Senator_Voting

October 15, 2019

0.1 PCA and Senate Voting Data

0.1.1 Places where you have to write code are marked with #TODO

In this problem we are given, X the $n \times m$ data matrix with entries in $\{-1, 0, 1\}$, where each row corresponds to a Senator, and each column to a bill.

[1]:
```
# Import the necessary packages for data manipulation, computation and PCA
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
%matplotlib inline

np.random.seed(7)
```

[2]:
```
senator_df =  pd.read_csv('../data/senator_data_matrix.csv')
affiliation_file = open("../data/politician_labels.txt", "r")
affiliations = [line.split('\n')[0].split(' ')[1] for line in affiliation_file.
 ↪readlines()]
X = np.array(senator_df.values[:, 3:].T, dtype='float64') #transpose to get␣
 ↪senators as rows
print("X.shape: ", X.shape)
n = X.shape[0] #Number of senators
m = X.shape[1] #Number of bills
```

```
X.shape:  (100, 542)
```

We see that the number of rows n, is the number of senators and is equal to 100. The number of columns, m is the number of bills and is equal to 542.

[3]:
```
typical_row = X[0,:]
print(typical_row.shape)
print(typical_row)
```

```
(542,)
[ 1.  1.  1. -1. -1.  1.  1.  1.  1. -1.  1. -1. -1.  1.  1. -1.  1.  1.
```

```
 1.  1.  1. -1.  1.  1.  1. -1.  1. -1.  1.  1.  1.  1.  1. -1.  1. -1.
-1. -1. -1.  1.  1. -1. -1. -1. -1.  1.  1.  1. -1.  1.  1. -1.  1.  1.
-1.  1.  1.  1.  1. -1.  1. -1. -1. -1.  1.  1.  1.  1.  1.  1.  1.  1.
 1.  1. -1.  0. -1.  1.  1.  1. -1. -1.  1.  1. -1. -1.  1.  1.  1. -1.
 1. -1.  1. -1.  1.  1. -1. -1. -1.  1.  1.  1. -1. -1. -1. -1. -1. -1.
 1. -1.  1.  1. -1. -1. -1.  1. -1.  1. -1.  1.  0.  0.  1.  1. -1.  1.
 1. -1.  1.  1. -1.  1. -1. -1.  1.  1.  1.  1.  0. -1. -1.  1.  1. -1.
 1.  1.  1.  1.  1.  0.  1.  0.  1.  1.  1.  1.  1.  1.  1. -1.  1.  1.
-1.  1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1. -1.  1.
 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1. -1. -1.  1.  1.  1.
 1.  1.  1.  1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1.  1.  1.  1. -1.
 1.  1.  0.  1.  0. -1.  1.  1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1.
 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
 1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1. -1.  1.  1.  1. -1.
 1.  1.  1.  1.  1.  1. -1. -1. -1.  1.  1. -1.  1. -1. -1.  1.  1.  1.
-1.  1.  1.  1. -1.  1. -1.  1. -1. -1.  1. -1. -1.  1.  1.  1. -1.  1.
 1.  1.  1.  1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1. -1. -1.
 1. -1.  1. -1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
 1.  1.  1.  1.  1.  1.  1.  1.  1. -1.  1.  1.  1.  1.  1. -1.  1. -1.
 1.  1.  1.  1. -1. -1.  1.  1. -1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1. -1.  1.  1.  1.
-1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
 1.  1. -1. -1. -1.  1.  1.  1.  1. -1. -1.  1.  1. -1.  1.  1.  1.  1.
 1.  1.]
```

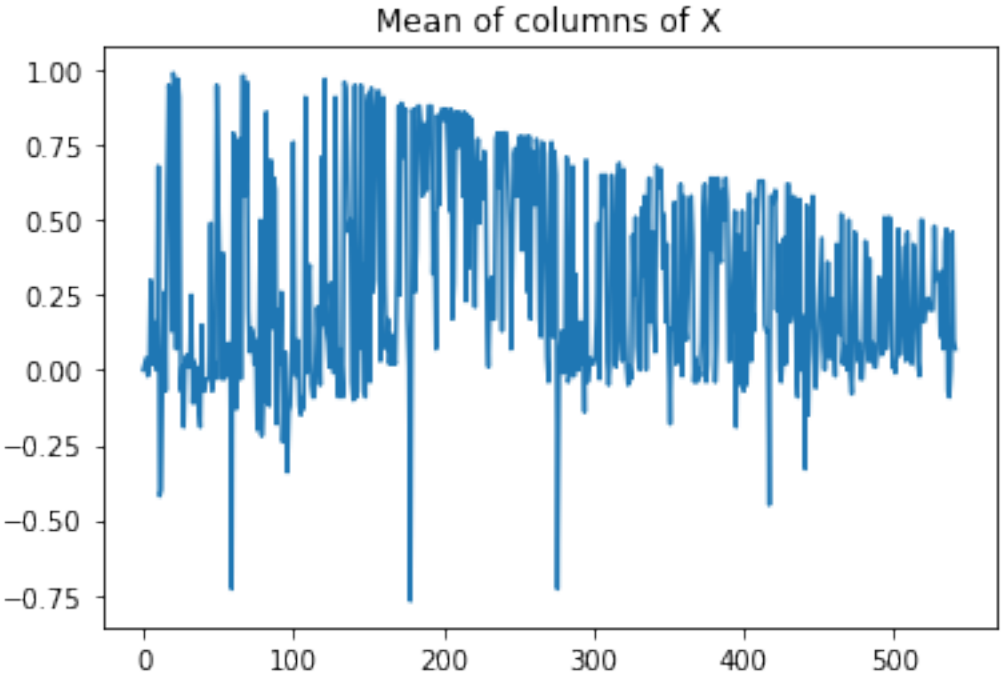
A typical row of X consists of entries -1 (senator voted against), 1(senator voted for) and 0(senator abstained) for each bill.

