Checkpoint 2_Random Forest

May 8, 2023

Data processing

In [1]:
```
import pandas as pd
from ast import literal_eval
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
path = '/Users/wangchenhui/UChicago/ML/Final_Project/dataset/'
```

In [3]:
```
# load downloaded data
df_convos = pd.read_csv(path+'/conversations.csv')
df_speakers = pd.read_csv(path+'/speakers.csv')
df_utts = pd.read_csv(path+'/utterances.csv')
df_cases = pd.read_json(path_or_buf='https://zissou.infosci.cornell.edu/convokit/datasets/supreme-corpu
df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) & (df_cases['win_side'].is
```

In [4]:
```
# combine text from all utterances in a conversation back into one string based on the conversation_id
utt_per_conv = df_utts.groupby('conversation_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
utt_per_conv['num_utterances'] = df_utts.groupby('conversation_id')['text'].count().reset_index()['tex

# add the combined text to the conversations dataframe, merge on conversation_id in utt_per_conv and i
df_convos_utt = df_convos.merge(utt_per_conv, left_on='id', right_on='conversation_id', how='left')
```

In [5]:
```
# combine text from all conversation in a cases into one string based on the meta.case_id
conv_per_case = df_convos_utt.groupby('meta.case_id')['text'].apply(lambda x: ' '.join(x)).reset_index
conv_per_case['num_conversations'] = df_convos_utt.groupby('meta.case_id')['text'].count().reset_index
conv_per_case['num_utterances'] = df_convos_utt.groupby('meta.case_id')['num_utterances'].sum().reset_

# add the combined text case dataframe, merge on meta.case_id and id
df_cases_convo = df_cases.merge(conv_per_case, left_on='id', right_on='meta.case_id', how='left')
```

In [6]:
```
df_cases_convo.dropna(subset=['text'], inplace=True)
```

In [7]:
```
# transform to pd.to_datetime
df_cases_convo.decided_date = pd.to_datetime(df_cases_convo.decided_date)
```

In [8]:
```
df_cases_convo.to_csv('df_cases_convo.csv', index=False)
```

Clean Data

In [9]:

```
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [10]:

```
# Cleaning the text
def preprocess_text(text):
    text = text.lower() # Lowercase the text
    text = re.sub('[^a-z]+', ' ', text)  # Remove special characters and numbers
    text = re.sub(r'\b\w{1,3}\b', '', text) # Remove words with length less than 3
    words = nltk.word_tokenize(text) # Tokenize the text
    stop_words = set(stopwords.words('english')) # Remove stopwords
    words = [word for word in words if word not in stop_words]
    #lemmatizer = WordNetLemmatizer() # Lemmatize the words comment because slow
    #words = [lemmatizer.lemmatize(word) for word in words]
    text = ' '.join(words) # Reconstruct the text

    return text
```

In [11]:

```
df = df_cases_convo.copy()
```

In [12]:

```
# preprocess text
df.loc[:, 'text_pre'] = df['text'].apply(preprocess_text)
```

In [13]:

```
# preprocess develop time
df.loc[:, 'start_date'] = df['transcripts'].apply(lambda x : re.findall(r'[A-Z][a-z]+ \d{2}, \d{4}', x
df.start_date = pd.to_datetime(df.start_date)
df.loc[:, 'develop_time'] = df.loc[:, 'decided_date'] - df.loc[:, 'start_date']
# df['develop_time'] = df['develop_time'].apply(lambda x : x.days)
```

In [14]:

```
# get party of the judges

def check_party_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
              'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    rep_ct = 0

    for judge in x:
        if judge in rep_judge:
            rep_ct += 1

    return rep_ct/len(x)
```

In [15]:
```
df['votes_side'][1]['j__john_g_roberts_jr']
```

Out[15]:
```
0.0
```

In [16]:
```
# get rep_judge yes

def check_rep_j_y_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
             'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    rep_y_ct = 0

    for judge in x:
        if judge in rep_judge:
            if x[judge] > 0:
                rep_y_ct += 1

    return rep_y_ct/len(x)
```

In [17]:
```
# get dem_judge yes

def check_dem_j_y_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
             'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    dem_y_ct = 0

    for judge in x:
        if judge in dem_judge:
            if x[judge] > 0:
                dem_y_ct += 1

    return dem_y_ct/len(x)
```

In [18]:
```
def check_party(x):
    if x > 2009:
        return 0
    else:
        return 1
```

In [19]:

