ir_pca, y_train_ir, y_test_
```



6 Results Summary

```
In [78]: summary = pd.DataFrame(
             columns= ['RandomForest', 'RandomForest (DR)', 'KNN', 'KNN (DR)'],
             index = ['Reckless vs Careful', 'Happy vs Sad', 'Interpretive vs React

         def train_test_acc(X_train, X_test, y_train, y_test, cls):
             cls.fit(X_train, y_train)
             return '%.3f, %.3f' % (cls.score(X_train, y_train), cls.score(X_test,

         # Standard knn with p=2 and nn=5
         def new_knn(p=2, n_neighbors=5):
             return KNeighborsClassifier(n_neighbors=n_neighbors,
                                         p=p, metric='minkowski')

         # Standard rf with trees=6 and depth=4
         def new_rf(n_estimators=5, max_depth=4):
             return RandomForestClassifier(criterion='entropy',
                                           n_estimators=n_estimators,
                                           max_depth=max_depth,
```

```
                                        random_state=11,
                                        n_jobs=-1)
```

In [79]:
```
# Reckless vs Careful

summary.set_value('Reckless vs Careful', 'RandomForest',
                  train_test_acc(X_train_rc, X_test_rc, y_train_rc, y_test

summary.set_value('Reckless vs Careful', 'RandomForest (DR)',
                  train_test_acc(X_train_rc_pca, X_test_rc_pca, y_train_rc

summary.set_value('Reckless vs Careful', 'KNN',
                  train_test_acc(X_train_rc, X_test_rc, y_train_rc, y_test

summary.set_value('Reckless vs Careful', 'KNN (DR)',
                  train_test_acc(X_train_rc_pca, X_test_rc_pca, y_train_rc


# Happy vs Sad

summary.set_value('Happy vs Sad', 'RandomForest',
                  train_test_acc(X_train_hs, X_test_hs, y_train_hs, y_test

summary.set_value('Happy vs Sad', 'RandomForest (DR)',
                  train_test_acc(X_train_hs_pca, X_test_hs_pca, y_train_hs

summary.set_value('Happy vs Sad', 'KNN',
                  train_test_acc(X_train_hs, X_test_hs, y_train_hs, y_test

summary.set_value('Happy vs Sad', 'KNN (DR)',
                  train_test_acc(X_train_hs_pca, X_test_hs_pca, y_train_hs



# Interpretive vs Reactive

summary.set_value('Interpretive vs Reactive', 'RandomForest',
                  train_test_acc(X_train_ir, X_test_ir, y_train_ir, y_test

summary.set_value('Interpretive vs Reactive', 'RandomForest (DR)',
                  train_test_acc(X_train_ir_pca, X_test_ir_pca, y_train_ir

summary.set_value('Interpretive vs Reactive', 'KNN',
                  train_test_acc(X_train_ir, X_test_ir, y_train_ir, y_test

summary.set_value('Interpretive vs Reactive', 'KNN (DR)',
                  train_test_acc(X_train_ir_pca, X_test_ir_pca, y_train_ir
```

```
a=1 #hide cell output
```

6.0.1 Training and Test Accuracy for various algorithms applied to each dataset with and without dimensionality reduction

```
In [80]: summary
```

```
Out[80]:
```

	RandomForest	RandomForest (DR)	KNN	KNN (DR)
Reckless vs Careful	0.763, 0.659	0.659, 0.626	0.735, 0.560	0.730, 0.582
Happy vs Sad	0.701, 0.484	0.730, 0.637	0.664, 0.571	0.720, 0.582
Interpretive vs Reactive	0.782, 0.429	0.701, 0.538	0.687, 0.484	0.706, 0.473

Random forest had slightly less overfitting and therefore slightly higer test scores after dimensionality reduction was applied. There is little difference between KNN and KNN (DR).

7 Limitations and Conclusion

There were many likely limitations that caused low performance among the classifiers. The primary limitation is the brain scanner itself. The Mindwave Mobile is not intended to be used for research but as a toy for kids and adults. Another possible limitation is the size of the recorded datasets. The Mindwave Mobile records approximately 2 datapoints per second. This means that after doing each activity for about 5 minutes, we end up with approximately 150 meaningful datapoints in each set.

Despite these limitations, we were able to see moderate performance from both the training and testing data for each of the dataset comparisons. This suggests that with proper equipment and more robust tests, determining a reliable brain state based on brain waves is likely achievable.