[4]:
```
typical_column = X[:,0]
print(typical_column.shape)
print(typical_column)
```

```
(100,)
[ 1.  1.  1.  1.  1.  1.  1. -1.  1. -1.  1. -1.  1. -1. -1. -1.  1.  1.
 -1.  1.  1. -1.  1. -1.  1.  1.  1. -1. -1.  1.  1.  1. -1.  1.  1.  1.
 -1. -1. -1. -1.  1. -1. -1.  1.  1. -1. -1. -1. -1. -1.  1.  1. -1. -1.
  1.  1. -1. -1. -1. -1. -1.  1.  1.  1.  1.  1. -1. -1. -1.  1. -1. -1.
  1. -1. -1.  1.  1.  1. -1. -1. -1.  1.  1. -1.  1. -1.  1.  1.  1. -1.
 -1. -1. -1. -1.  1.  1.  1. -1. -1. -1.]
```

A typical row of X consists of entries in {-1, 0, 1} based on how each senator voted for that particular bill.

[5]:
```
X_mean = np.mean(X, axis = 0)
plt.plot(X_mean)
plt.title('Mean of columns of X')
plt.show()
```



We see that the mean of the columns is not zero so we center the data by subtracting the mean

[6]:
```
X_original = X.copy()
X = X - np.mean(X, axis = 0)
```

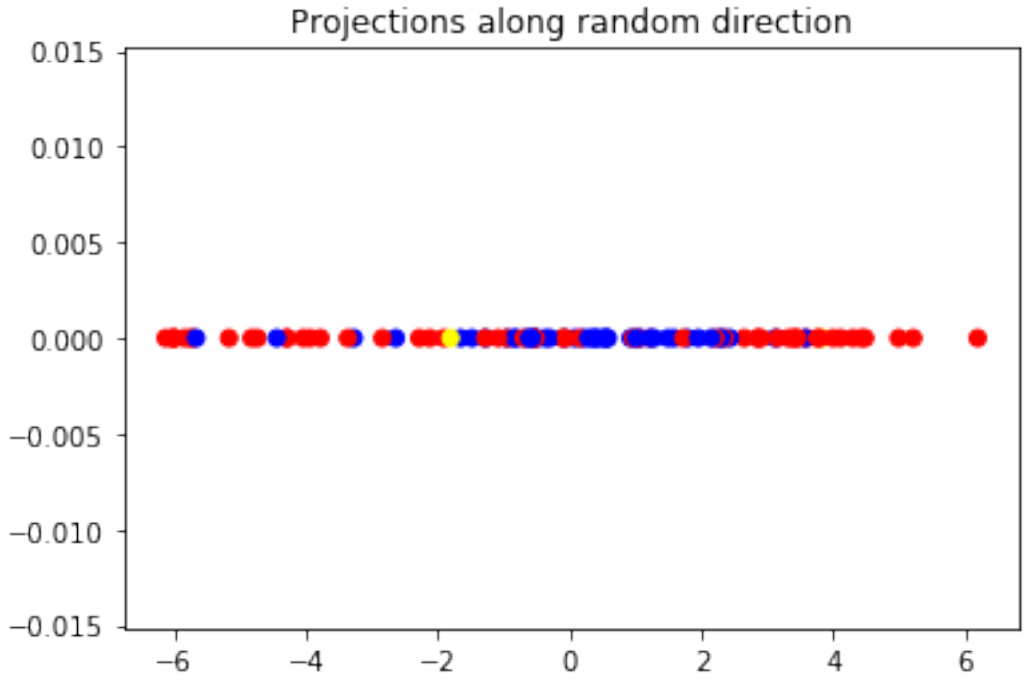
0.1.2 Part a) Finding a unit-norm m-vector a to maximize variance

This is a function to calculate the scores, $f(X, a)$.

[7]:
```
def f(X, a):
    return np.matmul(X,a)
```

Before we calculate the a that maximizes variance, let us observe how the scalar projections on a random direction a look like.

[8]:
```
a_rand = np.random.rand(542,1) #generate a random direction
a_rand = a_rand/np.linalg.norm(a_rand) #we normalize the vector
scores_rand = f(X, a_rand) #recall definition of f above
# Now we visualize the scores along a_rand
plt.scatter(scores_rand, np.zeros_like(scores_rand), c=affiliations)
plt.title('Projections along random direction')
plt.show()

print("Variance along random direction: ", np.round(scores_rand.var(),3))
```



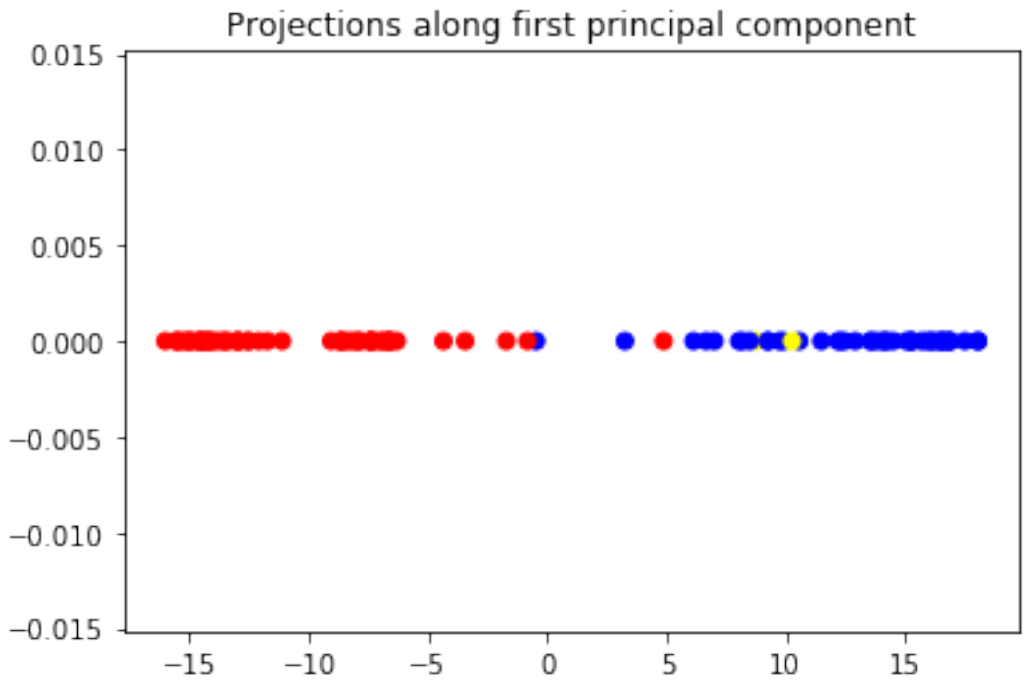

```
Variance along random direction:  9.267
```