```
# only numbers can apply to Random Forest Model
df_rf = pd.DataFrame()
df_rf.loc[:, 'text_len'] = df['text'].apply(lambda x : len(x))
df_rf.loc[:, 'text_pre_len'] = df['text_pre'].apply(lambda x : len(x))
df_rf.loc[:, 'num_utterances'] = df['num_utterances']
df_rf.loc[:, 'win_side'] = df['win_side']
df_rf.loc[:, 'develop_time'] = df['develop_time'].apply(lambda x : x.days)
df_rf.loc[:, 'rep_jpc'] = df['votes_side'].apply(check_party_pc)
df_rf.loc[:, 'rep_j_y_pc'] = df['votes_side'].apply(check_rep_j_y_pc)
df_rf.loc[:, 'dem_j_y_pc'] = df['votes_side'].apply(check_dem_j_y_pc)
df_rf.loc[:, 'party'] = df['year'].apply(check_party) # 1:  rep, 0: dem

df_rf
```

Out[19]:

	text_len	text_pre_len	num_utterances	win_side	develop_time	rep_jpc	rep_j_y_pc	dem_j_y_pc	party
1	81913	44829	295.0	0.0	61	0.625000	0.375000	0.000000	0
2	66589	34303	239.0	0.0	96	0.555556	0.333333	0.000000	0
3	55436	29849	201.0	1.0	63	0.555556	0.555556	0.444444	0
4	55012	29892	191.0	0.0	92	0.555556	0.000000	0.333333	0
6	59768	31534	210.0	1.0	134	0.555556	0.555556	0.444444	0
...	...	...	...	...	...	...	...	...	...
595	65067	35907	167.0	1.0	62	0.555556	0.555556	0.444444	0
596	61137	32779	179.0	0.0	91	0.555556	0.333333	0.111111	0
597	58012	32112	220.0	0.0	224	0.555556	0.222222	0.111111	0
598	67120	34254	319.0	0.0	140	0.555556	0.444444	0.000000	0
599	56642	29786	250.0	1.0	57	0.500000	0.500000	0.500000	0

521 rows × 9 columns

Random Forest

In [20]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, precisi
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

import itertools
```

In [60]:

```
def get_accuracy(feature_lst, X, df):

    #set y dataset
    y = df['win_side']
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # create the model
    model = RandomForestClassifier()
    # train the model
    model.fit(X_train, y_train)
    # Test the model
    predictions = model.predict(X_train)
    # Make the predictions
    y_pred = model.predict(X_test)

    dict1 = {'features': tuple(feature_lst),
             'features_num': len(feature_lst),
             'accuracy': accuracy_score(y_test, y_pred),
             'f1': f1_score(y_test, y_pred),
             'precision': precision_score(y_test, y_pred),
             'recall': recall_score(y_test, y_pred)}
    return dict1
```

