In [305]:

```
#Importing Libraries and creating dataframe.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

data = pd.read_csv('/Users/juanrquilesjr/Documents/Machine_Learning-Spring_2019/project/training.txt',sep='\t', names = ('review', 'rating'))
```

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

In [306]:

```
pd.set_option('max_colwidth',900)
data.head()
```

Out[306]:

	review	rating
0	Drinks were bad, the hot chocolate was watered down and the latte had a burnt taste to it. The food was also poor quality, but the service was the worst part, their cashier was very rude.	1.0
1	This was the worst experience I've ever had a casual coffee/light fare place. The server disappeared for 20 minutes, just talking to his friend by the window as my girlfriend and I sat dumbfounded that this dude had the nerve to do that on the job. We're trying to make eye contact, but clearly getting paid to talk to his bud was more important to him. My girlfriend went up to the counter once the server disappeared into the back for another 5 minutes (what is this guy doing?) and asked if she should order food up there or something. The girl at the counter gives her a weird look and just says "I'll get your server." When they arrive from the back, they look over at our table and have a laugh. Yeah, leaving us hanging for half-a-goddamn hour at a place with only two other customers is not funny - but in retrospect, your collective incompetence and false sense of entitlement certa...	1.0
2	This is located on the site of the old Spruce St. Video. The mild cofee is very good and the pastris are great. At times, the service is slow even when it is not busy and at other times some patrons receive a complimentary mimosa drink. The WIFI is good.	3.0
3	I enjoyed coffee and breakfast twice at Toast during my recent visit to Philly. The first morning I enjoyed the Omelette du Jour which had a savory filling of roast tomato, portobello, artichoke, goat cheese, and wilted spinach. It was accompanied by a crisp small side salad of baby greens, tomato, and berries. The house dressing was light and complementary, not at all over powering. I just HAD to complement the chef. The barista recommended a pour over coffee and it did not disappoint. He prepared my cup with care and attention to time and form. I followed the pour over with a wonderful latte, again prepared very well. As I was leaving town, I decided to visit again for breakfast before my long trip ahead. This breakfast was a delicious Eggs Benedict combo that was perfectly done and complemented by mushrooms, roast tomato, and a generous helping of wilted greens. A rich cup of melt...	5.0
4	I love Toast! The food choices are fantastic - I love that they serve brunch all day, and their coffee is well brewed and prepared. I'm a fan of the large windows - it's the perfect location to sit and people watch for a little while. Nestled in Center City, Toast provides people like me who travel into the city for work a little haven to de-stress and kick back a little bit. The staff is wonderfully friendly and always eager to provide their own suggestions when you're not sure what to get. Now that school is starting back up it also makes it the perfect environment to settle down and do some studying. I love the fact that they play Maps by the Yeah Yeah Yeah's - and related Pandora stations - it really sets the mood to unwind or have casual conversation with some friends. Can't wait to go back!	5.0

In [3]:

```
data.shape
```

Out[3]:

```
(9994, 2)
```

In [4]:

```
data.rating.value_counts()
```

Out[4]:

```
4.0    2000
5.0    2000
1.0    2000
2.0    1999
3.0    1995
Name: rating, dtype: int64
```

In [337]:

```
#creating stop_word lists
stop_word = []
for i in data.review:
    words = word_tokenize(i)
    for j in words:
        if j.isalpha() != True:
            stop_word.append(j)
```

In [268]:

```
pos_list =[]
for i in open('/Users/juanrquilesjr/Documents/Machine_Learning-Spring_2019/project/positive-words.txt', 'r'):
    x = i.strip('\n')
    pos_list.append(x)
```

In [269]:

```
neg_list =[]
for i in open('/Users/juanrquilesjr/Documents/Machine_Learning-Spring_2019/project/negative-words.txt', 'r', encoding = "ISO-8859-1"):
    x = i.strip('\n')
    neg_list.append(x)
```

In [489]:

```
from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
drop = []
for x in ENGLISH_STOP_WORDS:
    drop.append(x)
```