Note here that projecting along the random vector a_rand does not explain much variance at all! It is clear that this direction does not give us any information about the senators' affiliations.

Next let us find direction a_1 that maximizes variance. This will be the first principal component of X.

[9]:
```
#TODO: write code to get a_1, the first principal component of X(Note that␣
↪shape of a_1 must be (542, 1))
pca = PCA(n_components=100)
pca.fit(X)
```

```
# pdb.set_trace()
# a_1 = a_rand #TODO replace this line with code to get a_1 that maximizes␣
 ↪variance
a_1 = pca.components_[0].reshape((542,1))
#Hint: the PCA package imported from sklearn.decomposition will be useful here.␣
 ↪Look up the function
#pca.fit() from its documentation

if(np.shape(a_1)[1]!=1): print("ERROR: Shape of a_1 must be (542, 1)")
#Next we compute scores along first principal component
scores_a_1 = f(X, a_1) #recall definition of f above
plt.scatter(scores_a_1, np.zeros_like(scores_a_1), c=affiliations)
plt.title('Projections along first principal component')
plt.show()

print("Variance along first principal component: ", np.round(scores_a_1.
 ↪var(),3))
```




```
Variance along first principal component:  149.749
```

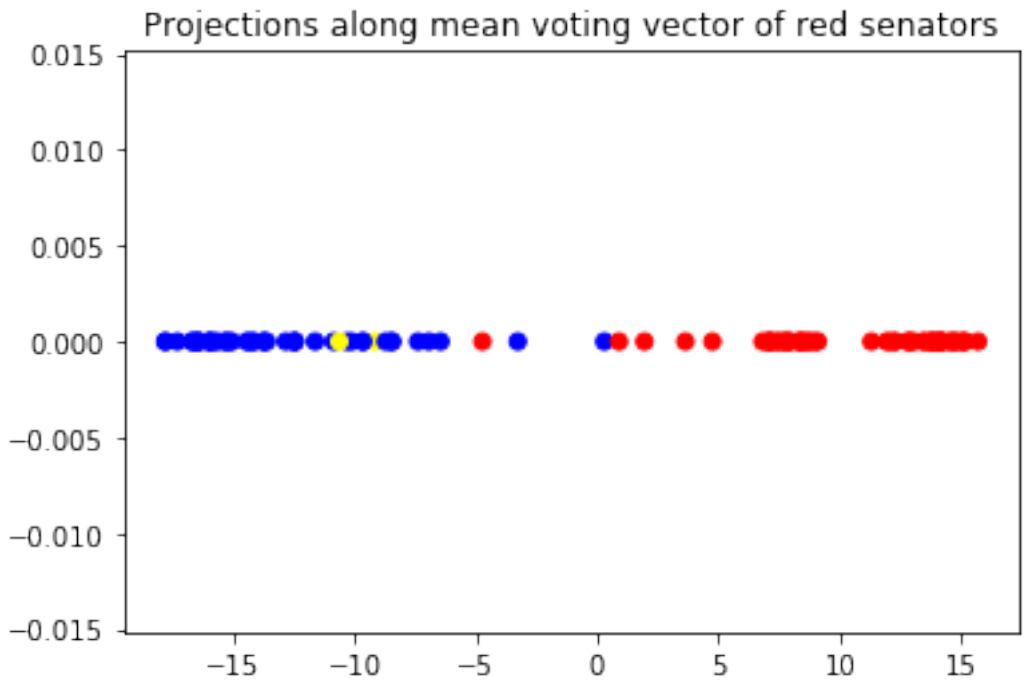
We can see that majority of the blue is close to one side of the axis and red is close to the other side. This shows that the first principal component direction explains the vote spread tends to align with party affiliation.

0.1.3 Part b) Comparison to party averages

Building on the observation that senators vote in line with their party average let us compute variance along the following two directions: a_mean_red: Unit vector along mean of rows corresponding to RED senators
a_mean_blue: Unit vector along mean of rows corresponding to BLUE senators ### #TODO Find the expression for mu_red (with shape (542,1)) in the first line, corresponding to the mean of rows of X corresponding to Red senators as given by affiliations

