Question 1:

TRAINING ERROR RATES FOR TWO PASSES

Error Rate Normal : 0.03853211009174312
Error Rate Voted: 0.03853211009174312
Error Rate Average: 0.05504587155963303

TESTING ERROR RATES FOR TWO PASSES

Error Rate Normal : 0.0610079575596817
Error Rate Voted: 0.05305039787798409
Error Rate Average: 0.07957559681697612

TRAINING ERROR RATES FOR THREE PASSES

Error Rate Normal: 0.02018348623853211
Error Rate Voted: 0.031192660550458717
Error Rate Average: 0.03577981651376147

TESTING ERROR RATES FOR THREE PASSES

Error Rate Normal : 0.04509283819628647
Error Rate Voted: 0.04774535809018567
Error Rate Average: 0.0610079575596817

TRAINING ERROR RATES FOR FOUR PASSES

Error Rate Normal: 0.01834862385321101
Error Rate Voted: 0.02110091743119266
Error Rate Average: 0.03211009174311927

TESTING ERROR RATES FOR FOUR PASSES

Error Rate Normal : 0.04774535809018567
Error Rate Voted: 0.04509283819628647
Error Rate Average: 0.05305039787798409

Question 2:

Strong Positive Predictors(rec.sport.baseball): “game”, “team”, “he”
Strong Negative Predictors (comp.Windows): “file”, “program”, “line”

Question 3: Confusion Matrix

0.16574586	**0.18647166**	**0.44933078**	**0.24758221**	**0.14864865**	**0.03474903**
0.79558011	**0.00365631**	**0.00956023**	**0.00386847**	**0.0019305**	**0.00284091**
0.01104972	**0.76599634**	**0.03441683**	**0.01547389**	**0.01351351**	**0.01988636**
0.	**0.00182815**	**0.43403442**	**0.00580271**	**0.00579151**	**0.00852273**
0.01473297	**0.01462523**	**0.01529637**	**0.71760155**	**0.00579151**	**0.00852273**
0.0092081	**0.02010969**	**0.03441683**	**0.00773694**	**0.78957529**	**0.10511364**
0.00368324	**0.00731261**	**0.02294455**	**0.00193424**	**0.03474903**	**0.50568182**

1. **The perceptron has the largest accuracy for the class 1 (comp.windows)**
2. **The perceptron has the lowest accuracy for the class 3 (sci.med)**
3. **Class 6 is mistaken for Class 5 the most often**

PA 3: Multiclass Classification

In [1]:

```
import pandas as pd
import numpy as np
import random as rand
```

In [2]:

```
# Load in Datasets

f = open('./data/pa3dictionary.txt', 'r')
words = f.readlines()
f.close()

training_data = pd.read_csv('./data/pa3train.txt', sep=" ", header=None)
testing_data = pd.read_csv('./data/pa3test.txt', sep=" ", header=None)

label_index = training_data.shape[1] - 1
training_data_q1 = training_data.loc[training_data[label_index] <= 2]
testing_data_q1 = testing_data.loc[testing_data[label_index] <= 2]

# Map all 2 and 1 values to 1, -1

for i, row in training_data_q1.iterrows():
    if row[label_index] == 1:
        row[label_index] = -1
    else:
        row[label_index] = 1

# Map all 2 and 1 values to 1, -1
for i, row in testing_data_q1.iterrows():
    if row[label_index] == 1:
        row[label_index] = -1
    else:
        row[label_index] = 1
```

In [3]:

```
def calc_error(Y_pred, Y_label):
    # Calculate Error Rate for Predicted Labels
    error = [0 for x,y in zip(Y_pred,Y_label) if x != y]
    error_rate = len(error)/len(Y_pred)
    return error_rate
```