In [380]:

```
missed_words = ['abiding', 'absent', 'acted', 'advised', 'all', 'american', 'anti', 'around', 'art', 'back', 'backlit', 'behaved', 'being', 'bid',
'blowing', 'blown', 'brand', 'bred', 'bull', 'cal', 'capacity', 'cash', 'cat', 'catch', 'catching', 'cats', 'class', 'coat', 'coated', 'conceived',
'connected', 'consuming', 'cost', 'counter', 'coup', 'cut', 'dally', 'defined', 'designed', 'determination', 'developed', 'dilly', 'ditch', 'down',
'drop', 'droping', 'dropping', 'duck', 'dummy', 'election', 'energy', 'established', 'expected', 'eye', 'faced', 'faces', 'far', 'fated', 'feature',
'fed', 'feed', 'fetched', 'first', 'flat', 'flexing', 'formed', 'full', 'genic', 'get', 'given', 'god', 'growing', 'handed', 'hands', 'hardline', 'h
ead', 'hearted', 'heavy', 'high', 'hit', 'ho', 'hyped', 'ill', 'in', 'informed', 'intentioned', 'interest', 'interested', 'israeli', 'jaw', 'job',
'kid', 'known', 'laid', 'large', 'last', 'lasting', 'law', 'leaning', 'left', 'less', 'lesser', 'life', 'light', 'line', 'liner', 'little', 'lived',
'logged', 'long', 'looking', 'low', 'lower', 'made', 'managed', 'mannered', 'mind', 'minded', 'mn', 'moving', 'multi', 'muscle', 'natured', 'new',
'non', 'notch', 'occupation', 'of', 'off', 'on', 'one', 'out', 'outs', 'over', 'paced', 'par', 'performing', 'polarization', 'positioned', 'pre', 'p
rice', 'priced', 'proliferation', 'proof', 'purpose', 'quality', 'rate', 'rated', 'razor', 'received', 'record', 'regarded', 'rigger', 'rigging', 'r
ock', 'rounded', 'run', 'satisfaction', 'saving', 'screw', 'second', 'seeking', 'self', 'selling', 'semi', 'semites', 'send', 'serving', 'set', 'set
ting', 'sh', 'short', 'sided', 'so', 'social', 'solver', 'sorted', 'spoon', 'star', 'stars', 'state', 'strapped', 'sub', 'sufficiency', 'sugar', 'te
mpered', 'than', 'the', 'thumb', 'thumbs', 'tier', 'time', 'tin', 'to', 'treated', 'treatment', 'two', 'ultra', 'un', 'up', 'ups', 'us', 'usage', 'u
se', 'used', 'user', 'valuation', 'violence', 'war', 'washed', 'water', 'watered', 'while', 'white', 'winded', 'wishers', 'wood', 'woods', 'working'
, 'world', 'yet']
```

```
In [372]: len(missed_words)

Out[372]: 107
```

```
In [452]: my_stop_words = pos_list + neg_list
          len(my_stop_words)

Out[452]: 6789
```

```
In [453]: #Splitting data into testing and training sets
          from sklearn.model_selection import train_test_split

          X_test, X_train, y_test, y_train = train_test_split(data['review'], data['rating'], test_size = .30)
```

```
In [454]: print(X_test.shape)
          print(X_train.shape)
          print(y_test.shape)
          print(y_test.shape)

          (6995,)
          (2999,)
          (6995,)
          (6995,)
```

```
In [496]: #initializing the vectorizer
          vectorizer = TfidfVectorizer(min_df = .03, max_df = 1.1, stop_words = my_stop_words , ngram_range = (1,3))
          #Learn vocabulary and idf, return term-document matrix.
          review_vect_train = vectorizer.fit_transform(X_train)
          review_vect_test = vectorizer.transform(X_test)

          /Users/juanrquilesjr/anaconda3/lib/python3.6/site-packages/sklearn/feature_extraction/text.py:301: UserWarning: Your stop_words may be inconsisten
          t with your preprocessing. Tokenizing the stop words generated tokens ['abiding', 'absent', 'acted', 'advised', 'all', 'american', 'anti', 'aroun
          d', 'art', 'back', 'backlit', 'behaved', 'being', 'bid', 'blowing', 'blown', 'brand', 'bred', 'bull', 'cal', 'capacity', 'cash', 'cat', 'catch',
          'catching', 'cats', 'class', 'coat', 'coated', 'conceived', 'connected', 'consuming', 'cost', 'counter', 'coup', 'cut', 'dally', 'defined', 'desig
          ned', 'determination', 'developed', 'dilly', 'ditch', 'down', 'drop', 'droping', 'dropping', 'duck', 'dummy', 'election', 'energy', 'established',
          'expected', 'eye', 'faced', 'faces', 'far', 'fated', 'feature', 'fed', 'feed', 'fetched', 'first', 'flat', 'flexing', 'formed', 'full', 'genic',
          'get', 'given', 'god', 'growing', 'handed', 'hands', 'hardline', 'head', 'hearted', 'heavy', 'high', 'hit', 'ho', 'hyped', 'ill', 'in', 'informe
          d', 'intentioned', 'interest', 'interested', 'israeli', 'jaw', 'job', 'kid', 'known', 'laid', 'large', 'last', 'lasting', 'law', 'leaning', 'lef
          t', 'less', 'lesser', 'life', 'light', 'line', 'liner', 'little', 'lived', 'logged', 'long', 'looking', 'low', 'lower', 'made', 'managed', 'manner
          ed', 'mind', 'minded', 'mn', 'moving', 'multi', 'muscle', 'natured', 'new', 'non', 'notch', 'occupation', 'of', 'off', 'on', 'one', 'out', 'outs',
          'over', 'paced', 'par', 'performing', 'polarization', 'positioned', 'pre', 'price', 'priced', 'proliferation', 'proof', 'purpose', 'quality', 'rat
          e', 'rated', 'razor', 'received', 'record', 'regarded', 'rigger', 'rigging', 'rock', 'rounded', 'run', 'satisfaction', 'saving', 'screw', 'secon
          d', 'seeking', 'self', 'selling', 'semi', 'semites', 'send', 'serving', 'set', 'setting', 'sh', 'short', 'sided', 'so', 'social', 'solver', 'sorte
          d', 'spoon', 'star', 'stars', 'state', 'strapped', 'sub', 'sufficiency', 'sugar', 'tempered', 'than', 'the', 'thumb', 'thumbs', 'tier', 'time', 't
          in', 'to', 'treated', 'treatment', 'two', 'ultra', 'un', 'up', 'ups', 'us', 'usage', 'use', 'used', 'user', 'valuation', 'violence', 'war', 'washe
          d', 'water', 'watered', 'while', 'white', 'winded', 'wishers', 'wood', 'woods', 'working', 'world', 'yet'] not in stop_words.
            'stop_words.' % sorted(inconsistent))
```

```
In [497]: # Model Generation Using Multinomial Naive BayesMultinomialNB()
          # Used to see how performing overall

          from sklearn.naive_bayes import MultinomialNB
          from sklearn import metrics

          clf = MultinomialNB()
          clf.fit(review_vect_train, y_train)
          pred= clf.predict(review_vect_test)
          print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, pred))

          MultinomialNB Accuracy: 0.4344344344344344
```

```
In [457]: from sklearn.naive_bayes import MultinomialNB
          from sklearn.linear_model import LogisticRegression
          from sklearn.neural_network import MLPClassifier
          from sklearn.svm import SVC
          from sklearn.ensemble import AdaBoostClassifier

          from sklearn.metrics import classification_report
```

```
In [458]: def get_score (model, X_train, X_test, y_train, y_test):
              model.fit(X_train, y_train)
              pred = model.predict(X_test)
              return metrics.accuracy_score(y_test, pred)
```

```
In [459]: from sklearn.model_selection import StratifiedKFold
          folds =  StratifiedKFold(n_splits = 5)
```

```
In [498]: #MultinomialNB
          mnb_score = []

          for train_index, test_index in folds.split(data['review'],data['rating']):
              X_train, X_test = data['review'][train_index], data['review'][test_index]
              y_train, y_test = data['rating'][train_index], data['rating'][test_index]
              review_vect_train = vectorizer.fit_transform(X_train)
              review_vect_test = vectorizer.transform(X_test)

              mnb_score.append(get_score(MultinomialNB(), review_vect_train, review_vect_test, y_train, y_test ))
```

```
In [499]: mnb_score

Out[499]: [0.36718359179589793,
           0.34217108554277137,
           0.3506753376688344,
           0.4317158579289645,
           0.4344344344344344]
```

```
In [500]: np.round(np.mean(mnb_score), decimals = 4)

Out[500]: 0.3852
```

```
In [501]: #Logistic Regression
          lr_score = []

          for train_index, test_index in folds.split(data['review'],data['rating']):
              X_train, X_test = data['review'][train_index], data['review'][test_index]
              y_train, y_test = data['rating'][train_index], data['rating'][test_index]
              review_vect_train1 = vectorizer.fit_transform(X_train)
              review_vect_test1 = vectorizer.transform(X_test)

              lr_score.append(get_score(LogisticRegression(multi_class = 'multinomial', max_iter = 3000, solver = 'saga'), review_vect_train1, review_vect_tes
          t1, y_train, y_test ))
```

```
In [467]: lr_score
```

```
Out[467]: [0.48024012006003003,
           0.4337168584292146,
           0.43571785892946474,
           0.46823411705852924,
           0.45295295295295296]
```

```
In [502]: np.round(np.mean(lr_score), decimals = 4)
```

```
Out[502]: 0.4162
```

```
In [503]: #Logistic Regression
          def get_score (model, X_train, X_test, y_train, y_test):
              model.fit(X_train, y_train)
              pred = model.predict(X_test)
              return classification_report(y_test, pred, output_dict = True)

          for train_index, test_index in folds.split(data['review'],data['rating']):
              X_train, X_test = data['review'][train_index], data['review'][test_index]
              y_train, y_test = data['rating'][train_index], data['rating'][test_index]
              review_vect_train1 = vectorizer.fit_transform(X_train)
              review_vect_test1 = vectorizer.transform(X_test)

          lr_classReport = get_score(LogisticRegression(multi_class = 'multinomial', max_iter = 3000, solver = 'saga', penalty = 'l1', C = 1 ), review_vect_tr
          ain1, review_vect_test1, y_train, y_test )
          lr_classReport
```

```
Out[503]: {'1.0': {'precision': 0.6263982102908278,
            'recall': 0.7,
            'f1-score': 0.6611570247933884,
            'support': 400},
           '2.0': {'precision': 0.391812865497076,
            'recall': 0.3358395989974937,
            'f1-score': 0.3616734143049932,
            'support': 399},
           '3.0': {'precision': 0.35172413793103446,
            'recall': 0.38345864661654133,
            'f1-score': 0.3669064748201439,
            'support': 399},
           '4.0': {'precision': 0.3401360544217687,
            'recall': 0.25,
            'f1-score': 0.2881844380403458,
            'support': 400},
           '5.0': {'precision': 0.48333333333333334,
            'recall': 0.58,
            'f1-score': 0.5272727272727273,
            'support': 400},
           'micro avg': {'precision': 0.44994994994994997,
            'recall': 0.44994994994994997,
            'f1-score': 0.44994994994994997,
            'support': 1998},
           'macro avg': {'precision': 0.438680920294808,
            'recall': 0.44985964912280696,
            'f1-score': 0.44103881584631976,
            'support': 1998},
           'weighted avg': {'precision': 0.43874789969278677,
            'recall': 0.44994994994994997,
            'f1-score': 0.4411156415433005,
            'support': 1998}}
```

```
In [570]: #SVM
          def get_score (model, X_train, X_test, y_train, y_test):
              model.fit(X_train, y_train)
              pred = model.predict(X_test)
              return metrics.accuracy_score(y_test, pred)

          svc_score = []

          for train_index, test_index in folds.split(data['review'],data['rating']):
              X_train, X_test = data['review'][train_index], data['review'][test_index]
              y_train, y_test = data['rating'][train_index], data['rating'][test_index]
              review_vect_train1 = vectorizer.fit_transform(X_train)
              review_vect_test1 = vectorizer.transform(X_test)

              svc_score.append(get_score(SVC(gamma = 'auto', kernel = 'linear', C =2), review_vect_train1, review_vect_test1, y_train, y_test ))
```

```
In [506]: svc_score
```

```
Out[506]: [0.27313656828414207,
           0.2911455727863932,
           0.304152076038019,
           0.36068034017008505,
           0.4014014014014014]
```

```
In [507]: np.round(np.mean(svc_score), decimals = 4)
```

```
Out[507]: 0.3261
```