In [22]:

```
# Define the list
#'rep_jpc', 'dem_j_y_pc', 'rep_j_y_pc'
features_list = ['text_len', 'text_pre_len', 'num_utterances', 'develop_time',
                 'rep_jpc', 'party']

# Get all possible combinations of the list
combinations = []
for i in range(1, len(features_list) + 1):
    combinations += list(itertools.combinations(features_list, i))

for i in range(len(combinations)):
    combinations[i] = list(combinations[i] )
combinations
```

```
Out[22]: [['text_len'],
          ['text_pre_len'],
          ['num_utterances'],
          ['develop_time'],
          ['rep_jpc'],
          ['party'],
          ['text_len', 'text_pre_len'],
          ['text_len', 'num_utterances'],
          ['text_len', 'develop_time'],
          ['text_len', 'rep_jpc'],
          ['text_len', 'party'],
          ['text_pre_len', 'num_utterances'],
          ['text_pre_len', 'develop_time'],
          ['text_pre_len', 'rep_jpc'],
          ['text_pre_len', 'party'],
          ['num_utterances', 'develop_time'],
          ['num_utterances', 'rep_jpc'],
          ['num_utterances', 'party'],
          ['develop_time', 'rep_jpc'],
          ['develop_time', 'party'],
          ['rep_jpc', 'party'],
          ['text_len', 'text_pre_len', 'num_utterances'],
          ['text_len', 'text_pre_len', 'develop_time'],
          ['text_len', 'text_pre_len', 'rep_jpc'],
          ['text_len', 'text_pre_len', 'party'],
          ['text_len', 'num_utterances', 'develop_time'],
          ['text_len', 'num_utterances', 'rep_jpc'],
          ['text_len', 'num_utterances', 'party'],
          ['text_len', 'develop_time', 'rep_jpc'],
          ['text_len', 'develop_time', 'party'],
          ['text_len', 'rep_jpc', 'party'],
          ['text_pre_len', 'num_utterances', 'develop_time'],
          ['text_pre_len', 'num_utterances', 'rep_jpc'],
          ['text_pre_len', 'num_utterances', 'party'],
          ['text_pre_len', 'develop_time', 'rep_jpc'],
          ['text_pre_len', 'develop_time', 'party'],
          ['text_pre_len', 'rep_jpc', 'party'],
          ['num_utterances', 'develop_time', 'rep_jpc'],
          ['num_utterances', 'develop_time', 'party'],
          ['num_utterances', 'rep_jpc', 'party'],
          ['develop_time', 'rep_jpc', 'party'],
          ['text_len', 'text_pre_len', 'num_utterances', 'develop_time'],
          ['text_len', 'text_pre_len', 'num_utterances', 'rep_jpc'],
          ['text_len', 'text_pre_len', 'num_utterances', 'party'],
          ['text_len', 'text_pre_len', 'develop_time', 'rep_jpc'],
          ['text_len', 'text_pre_len', 'develop_time', 'party'],
          ['text_len', 'text_pre_len', 'rep_jpc', 'party'],
          ['text_len', 'num_utterances', 'develop_time', 'rep_jpc'],
          ['text_len', 'num_utterances', 'develop_time', 'party'],
          ['text_len', 'num_utterances', 'rep_jpc', 'party'],
          ['text_len', 'develop_time', 'rep_jpc', 'party'],
          ['text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc'],
          ['text_pre_len', 'num_utterances', 'develop_time', 'party'],
          ['text_pre_len', 'num_utterances', 'rep_jpc', 'party'],
          ['text_pre_len', 'develop_time', 'rep_jpc', 'party'],
          ['num_utterances', 'develop_time', 'rep_jpc', 'party'],
          ['text_len', 'text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc'],
          ['text_len', 'text_pre_len', 'num_utterances', 'develop_time', 'party'],
          ['text_len', 'text_pre_len', 'num_utterances', 'rep_jpc', 'party'],
          ['text_len', 'text_pre_len', 'develop_time', 'rep_jpc', 'party'],
          ['text_len', 'num_utterances', 'develop_time', 'rep_jpc', 'party'],
          ['text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc', 'party'],
          ['text_len',
           'text_pre_len',
           'num_utterances',
           'develop_time',
           'rep_jpc',
           'party']]
```

```
In [53]: len(combinations)
```

```
Out[53]: 63
```

For one epoch

In [73]:

```
results = []

# get accu from each diff features combinations
for feature_lst in combinations:
    results.append(get_accuracy(feature_lst, df_rf.loc[:,feature_lst], df_rf))

# get accu from ngram, bigram
results.append(get_accuracy(['ngram_text'], CountVectorizer().fit_transform(df['text_pre']), df))
results.append(get_accuracy(['bigram_text'], TfidfVectorizer().fit_transform(df['text_pre']), df))
```