[10]:
```
import pdb
mu_red = np.ones((542,1))
# red_indices = np.where(affiliations=="Red")
red_indices = [i for i in range(0,len(affiliations)) if affiliations[i] ==␣
 ↪"Red"]
mu_red = np.mean(X[red_indices,:],axis=0)
mu_red.reshape((542,1))


#Hint: Print out affiliations and check what its entries are:
# print(len(affiliations))
# print(affiliations)

print(mu_red.shape)
a_mean_red = mu_red/np.linalg.norm(mu_red) # normalize the vector
scores_mean_red = f( X, a_mean_red)
plt.scatter(scores_mean_red, np.zeros_like(scores_mean_red), c=affiliations)
plt.title('Projections along mean voting vector of red senators')
plt.show()
# pdb.set_trace()

print("Variance along mean voting vector of red senators: ", np.
 ↪round(scores_mean_red.var(),3))

#Let us check angle between this and the first prinicpal component
dot_product_red_a1 = float(np.dot(a_mean_red.T, a_1))
angle_red_a1 = np.arccos(dot_product_red_a1)*180/np.pi

print("Dot product of a_mean_red and a_1:", np.round(dot_product_red_a1,3))
print("Angle between a_mean_red and a_1 in degrees:", np.round(angle_red_a1,3))
```

```
(542,)
```



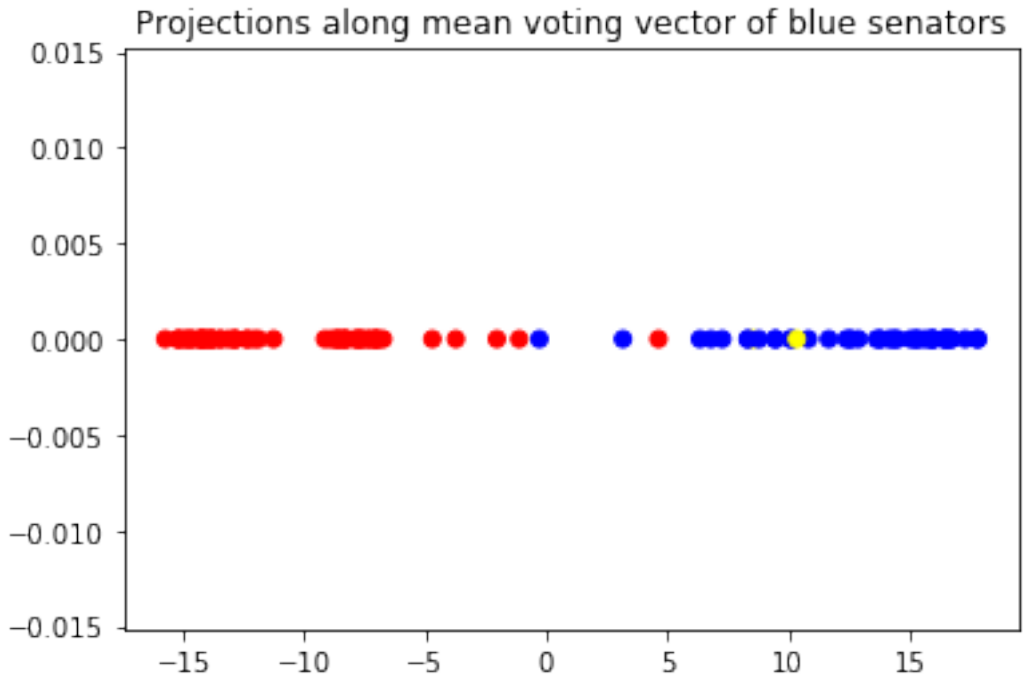

```
Variance along mean voting vector of red senators:  148.807
Dot product of a_mean_red and a_1: -0.997
Angle between a_mean_red and a_1 in degrees: 175.229
```

0.1.4 #TODO Find the expression for mu_blue (with shape (542,1)) in the first line, corresponding to the mean of rows of X corresponding to Blue senators as given by affiliations

[11]:
```
mu_blue = np.ones((542,1))

blue_indices = [i for i in range(0,len(affiliations)) if affiliations[i] ==␣
↪"Blue"]
mu_blue = np.mean(X[blue_indices,:],axis=0)
mu_blue.reshape((542,1))
#Hint: Print out affiliations and check what its entries are:
# print(len(affiliations))
# print(affiliations)

print(mu_blue.shape)

a_mean_blue = mu_blue/np.linalg.norm(mu_blue) # normalize the vector
scores_mean_blue = f( X, a_mean_blue)
plt.scatter(scores_mean_blue, np.zeros_like(scores_mean_blue), c=affiliations)
plt.title('Projections along mean voting vector of blue senators')
```

```
plt.show()

print("Variance along mean voting vector of blue senators: ", np.
 ↪round(scores_mean_blue.var(),3))

#Let us check angle between this and the first prinicpal component
dot_product_blue_a1 = float(np.dot(a_mean_blue.T, a_1))
angle_blue_a1 = np.arccos(dot_product_blue_a1)*180/np.pi

print("Dot product of a_mean_blue and a_1:", np.round(dot_product_blue_a1,3))
print("Angle between a_mean_blue and a_1 in degrees:", np.
 ↪round(angle_blue_a1,3))
```

```
(542,)
```




```
Variance along mean voting vector of blue senators:  148.909
Dot product of a_mean_blue and a_1: 0.997
Angle between a_mean_blue and a_1 in degrees: 4.452
```

[12]:
```
#Finally let us compute angle between a_mean_red and a_mean_blue:
dot_product_blue_red = float(np.dot(a_mean_blue.T, a_mean_red))
angle_blue_red = np.arccos(dot_product_blue_red)*180/np.pi

print("Dot product of a_mean_blue and mean_red:", np.
 ↪round(dot_product_blue_red,3))
```

```
print("Angle between a_mean_blue and mean_red in degrees:", np.
 ↪round(angle_blue_red,3))
```

```
Dot product of a_mean_blue and mean_red: -0.999
Angle between a_mean_blue and mean_red in degrees: 177.759
```

0.1.5 #TODO Fill in code to obtain mu_red, and mu_blue in the cells above. Comment on your observations about how the party averages (a_mean_red and a_mean_blue) are related to the first principal component (a_1).