In [508]:
```
#SVM
def get_score (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    return classification_report(y_test, pred, output_dict = True)

for train_index, test_index in folds.split(data['review'],data['rating']):
    X_train, X_test = data['review'][train_index], data['review'][test_index]
    y_train, y_test = data['rating'][train_index], data['rating'][test_index]
    review_vect_train1 = vectorizer.fit_transform(X_train)
    review_vect_test1 = vectorizer.transform(X_test)

svc_classReport = get_score(SVC(gamma = 'auto', kernel = 'linear', C =2), review_vect_train1, review_vect_test1, y_train, y_test )
svc_classReport
```

```
/Users/juanrquilesjr/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Out[508]:
```
{'1.0': {'precision': 0.6064073226544623,
  'recall': 0.6625,
  'f1-score': 0.6332138590203106,
  'support': 400},
 '2.0': {'precision': 0.2979865771812081,
  'recall': 0.556390977443609,
  'f1-score': 0.3881118881118881,
  'support': 399},
 '3.0': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 399},
 '4.0': {'precision': 0.35735735735735735,
  'recall': 0.2975,
  'f1-score': 0.3246930422919509,
  'support': 400},
 '5.0': {'precision': 0.4057971014492754,
  'recall': 0.49,
  'f1-score': 0.44394110985277463,
  'support': 400},
 'micro avg': {'precision': 0.4014014014014014,
  'recall': 0.4014014014014014,
  'f1-score': 0.4014014014014014,
  'support': 1998},
 'macro avg': {'precision': 0.33350967172846063,
  'recall': 0.40127819548872184,
  'f1-score': 0.3579919798553849,
  'support': 1998},
 'weighted avg': {'precision': 0.3336943728126827,
  'recall': 0.4014014014014014,
  'f1-score': 0.3581560799913202,
  'support': 1998}}
```

In [509]:
```
#Neural Networks
def get_score (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    return metrics.accuracy_score(y_test, pred)