In [4]:

```
class perceptron:

    # Constructor
    def __init__(self):
        self.w = np.empty((0,0))
        self.w_list = []
        self.c_list = []

    # Make Predictions
    def predict(self, test_data, model):
        predictions = []
        # Error Checking
        if model not in ['normal','voted','averaged']:
            print("ERROR")
            return []
        for i,row in test_data.iterrows():
            sample_point = row[:test_data.shape[1]-1]
            if model == 'normal':
                prediction = np.sign(np.dot(self.w, sample_point))
                if prediction == 0:
                    prediction = rand.choice([1,-1])
                predictions.append(prediction)
            elif model == 'voted':
                prediction = np.sign(sum(c*(np.sign(np.dot(w,sample_poin
t))) for c, w in zip(self.c_list, self.w_list)))
                if prediction == 0:
                    prediction = rand.choice([1,-1])
                predictions.append(prediction)
            elif model == 'averaged':
                prediction = np.sign(np.dot((sum(c*w for c, w in zip(sel
f.c_list, self.w_list))),sample_point))
                if prediction == 0:
                    prediction = rand.choice([1,-1])
                predictions.append(prediction)
        return predictions


    # Train Classifier
    def train_multiclass(self, data, positive_label, num_passes):
        w_list = []
        c_list = []
        c = 0
        w = np.zeros((data.shape[1]-1,))
        for p in range(num_passes):
            for i, row in data.iterrows():
                X = row[:data.shape[1]-1]
                Y = row[data.shape[1]-1]
                if Y == positive_label:
                    Y = 1
                else:
                    Y = -1
                if Y*(np.dot(w,np.transpose(X))) <= 0:
                    # Adjust decision boundary, otherwise keep it
                    w = np.add(w,Y*X)
                    w_list.append(w)
                    c_list.append(c)
                    c = 1
                else:
                    c += 1
        c_list.append(c)
        self.w_list = w_list
        self.w = w_list[-1]
        self.c_list = c_list

    def train(self, data, num_passes):
        w_list = []
        c_list = []
        c = 0
        w = np.zeros((data.shape[1]-1,))
        for p in range(num_passes):
            for i, row in data.iterrows():
                X = row[:data.shape[1]-1]
                Y = row[data.shape[1]-1]
                if Y*(np.dot(w,np.transpose(X))) <= 0:
                    # Adjust decision boundary, otherwise keep it
                    w = np.add(w,Y*X)
                    w_list.append(w)
                    c_list.append(c)
                    c = 1
                else:
                    c += 1
        c_list.append(c)
        self.w_list = w_list
        self.w = w_list[-1]
        self.c_list = c_list
```

Question 1

In [5]:

```
clf1 = perceptron()
clf1.train(training_data_q1, 2)
y_pred_normal = clf1.predict(training_data_q1, 'normal')
y_pred_avg = clf1.predict(training_data_q1, 'averaged')
y_pred_voted = clf1.predict(training_data_q1, 'voted')

y_pred_normal_test = clf1.predict(testing_data_q1, 'normal')
y_pred_avg_test = clf1.predict(testing_data_q1, 'averaged')
y_pred_voted_test = clf1.predict(testing_data_q1, 'voted')
```

In [6]:

```
y_true = training_data_q1[label_index]
y_true_test = testing_data_q1[label_index]
print("TRAINING ERROR RATES FOR TWO PASSES")
print("Error Rate Normal : ",calc_error(y_pred_normal, y_true))
print("Error Rate Voted: ",calc_error(y_pred_voted, y_true))
print("Error Rate Average: ",calc_error(y_pred_avg, y_true))

print("TESTING ERROR RATES FOR TWO PASSES")
print("Error Rate Normal : ",calc_error(y_pred_normal_test, y_true_test
))
print("Error Rate Voted: ",calc_error(y_pred_voted_test, y_true_test))
print("Error Rate Average: ",calc_error(y_pred_avg_test, y_true_test))
```

```
TRAINING ERROR RATES FOR TWO PASSES
Error Rate Normal :  0.03669724770642202
Error Rate Voted:  0.03761467889908257
Error Rate Average:  0.05412844036697248
TESTING ERROR RATES FOR TWO PASSES
Error Rate Normal :  0.0610079575596817
Error Rate Voted:  0.05305039787798409
Error Rate Average:  0.07957559681697612
```

In [7]:

```
clf2 = perceptron()
clf2.train(training_data_q1, 3)
y_pred_normal = clf2.predict(training_data_q1, 'normal')
y_pred_avg = clf2.predict(training_data_q1, 'averaged')
y_pred_voted = clf2.predict(training_data_q1, 'voted')

y_pred_normal_test = clf2.predict(testing_data_q1, 'normal')
y_pred_avg_test = clf2.predict(testing_data_q1, 'averaged')
y_pred_voted_test = clf2.predict(testing_data_q1, 'voted')
```

In [8]:

```
y_true = training_data_q1[label_index]
print("ERROR RATES FOR THREE PASSES")
print("Error Rate Normal: ",calc_error(y_pred_normal, y_true))
print("Error Rate Voted: ",calc_error(y_pred_voted, y_true))
print("Error Rate Average: ",calc_error(y_pred_avg, y_true))

print("TESTING ERROR RATES FOR THREE PASSES")
print("Error Rate Normal : ",calc_error(y_pred_normal_test, y_true_test
))
print("Error Rate Voted: ",calc_error(y_pred_voted_test, y_true_test))
print("Error Rate Average: ",calc_error(y_pred_avg_test, y_true_test))
```

```
ERROR RATES FOR THREE PASSES
Error Rate Normal:  0.01926605504587156
Error Rate Voted:  0.030275229357798167
Error Rate Average:  0.03669724770642202
TESTING ERROR RATES FOR THREE PASSES
Error Rate Normal :  0.04509283819628647
Error Rate Voted:  0.04774535809018567
Error Rate Average:  0.0610079575596817
```