Comments
The a_mean_red and a_mean_blue are exactly the first principal components. Meaning that red and blue senators tend to vote with their party.

0.1.6 Part c) Computing total variance. Fill in code in cell below to obtain total variance along first two principal components. (Refer to the latex file for more details on the question).

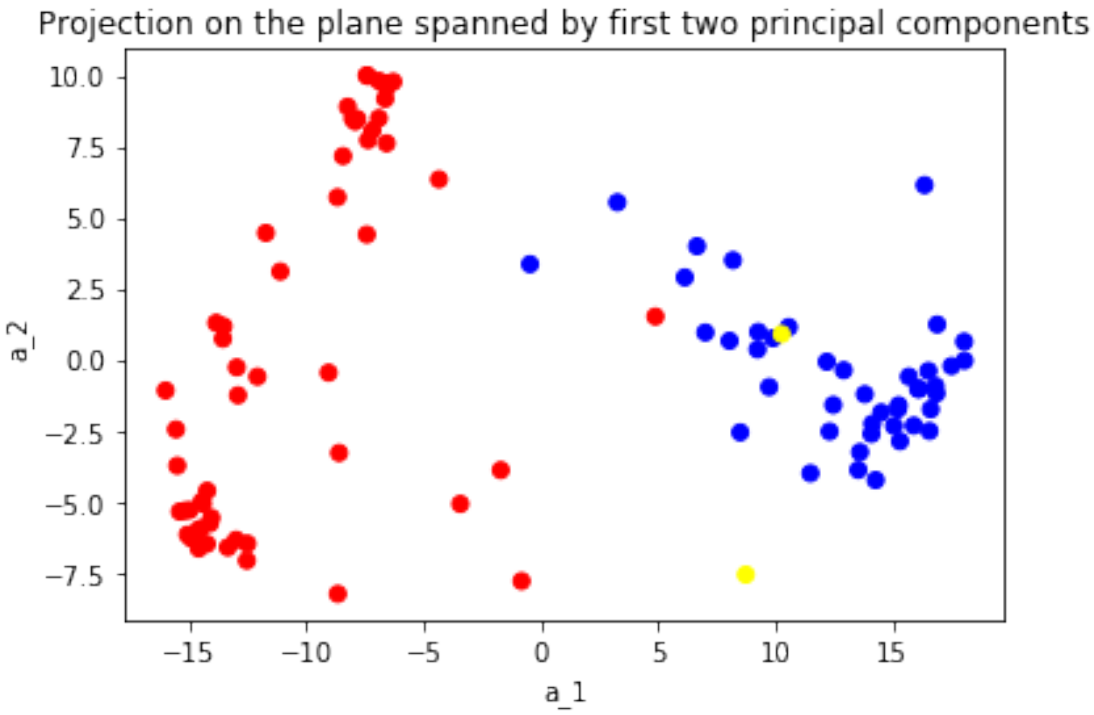
[13]:
```
X_bar = np.matmul(X.T, X)/n

total_variance = sum(np.sqrt(pca.singular_values_[i]) for i in [0,1]) #Replace␣
 ↪with correct answer, the sum of largest two eigenvalues lambda_1, lambda_2␣
 ↪of X_bar

print("Total variance explained by first two principal components: ", np.
 ↪round(total_variance,3))
```

```
Total variance explained by first two principal components:  18.163
```

Next we find the projection onto the plane spanned by the first two principal components

[14]:
```
pca = PCA(n_components=2)
projected = pca.fit_transform(X)

print(projected.shape)
plt.scatter(projected[:, 0], projected[:, 1], c=affiliations)
plt.xlabel('a_1')
plt.ylabel('a_2')
plt.title('Projection on the plane spanned by first two principal components')
plt.show()
```

```
(100, 2)
```



0.2 Part d) Finding bills that are the most/least contentious

0.2.1 Approach 1: Finding variance of columns of X. Note that the variance of column j can be viewed as the variance of scores along the direction e_j, where e_j is a basis vector with one in the jth entry and zero elsewhere.

[15]:
```
list_variances = X.var(0) # projects the standard basis in R^n for all bills;␣
 ↪returns variances of each column
bills = senator_df['bill_type bill_name bill_ID'].values

# sorted_idx_variances = np.arange(list_variances.shape[0]) #TODO remove this␣
 ↪line and replace it with code to
#compute sorted_idx_variances: a np.array of shape (542,) containing integer␣
 ↪entries that are indices
# corresponding to decreasing order of variance of scores in list_variances.␣
 ↪Hint: Use np.argsort()
#Eg. If list_variances = [1,3,2,4], then sorted_idx_variances should be np.
 ↪array([3,1,2,0])
# sorted_idx_variances = pca.singular_values_
sorted_idx_variances = np.argsort(list_variances[::-1])

# pdb.set_trace()
```