In [74]:

```
# Create a DataFrame from the results list
results_df = pd.DataFrame(results)
results_df
```

Out[74]:

	features	features_num	accuracy	f1	precision	recall
0	(text_len,)	1	0.542857	0.641791	0.614286	0.671875
1	(text_pre_len,)	1	0.561905	0.616667	0.649123	0.587302
2	(num_utterances,)	1	0.533333	0.642336	0.594595	0.698413
3	(develop_time,)	1	0.580952	0.685714	0.695652	0.676056
4	(rep_jpc,)	1	0.685714	0.811429	0.689320	0.986111
...	...	...	...	...	...	...
60	(text_len, num_utterances, develop_time, rep_j...	5	0.533333	0.652482	0.582278	0.741935
61	(text_pre_len, num_utterances, develop_time, r...	5	0.600000	0.704225	0.746269	0.666667
62	(text_len, text_pre_len, num_utterances, devel...	6	0.561905	0.705128	0.670732	0.743243
63	(ngram_text,)	1	0.647619	0.775758	0.640000	0.984615
64	(bigram_text,)	1	0.657143	0.793103	0.657143	1.000000

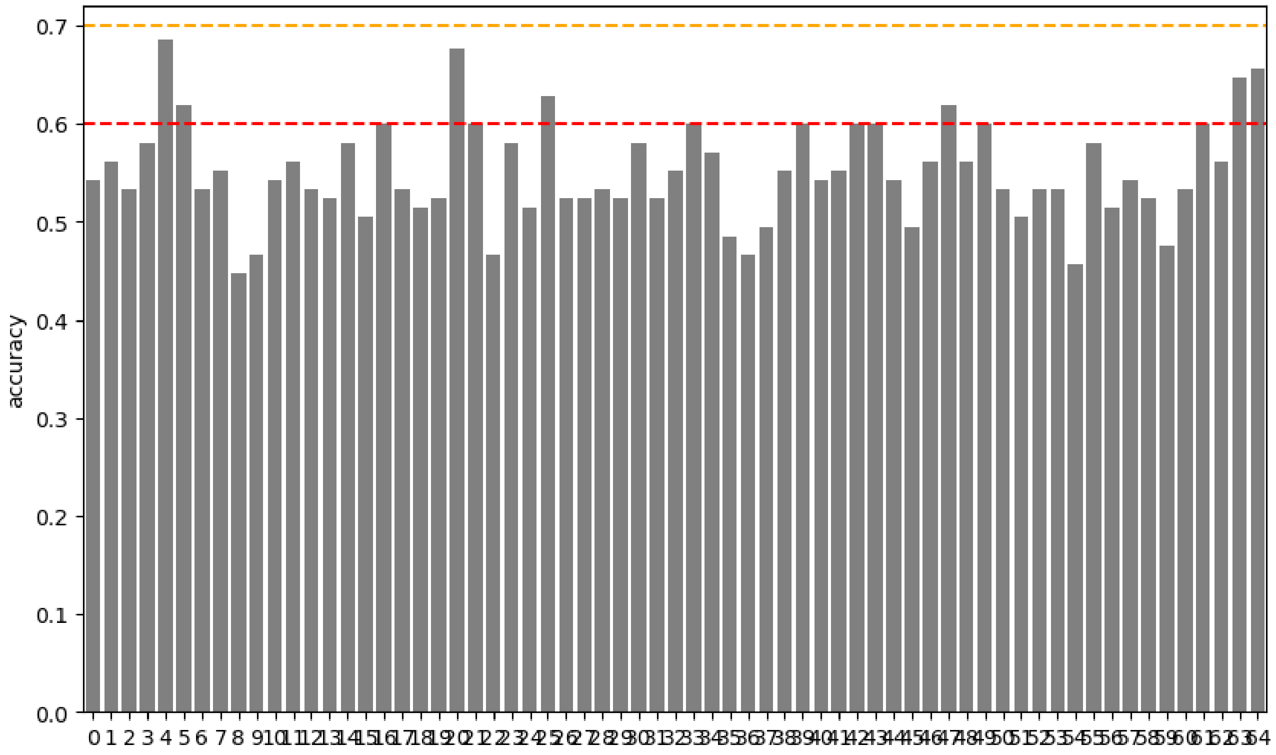
65 rows × 6 columns

In [75]:

```
plt.figure(figsize=(10, 6))
sns.barplot(x = results_df.index, y='accuracy', color='grey', data=results_df)
plt.axhline(y=0.6, color='red', linestyle='--')
plt.axhline(y=0.7, color='orange', linestyle='--')
```

Out[75]:

```
<matplotlib.lines.Line2D at 0x7fe8b30a6670>
```



In [76]:

```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.6), :]
```

Out[76]:

	features	features_num	accuracy	f1	precision	recall
4	(rep_jpc,)	1	0.685714	0.811429	0.689320	0.986111
5	(party,)	1	0.619048	0.764706	0.619048	1.000000
20	(rep_jpc, party)	2	0.676190	0.806818	0.682692	0.986111
25	(text_len, num_utterances, develop_time)	3	0.628571	0.731034	0.706667	0.757143
47	(text_len, num_utterances, develop_time, rep_jpc)	4	0.619048	0.740260	0.686747	0.802817