mlp_score = []

for train_index, test_index in folds.split(data['review'],data['rating']):
    X_train, X_test = data['review'][train_index], data['review'][test_index]
    y_train, y_test = data['rating'][train_index], data['rating'][test_index]
    review_vect_train1 = vectorizer.fit_transform(X_train)
    review_vect_test1 = vectorizer.transform(X_test)

    mlp_score.append(get_score(MLPClassifier(hidden_layer_sizes = (100,100)), review_vect_train1, review_vect_test1, y_train, y_test ))
```

In [510]:
```
mlp_score
```

Out[510]:
```
[0.35617808904452225,
 0.3461730865432716,
 0.3711855927963982,
 0.384192096048024,
 0.4114114114114114]
```

In [511]:
```
np.round(np.mean(mlp_score), decimals = 4)
```

Out[511]:
```
0.3738
```

In [513]:

```
#Neural Networks
def get_score (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    return classification_report(y_test, pred, output_dict = True)

for train_index, test_index in folds.split(data['review'],data['rating']):
    X_train, X_test = data['review'][train_index], data['review'][test_index]
    y_train, y_test = data['rating'][train_index], data['rating'][test_index]
    review_vect_train1 = vectorizer.fit_transform(X_train)
    review_vect_test1 = vectorizer.transform(X_test)

mlp_classReport = get_score(MLPClassifier(hidden_layer_sizes = (100,100)), review_vect_train1, review_vect_test1, y_train, y_test )
mlp_classReport
```

Out[513]:

```
{'1.0': {'precision': 0.568445475638051,
  'recall': 0.6125,
  'f1-score': 0.5896510228640193,
  'support': 400},
 '2.0': {'precision': 0.3643410852713178,
  'recall': 0.3533834586466165,
  'f1-score': 0.3587786259541985,
  'support': 399},
 '3.0': {'precision': 0.3016627078384798,
  'recall': 0.3182957393483709,
  'f1-score': 0.3097560975609756,
  'support': 399},
 '4.0': {'precision': 0.29737609329446063,
  'recall': 0.255,
  'f1-score': 0.27456258411843876,
  'support': 400},
 '5.0': {'precision': 0.44471153846153844,
  'recall': 0.4625,
  'f1-score': 0.4534313725490196,
  'support': 400},
 'micro avg': {'precision': 0.4004004004004004,
  'recall': 0.4004004004004004,
  'f1-score': 0.4004004004004004,
  'support': 1998},
 'macro avg': {'precision': 0.39530738010076955,
  'recall': 0.40033583959899743,
  'f1-score': 0.3972359406093303,
  'support': 1998},
 'weighted avg': {'precision': 0.39536974795216684,
  'recall': 0.4004004004004004,
  'f1-score': 0.39729897221979255,
  'support': 1998}}
```

In [515]:

```
#AdaBoosting
def get_score (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    return metrics.accuracy_score(y_test, pred)

abc_score = []

for train_index, test_index in folds.split(data['review'],data['rating']):
    X_train, X_test = data['review'][train_index], data['review'][test_index]
    y_train, y_test = data['rating'][train_index], data['rating'][test_index]
    review_vect_train1 = vectorizer.fit_transform(X_train)
    review_vect_test1 = vectorizer.transform(X_test)

    abc_score.append(get_score(AdaBoostClassifier(base_estimator = MultinomialNB(), n_estimators = 50), review_vect_train1, review_vect_test1, y_tra
in, y_test ))
```

In [516]:

```
abc_score
```

Out[516]:

```
[0.3196598299149575,
 0.33016508254127064,
 0.3381690845422711,
 0.4117058529264632,
 0.44494494494494496]
```

In [517]:

```
np.round(np.mean(abc_score), decimals = 4)
```

Out[517]:

```
0.3689
```

In [520]:

```
#AdaBoosting
def get_score (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    return classification_report(y_test, pred, output_dict = True)

for train_index, test_index in folds.split(data['review'],data['rating']):
    X_train, X_test = data['review'][train_index], data['review'][test_index]
    y_train, y_test = data['rating'][train_index], data['rating'][test_index]
    review_vect_train1 = vectorizer.fit_transform(X_train)
    review_vect_test1 = vectorizer.transform(X_test)

abc_classReport = get_score(AdaBoostClassifier(base_estimator = MultinomialNB(), n_estimators = 250), review_vect_train1, review_vect_test1, y_train
, y_test)
abc_classReport
```

Out[520]:

```
{'1.0': {'precision': 0.6839237057220708,
  'recall': 0.6275,
  'f1-score': 0.6544980443285529,
  'support': 400},
 '2.0': {'precision': 0.43037974683544306,
  'recall': 0.3408521303258145,
  'f1-score': 0.38041958041958035,
  'support': 399},
 '3.0': {'precision': 0.37656903765690375,
  'recall': 0.45112781954887216,
  'f1-score': 0.4104903078677309,
  'support': 399},
 '4.0': {'precision': 0.3316708229426434,
  'recall': 0.3325,
  'f1-score': 0.33208489388264667,
  'support': 400},
 '5.0': {'precision': 0.4518348623853211,
  'recall': 0.4925,
  'f1-score': 0.4712918660287081,
  'support': 400},
 'micro avg': {'precision': 0.44894894894894893,
  'recall': 0.44894894894894893,
  'f1-score': 0.44894894894894893,
  'support': 1998},
 'macro avg': {'precision': 0.4548756351084765,
  'recall': 0.4488959899749373,
  'f1-score': 0.44975693850544374,
  'support': 1998},
 'weighted avg': {'precision': 0.4549270878040343,
  'recall': 0.44894894894894893,
  'f1-score': 0.4498112948561563,
  'support': 1998}}
```

In [320]:

```
from sklearn.metrics import classification_report,confusion_matrix

print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[429 149  20   6   6]
 [138 307 119  17   9]
 [ 48 132 284  91  29]
 [ 30  49  96 255 170]
 [ 22  40  41 166 346]]
             precision    recall  f1-score   support

        1.0       0.64      0.70      0.67       610
        2.0       0.45      0.52      0.48       590
        3.0       0.51      0.49      0.50       584
        4.0       0.48      0.42      0.45       600
        5.0       0.62      0.56      0.59       615

avg / total       0.54      0.54      0.54      2999
```

In [581]:

```
X = data['review']
y = data['rating']
```

In [582]:

```
from sklearn.model_selection import cross_val_score

def scoring():
    scoring_methods = 'f1_weighted', 'precision_weighted', 'recall_weighted'
    for i in scoring_methods:
        cvs = cross_val_score(clf, vectorizer.fit_transform(X), y, cv = 5, scoring = i)
        print(i + ':' + str(np.round((cvs),decimals = 2)))
        print(i + "(mean)" + ':' + str(np.round(np.mean(cvs), decimals = 2)))
```

In [583]:

```
scoring()
```

```
f1_weighted:[0.37 0.34 0.35 0.42 0.43]
f1_weighted(mean):0.38
precision_weighted:[0.37 0.37 0.35 0.43 0.43]
precision_weighted(mean):0.39
recall_weighted:[0.37 0.34 0.36 0.43 0.44]
recall_weighted(mean):0.39
```

In [19]:

```
# Logisitc Regression
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(multi_class = 'multinomial', penalty = 'l1', solver = 'saga', max_iter = 3000, C = 5, n_jobs = 3)
def scoring():
    scoring_methods = 'f1_weighted', 'precision_weighted', 'recall_weighted'
    for i in scoring_methods:
        cvs = cross_val_score(lr, review_vect_train, y_train, cv = 5, scoring = i)
        print(i + ':' + str(np.round((cvs),decimals = 2)))
        print(i + "(mean)" + ':' + str(np.round(np.mean(cvs), decimals = 2)))
```

```
In [20]: cvs = cross_val_score(lr, review_vect_train, y_train, cv = 5, scoring = 'f1_weighted')
         cvs

Out[20]: array([0.512112  , 0.52372306, 0.52584481, 0.53382736, 0.52898687])
```

```
In [21]: np.mean(cvs)

Out[21]: 0.5248988211172453
```

```
In [41]: #Neural Networks
         from sklearn.neural_network import MLPClassifier

         mlp = MLPClassifier(hidden_layer_sizes = (100,100,100), activation = 'logistic', solver = 'sgd')
         def scoring():
             scoring_methods = 'f1_micro', 'precision_micro', 'recall_micro'
             for i in scoring_methods:
                 cvs = cross_val_score(mlp, vectorizer.fit_transform(X), np.array(y), cv = 5, scoring = i)
                 print(i + ':' + str(np.round((cvs),decimals = 5)))
                 print(i + "(mean)" + ':' + str(np.round(np.mean(cvs), decimals = 5)))
```

```
In [42]: scoring()

         f1_micro:[0.2001 0.2001 0.2001 0.2001 0.1997]
         f1_micro(mean):0.20002
         precision_micro:[0.2001 0.2001 0.2001 0.2001 0.1997]
         precision_micro(mean):0.20002
         recall_micro:[0.2001 0.2001 0.2001 0.2001 0.2002]
         recall_micro(mean):0.20012
```

```
In [327]: # SVM
          from sklearn.svm import SVC

          svc = SVC()
          def scoring():
              scoring_methods = 'f1_micro', 'precision_micro', 'recall_micro'
              for i in scoring_methods:
                  cvs = cross_val_score(svc, vectorizer.fit_transform(X), np.array(y), cv = 5, scoring = i)
                  print(i + ':' + str(np.round((cvs),decimals = 5)))
                  print(i + "(mean)" + ':' + str(np.round(np.mean(cvs), decimals = 5)))
```

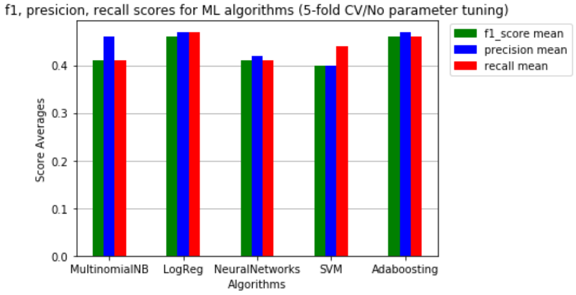
```
In [328]: scoring()

          f1_micro:[0.30665 0.32316 0.32816 0.38119 0.46046]
          f1_micro(mean):0.35993
          precision_micro:[0.30665 0.32316 0.32816 0.38119 0.46046]
          precision_micro(mean):0.35993
          recall_micro:[0.30665 0.32316 0.32816 0.38119 0.46046]
          recall_micro(mean):0.35993
```

```
In [443]: #AdaBoosting
          from sklearn.ensemble import AdaBoostClassifier

          abc = AdaBoostClassifier(n_estimators=75,learning_rate=1)
          def scoring():
              scoring_methods = 'f1_micro', 'precision_micro', 'recall_micro'
              for i in scoring_methods:
                  cvs = cross_val_score(abc, vectorizer.fit_transform(X), np.array(y), cv = 5, scoring = i)
                  print(i + ':' + str(np.round((cvs),decimals = 5)))
                  print(i + "(mean)" + ':' + str(np.round(np.mean(cvs), decimals = 5)))
```

```
In [444]: scoring()

          f1_micro:[0.43872 0.3987  0.4082  0.42221 0.43844]
          f1_micro(mean):0.42125
          precision_micro:[0.43872 0.3987  0.4082  0.42221 0.43844]
          precision_micro(mean):0.42125
          recall_micro:[0.43872 0.3987  0.4082  0.42221 0.43844]
          recall_micro(mean):0.42125
```

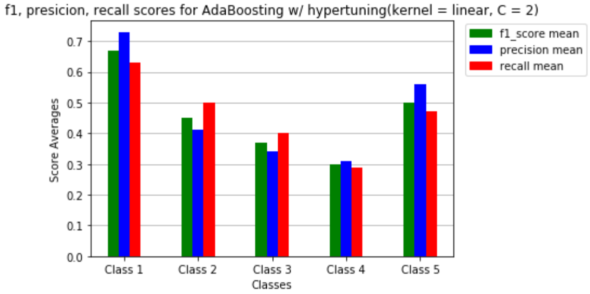
In [115]:

```python
# Graph No Hyperparameter Tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.41, 0.46, 0.41, 0.40,0.46]
precision_mean = [0.46, 0.47, 0.42, 0.40, 0.47]
recall_mean = [0.41, 0.47, 0.41,0.44, 0.46]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['MultinomialNB', 'LogReg', 'NeuralNetworks', 'SVM','Adaboosting'])
plt.title('f1, presicion, recall scores for ML algorithms (5-fold CV/No parameter tuning)')
plt.xlabel('Algorithms')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```

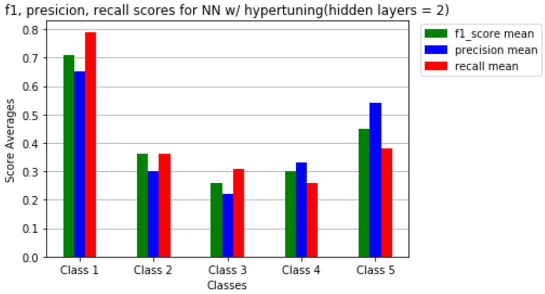


In [12]:

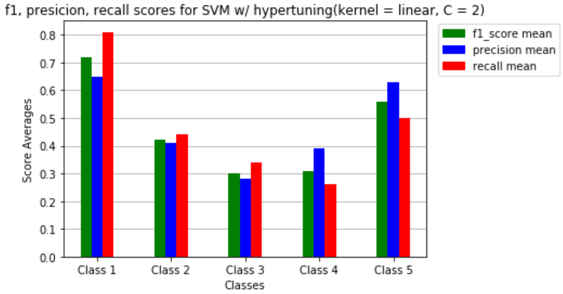
```python
# Graph AdaBoost Classes(F-1, Precision, and Recall) w/ hyperparameter tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.67, 0.45, 0.37, 0.30, 0.50]
precision_mean = [0.73, 0.41, 0.34, 0.31, 0.56]
recall_mean = [0.63, 0.50, 0.40, 0.29, 0.47]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for AdaBoosting w/ hypertuning(kernel = linear, C = 2)')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



```
In [17]: # Graph Neural Network(MLP) Classes(F-1, Precision, and Recall) w/ hyperparameter tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.71, 0.36, 0.26, 0.30, 0.45]
precision_mean = [0.65, 0.30, 0.22, 0.33, 0.54]
recall_mean = [0.79, 0.36, 0.31, 0.26, 0.38]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for NN w/ hypertuning(hidden layers = 2)')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```

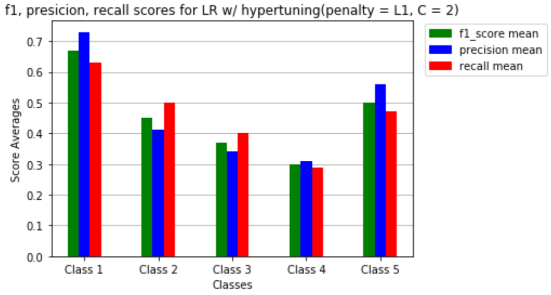


```
In [15]: # Graph SVM Classes(F-1, Precision, and Recall) w/ hyperparameter tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.72, 0.42, 0.30, 0.31, 0.56]
precision_mean = [0.65, 0.41, 0.28, 0.39, 0.63]
recall_mean = [0.81, 0.44, 0.34, 0.26, 0.50]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for SVM w/ hypertuning(kernel = linear, C = 2)')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



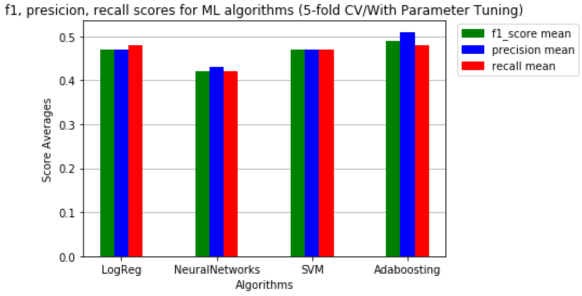
In [20]:

```
# Graph Linear Regression Classes(F-1, Precision, and Recall) w/ hyperparameter tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.67, 0.45, 0.37, 0.30, 0.50]
precision_mean = [0.73, 0.41, 0.34, 0.31, 0.56]
recall_mean = [0.63, 0.50, 0.40, 0.29, 0.47]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for LR w/ hypertuning(penalty = L1, C = 2)')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