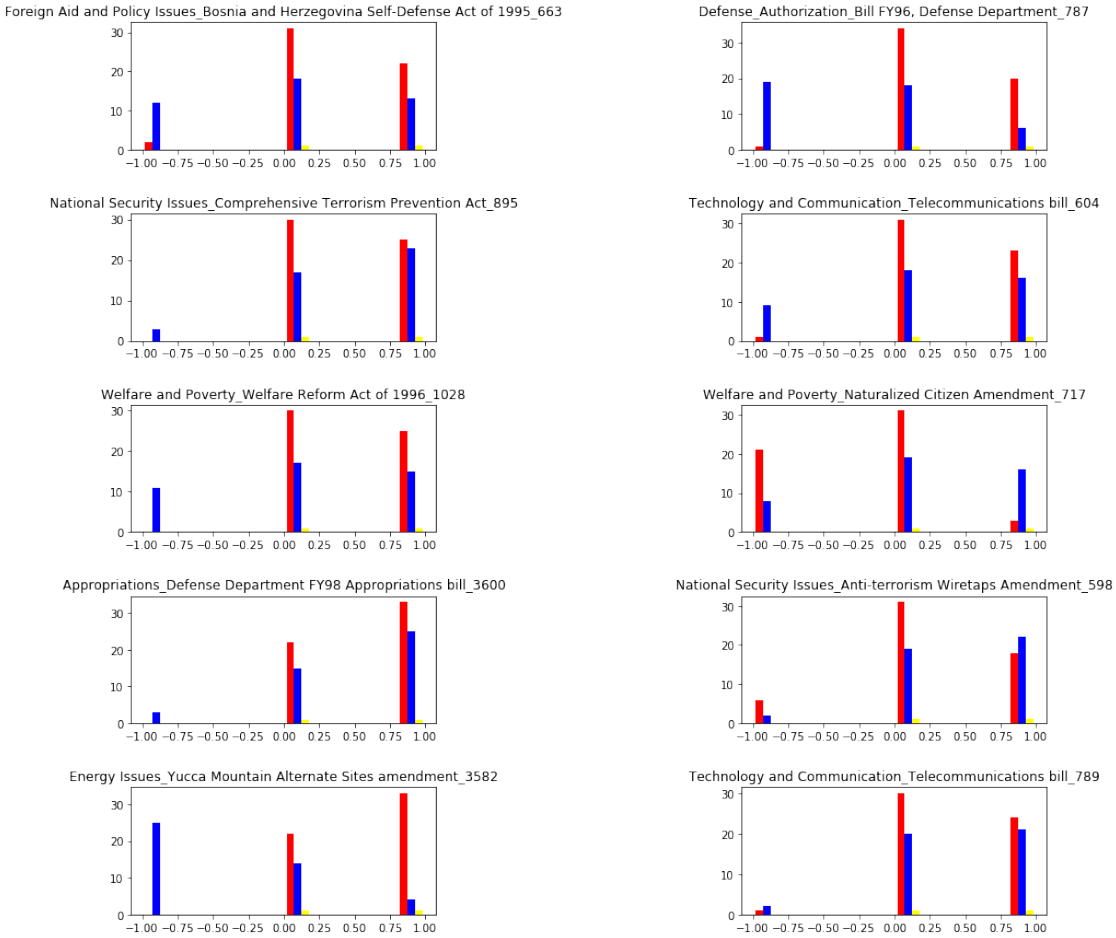
```
print(sorted_idx_variances.shape)
```

```
(542,)
```

0.2.2 #TODO: Part d i) Fill in code to compute sorted_idx_variances in the cell above

[16]:
```
# Retrive the bills with the top 5 variances and the lowest 5 variances
top_5 = [bills[sorted_idx_variances[i]] for i in range(5)]
# print(top_10)
bot_5 = [bills[sorted_idx_variances[-1-i]] for i in range(5)]

# print(bot_10)
#We look at voting pattern for bills with most and least variance using␣
 ↪original non-centered X matrix
fig, axes = plt.subplots(5,2, figsize=(15,15)) # 1 plot to make things easier␣
 ↪to see
for i in range(5):
    idx = sorted_idx_variances[i]

    X_red_c = X_original[np.array(affiliations) == 'Red',idx]
    X_blue_c = X_original[np.array(affiliations) == 'Blue',idx]
    X_yellow_c = X_original[np.array(affiliations) == 'Yellow',idx]

    axes[i,0].hist([X_red_c, X_blue_c, X_yellow_c], color = ['red', 'blue',␣
 ↪'yellow'])
    axes[i,0].set_title(bills[idx])


for i in range(1,6):
    idx2 = sorted_idx_variances[-i]
    X_red_c2 = X_original[np.array(affiliations) == 'Red',idx2]
    X_blue_c2 = X_original[np.array(affiliations) == 'Blue',idx2]
    X_yellow_c2 = X_original[np.array(affiliations) == 'Yellow',idx2]

    axes[i-1,1].hist([X_red_c2, X_blue_c2, X_yellow_c2], color = ['red',␣
 ↪'blue', 'yellow'])
    axes[i-1,1].set_title(bills[idx2])

plt.subplots_adjust(hspace=0.5, wspace = 1)
fig.suptitle('Most Variance -- Least Variance', fontsize=16)
plt.show()
```