In [9]:

```
clf3 = perceptron()
clf3.train(training_data_q1, 4)
y_pred_normal = clf3.predict(training_data_q1, 'normal')
y_pred_avg = clf3.predict(training_data_q1, 'averaged')
y_pred_voted = clf3.predict(training_data_q1, 'voted')

y_pred_normal_test = clf3.predict(testing_data_q1, 'normal')
y_pred_avg_test = clf3.predict(testing_data_q1, 'averaged')
y_pred_voted_test = clf3.predict(testing_data_q1, 'voted')
```

In [10]:

```
y_true = training_data_q1[label_index]
print("ERROR RATES FOR FOUR PASSES")
print("Error Rate Normal: ",calc_error(y_pred_normal, y_true))
print("Error Rate Voted: ",calc_error(y_pred_voted, y_true))
print("Error Rate Average: ",calc_error(y_pred_avg, y_true))

print("TESTING ERROR RATES FOR FOUR PASSES")
print("Error Rate Normal : ",calc_error(y_pred_normal_test, y_true_test
))
print("Error Rate Voted: ",calc_error(y_pred_voted_test, y_true_test))
print("Error Rate Average: ",calc_error(y_pred_avg_test, y_true_test))
```

```
ERROR RATES FOR FOUR PASSES
Error Rate Normal:  0.016513761467889 91
Error Rate Voted:  0.023853211009174313
Error Rate Average:  0.03211009174311927
TESTING ERROR RATES FOR FOUR PASSES
Error Rate Normal :  0.04509283819628647
Error Rate Voted:  0.04509283819628647
Error Rate Average:  0.05305039787798409
```

Question 2

In [11]:

```
w_avg = (sum(c*w for c, w in zip(clf2.c_list, clf2.w_list)))
sorted_w = np.argsort(w_avg)
print("Top (Strong Positive Predictors): \n", sorted_w[-3:])
print("Bottom (Strong Negative Predictors): \n", sorted_w[:3])
```

```
Top (Strong Positive Predictors):
 816    393
817    469
818     78
dtype: int64
Bottom (Strong Negative Predictors):
 0    438
1    466
2    203
dtype: int64
```

In [12]:

```
print(words[393].replace('\n',''),words[469].replace('\n',''),words[78].
replace('\n',''))
print(words[438].replace('\n',''),words[466].replace('\n',''),words[203]
.replace('\n',''))
```

```
game  team  he
file  program  line
```

Question 3:

In [13]:

```
class one_v_all:
    def __init__(self):
        self.classifiers = []

    def train(self, num_classes, training_data):
        classifiers = []
        for i in range(num_classes):
            clf = perceptron()
            clf.train_multiclass(training_data, i+1, 1)
            classifiers.append(clf)
        self.classifiers = classifiers

    def predictSample(self, test_sample):
        predictions = []
        for clf in self.classifiers:
            prediction = clf.predict(test_sample,'normal')
            predictions.append(prediction[0])
        if predictions.count(1) != 1:
            # Return "Don't Know"
            return 0
        else:
            # Predict whatever i classifier correctly predicted for
            return (predictions.index(1) + 1)

    def predict(self, test_data):
        predictions = []
        for i in range(test_data.shape[0]):
            prediction = self.predictSample(test_data.iloc[[i]])
            predictions.append(prediction)
        return predictions
```

In [14]:

```
def confusion_matrix(y_pred, y_true):
    # dim = number of labels in prediction task
    rows = len(set(y_pred))
    cols = len(set(y_true))
    C = np.zeros((rows, cols))
    # Find number of each class label in test data
    num_label = [0]*cols
    for i in range(len(num_label)):
        label = y_true.count(i+1)
        num_label[i] = label
    for x, y in zip(y_pred, y_true):
        C[x,y-1] += 1
    for i in range(rows):
        for j in range(cols):
            C[i,j] /= num_label[j]
    return C
```

In [15]:

```
clf = one_v_all()
clf.train(6, training_data)
y_pred = clf.predict(training_data)
y_true = training_data[label_index]
```

In [16]:

```
calc_error(y_pred, y_true)
```

Out[16]:

```
0.32133333333333336
```

In [17]:

```
C = confusion_matrix(y_pred, list(y_true))
print(C)
```

```
[[0.16390424 0.18829982 0.44933078 0.24758221 0.14671815 0.34943182]
 [0.79742173 0.00365631 0.00956023 0.00386847 0.0019305  0.00284091]
 [0.01104972 0.76416819 0.03441683 0.01547389 0.01351351 0.01988636]
 [0.         0.00182815 0.43403442 0.00580271 0.00579151 0.00852273]
 [0.01473297 0.01462523 0.01529637 0.71760155 0.00772201 0.00852273]
 [0.0092081  0.02010969 0.03441683 0.00773694 0.78957529 0.10511364]
 [0.00368324 0.00731261 0.02294455 0.00193424 0.03474903 0.50568182]]
```