63	(ngram_text,)	1	0.647619	0.775758	0.640000	0.984615
64	(bigram_text,)	1	0.657143	0.793103	0.657143	1.000000

In [77]:

```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.7), :]
```

Out[77]:

features	features_num	accuracy	f1	precision	recall

For 50 epoch

In [78]:

```
results = []
count = 0
while(count <= 50):
    count += 1
    # get accu from each diff features combinations
    for feature_lst in combinations:
        results.append(get_accuracy(feature_lst, df_rf.loc[:,feature_lst], df_rf))

    # get accu from ngram, bigram
    results.append(get_accuracy(['ngram_text'], CountVectorizer().fit_transform(df['text_pre']), df))
    results.append(get_accuracy(['bigram_text'], TfidfVectorizer().fit_transform(df['text_pre']), df))
```

In [79]:

```
# Create a DataFrame from the results list
results_df = pd.DataFrame(results)
results_df
```

Out[79]:

	features	features_num	accuracy	f1	precision	recall
0	(text_len,)	1	0.561905	0.676056	0.685714	0.666667
1	(text_pre_len,)	1	0.542857	0.647059	0.619718	0.676923
2	(num_utterances,)	1	0.523810	0.642857	0.642857	0.642857
3	(develop_time,)	1	0.533333	0.679739	0.584270	0.812500
4	(rep_jpc,)	1	0.580952	0.731707	0.588235	0.967742
...	...	...	...	...	...	...
3310	(text_len, num_utterances, develop_time, rep_j...	5	0.542857	0.657143	0.676471	0.638889
3311	(text_pre_len, num_utterances, develop_time, r...	5	0.485714	0.608696	0.608696	0.608696
3312	(text_len, text_pre_len, num_utterances, devel...	6	0.561905	0.697368	0.616279	0.803030
3313	(ngram_text,)	1	0.676190	0.804598	0.673077	1.000000
3314	(bigram_text,)	1	0.676190	0.806818	0.682692	0.986111

3315 rows × 6 columns

In [80]:

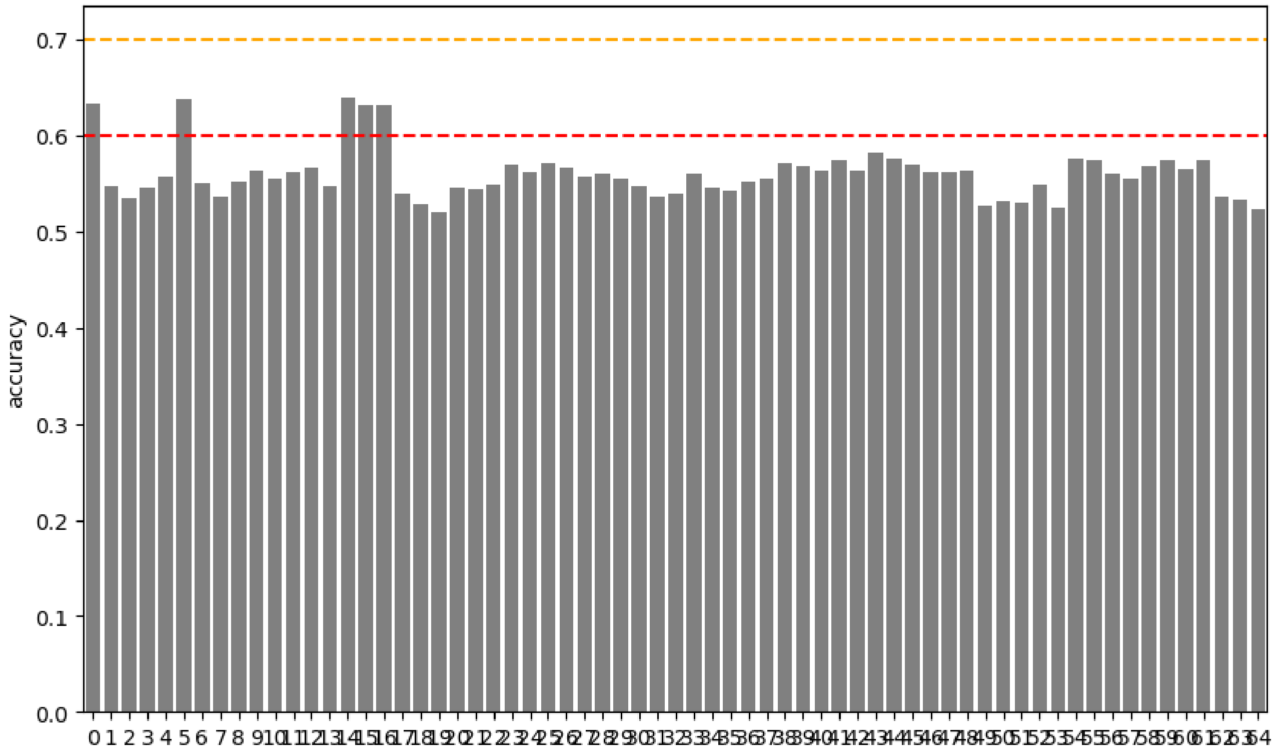
```
results_df = results_df.groupby('features', as_index = False).agg({'features_num': 'mean',
                                                                    'accuracy': 'mean',
                                                                    'f1': 'mean',
                                                                    'precision': 'mean',
                                                                    'recall': 'mean',})
```

In [81]:

```
plt.figure(figsize=(10, 6))
sns.barplot(x = results_df.index, y='accuracy', color='grey', data=results_df)
plt.axhline(y=0.6, color='red', linestyle='--')
plt.axhline(y=0.7, color='orange', linestyle='--')
```

Out[81]:

```
<matplotlib.lines.Line2D at 0x7fe907ae06a0>
```



In [84]:

```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.6), :].sort_values(by = 'accuracy', ascending = Fals
```

Out[84]:

	features	features_num	accuracy	f1	precision	recall
14	(party,)	1.0	0.639776	0.779434	0.639776	1.000000
5	(ngram_text,)	1.0	0.637908	0.776424	0.641069	0.986679
0	(bigram_text,)	1.0	0.633053	0.773451	0.636580	0.987719
16	(rep_jpc, party)	2.0	0.632680	0.773840	0.635639	0.992047
15	(rep_jpc,)	1.0	0.631559	0.772821	0.635128	0.989987

In [83]:

```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.7), :]
```

Out[83]:

features	features_num	accuracy	f1	precision	recall