Most Variance -- Least Variance



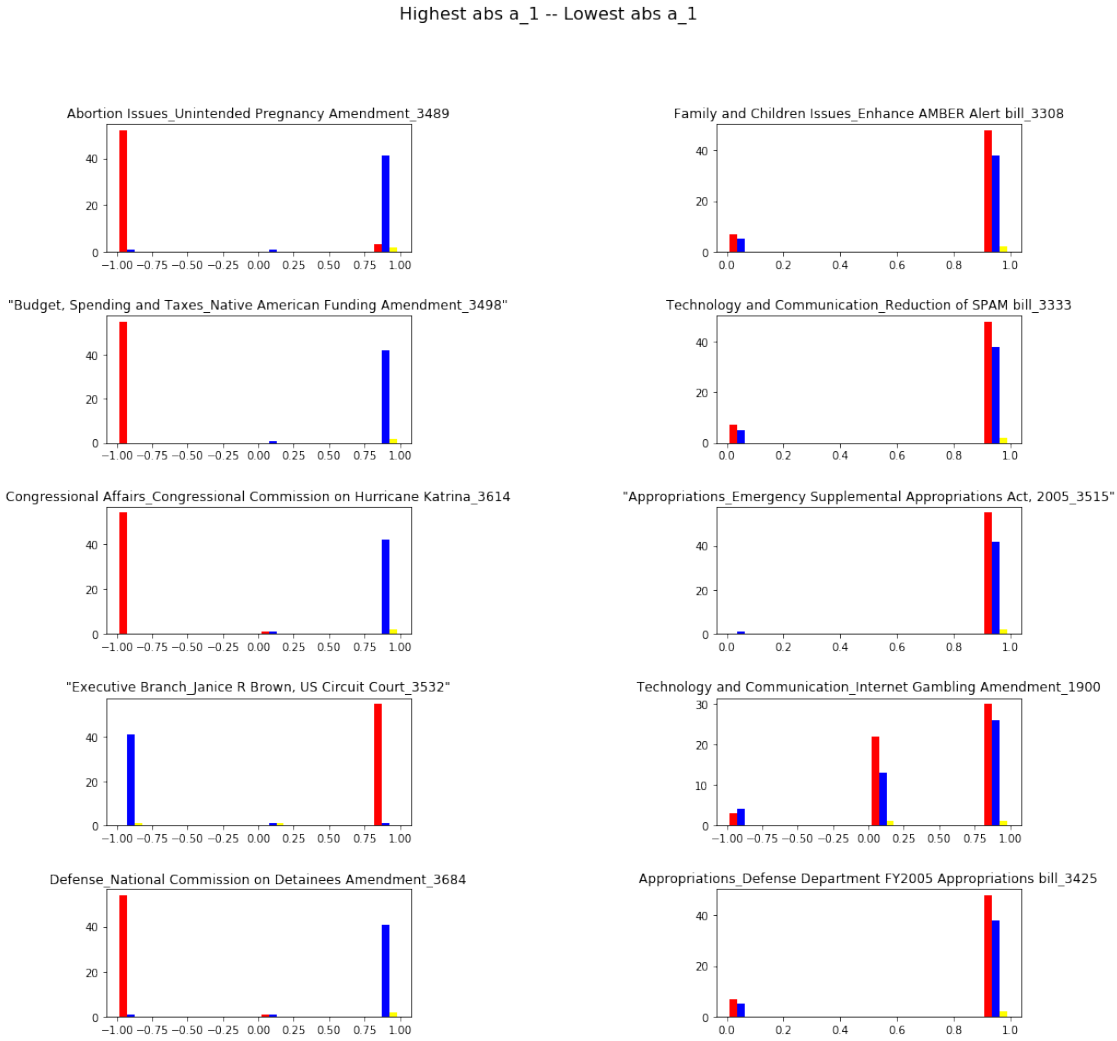
0.2.3 #TODO Part d ii) Comment on how the voting looks like for bills with most variance and bills with least variance

There is either a greater spread of votes between and within parties, for the bills with most variance. But actually I dont see a difference.

0.2.4 Approach 2: We find the projection of the basis vector corresponding to each bill on to the first principal components and choose those bills with highest absolute value of projections. Note that this is equivalent to choosing bills based on highest absolute values of a_1.

[17]:
```
# Recall that a_1_scores holds the projection onto the first principal component
a_1_flat = np.ndarray.flatten(a_1) # first, flatten the a_1 of len 542
abs_a_1 = np.abs(a_1_flat)

sorted_idxes = np.argsort(-abs_a_1) #in decreasing order
print(sorted_idxes.shape)

top_5_a1 = [bills[sorted_idxes[i]] for i in range(5)]
bot_5_a1 = [bills[sorted_idxes[-1-i]] for i in range(5)]


fig, axes = plt.subplots(5,2, figsize=(15,15)) # 1 plot to make things easier␣
 ↪to see

for i in range(5):
    idx = sorted_idxes[i]

    X_red_c = X_original[np.array(affiliations) == 'Red',idx]
    X_blue_c = X_original[np.array(affiliations) == 'Blue',idx]
    X_yellow_c = X_original[np.array(affiliations) == 'Yellow',idx]

    axes[i,0].hist([X_red_c, X_blue_c, X_yellow_c], color = ['red', 'blue',␣
 ↪'yellow'])
    axes[i,0].set_title(bills[idx])


for i in range(1,6):
    idx2 = sorted_idxes[-i]

    X_red_c2 = X_original[np.array(affiliations) == 'Red',idx2]
    X_blue_c2 = X_original[np.array(affiliations) == 'Blue',idx2]
    X_yellow_c2 = X_original[np.array(affiliations) == 'Yellow',idx2]

    axes[i-1,1].hist([X_red_c2, X_blue_c2, X_yellow_c2], color = ['red',␣
 ↪'blue', 'yellow'])
    axes[i-1,1].set_title(bills[idx2])

plt.subplots_adjust(hspace=0.5, wspace = 1)
fig.suptitle('Highest abs a_1 -- Lowest abs a_1', fontsize=16)
plt.show()
```

```
(542,)
```



0.2.5 #TODO Part d iii) Comment on how the voting looks like for bills with highest and lowest absolute values of a_1.

The difference is in whether differing parties voted similarly to eachother or not.

Next let us compare the bills found by the two approaches

[18]:
```
# The bills that are the same in both the top and bottom 10 using these␣
 ↪different methods:
print('Number of common bills in top:', len(np.intersect1d(top_5 ,top_5_a1)))
print('Number of common bills in bottom :', len(np.intersect1d(bot_5␣
 ↪,bot_5_a1)))
```

```
Number of common bills in top: 0
Number of common bills in bottom : 0
```

0.2.6 #TODO Part d iv) Are the bills in the two approaches the same? What do you think is the reason for the difference?

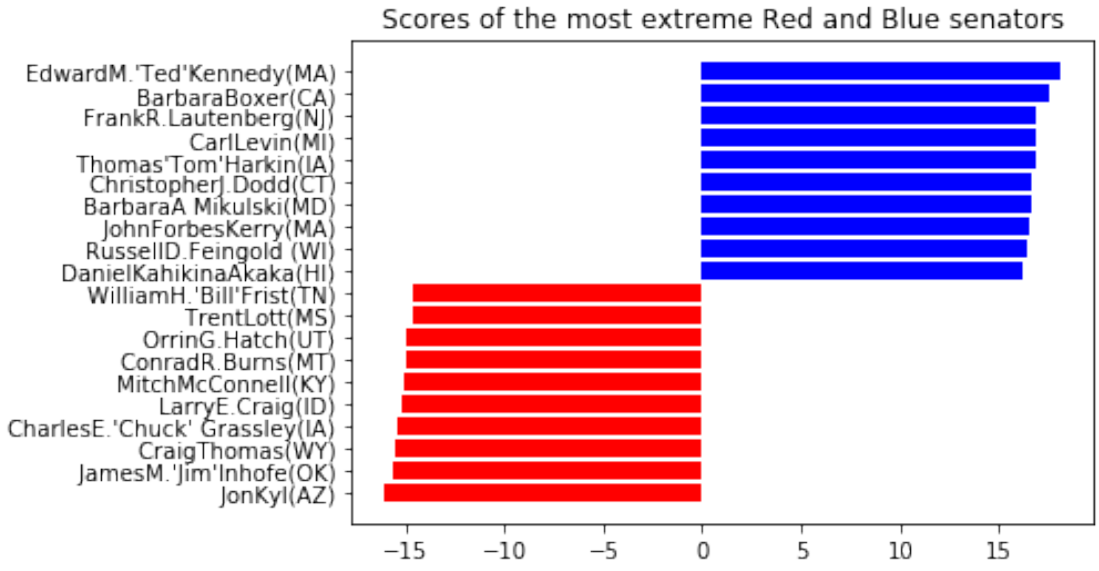
The bills in the two approaches look different. I think thats because the first, and all following principal components are very similar.

0.2.7 Part e) Finally, we will look at the scores for senators along the first principal direction and make the following classifications for senators:

a) Senators with the top 10 most positive scores and top 10 most negative scores are classified as most "extreme".
b) Senators with the 20 scores closest to 0 are classified as least "extreme".

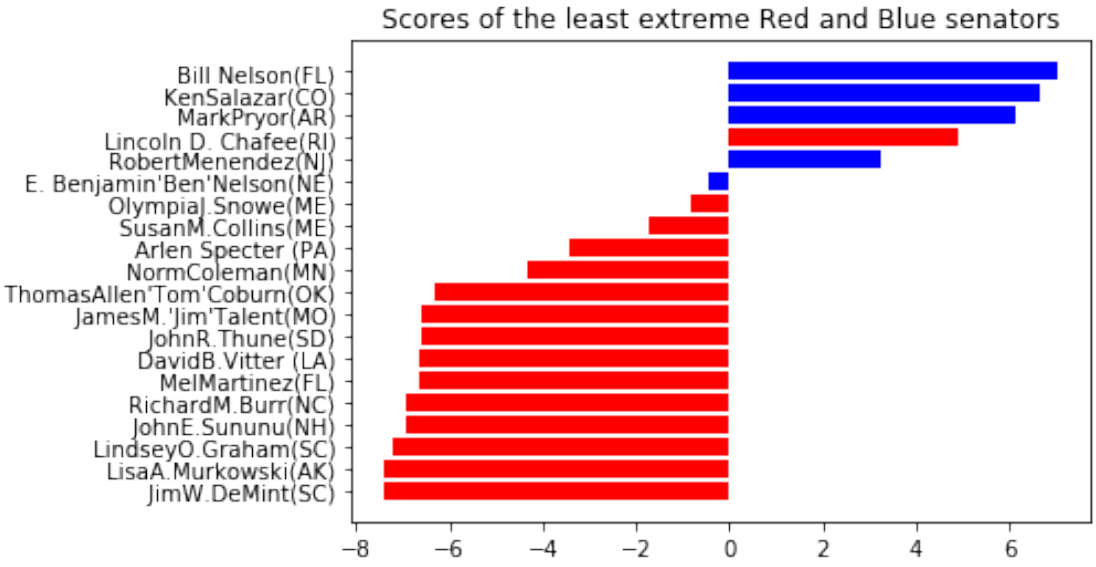
Most extreme senators

[19]:
```
senators = senator_df.columns.values[3:]

senator_scores = f(X,a_1)[:,0]
complete_sort_indices = np.argsort(senator_scores)

sort_indices = np.hstack([complete_sort_indices[:10], complete_sort_indices[-11:
 ↪-1]])
senators_sorted = senators[sort_indices]
senator_scores_sorted = senator_scores[sort_indices]
affiliations = np.array(affiliations)
affiliations_sorted = affiliations[sort_indices]

plt.barh(y = senators_sorted, width = senator_scores_sorted, color =␣
 ↪affiliations_sorted)
plt.title('Scores of the most extreme Red and Blue senators')
plt.show()
```



Least extreme senators

[20]:
```
senator_scores = f(X,a_1)[:,0]
# print(np.sort(np.abs(senator_scores)))
complete_sort_indices = np.argsort(np.abs(senator_scores))[:20]

senator_scores_le= senator_scores[complete_sort_indices]
senators_le = senators[complete_sort_indices]
affiliations = np.array(affiliations)
affiliations_le = affiliations[complete_sort_indices]
sort_indices = np.argsort(senator_scores_le)
senators_sorted = senators_le[sort_indices]
senator_scores_sorted = senator_scores_le[sort_indices]
affiliations_sorted = affiliations_le[sort_indices]

plt.barh(y = senators_sorted, width = senator_scores_sorted, color =␣
 ↪affiliations_sorted)
plt.title('Scores of the least extreme Red and Blue senators')
plt.show()
```



0.2.8 #TODO Comment on the sign of scores vs party affiliations.

Some party memebers tend to vote against the tendancy of the rest of their party.