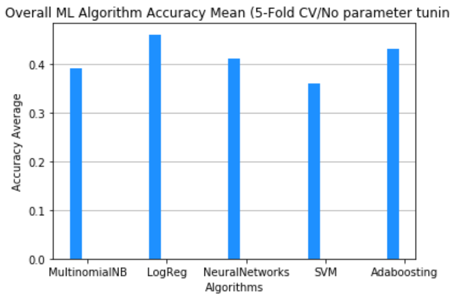
In [22]:

```
# Graph Hyperparameter Tuning
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [.47, .42, .47, .49 ]
precision_mean = [.47,.43, .47, .51 ]
recall_mean = [.48,.42, .47, .48 ]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['LogReg', 'NeuralNetworks', 'SVM','Adaboosting'])
plt.title('f1, presicion, recall scores for ML algorithms (5-fold CV/With Parameter Tuning)')
plt.xlabel('Algorithms')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



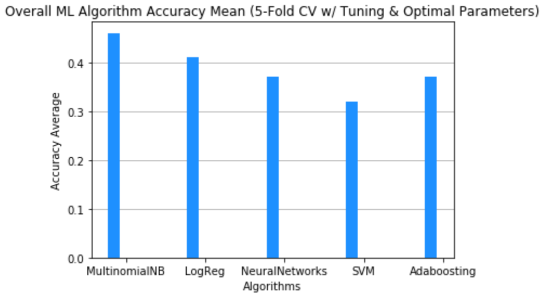
In [61]:

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
accuracy = [0.39, 0.46, 0.41, 0.36,0.43]
x_axis = np.arange(len(accuracy))
bar_width = 0.15
plt.bar(x_axis, accuracy, width = bar_width, color = 'dodgerblue', zorder = 2)

#labels
plt.xticks(x_axis + bar_width, ['MultinomialNB', 'LogReg', 'NeuralNetworks', 'SVM','Adaboosting'])
plt.title('Overall ML Algorithm Accuracy Mean (5-Fold CV/No parameter tuning)')
plt.xlabel('Algorithms')
plt.ylabel('Accuracy Average')

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



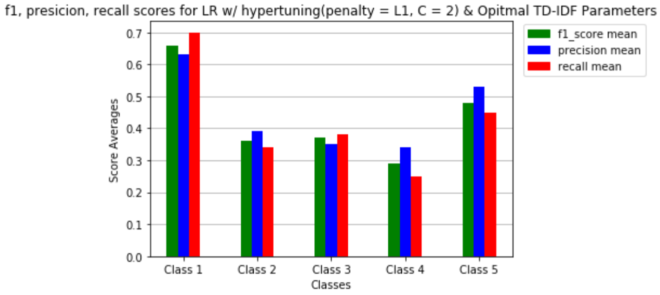
In [521]:

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
accuracy = [0.46, 0.41, 0.37, 0.32,0.37]
x_axis = np.arange(len(accuracy))
bar_width = 0.15
plt.bar(x_axis, accuracy, width = bar_width, color = 'dodgerblue', zorder = 2)

#labels
plt.xticks(x_axis + bar_width, ['MultinomialNB', 'LogReg', 'NeuralNetworks', 'SVM','Adaboosting'])
plt.title('Overall ML Algorithm Accuracy Mean (5-Fold CV w/ Tuning & Optimal Parameters)')
plt.xlabel('Algorithms')
plt.ylabel('Accuracy Average')

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



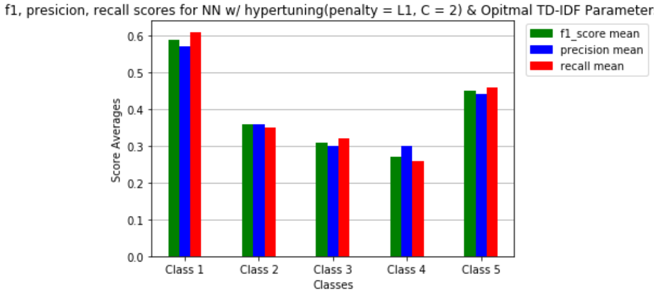
In [522]:

```
# Graph Linear Regression Classes(F-1, Precision, and Recall) w/ hyperparameter tuning & Optimal TD_IDF Parameters
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.66, 0.36, 0.37, 0.29, 0.48]
precision_mean = [0.63, 0.39, 0.35, 0.34, 0.53]
recall_mean = [0.70, 0.34, 0.38, 0.25, 0.45]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for LR w/ hypertuning(penalty = L1, C = 2) & Opitmal TD-IDF Parameters')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



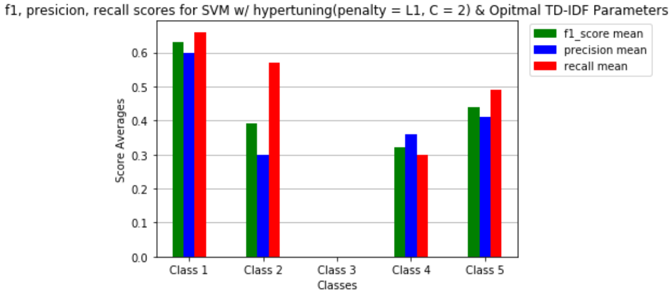
In [524]:

```
# Graph NN Classes(F-1, Precision, and Recall) w/ hyperparameter tuning & Optimal TD_IDF Parameters
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.59, 0.36, 0.31, 0.27, 0.45]
precision_mean = [0.57, 0.36, 0.30, 0.30, 0.44]
recall_mean = [0.61, 0.35, 0.32, 0.26, 0.46]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for NN w/ hypertuning(penalty = L1, C = 2) & Opitmal TD-IDF Parameters')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



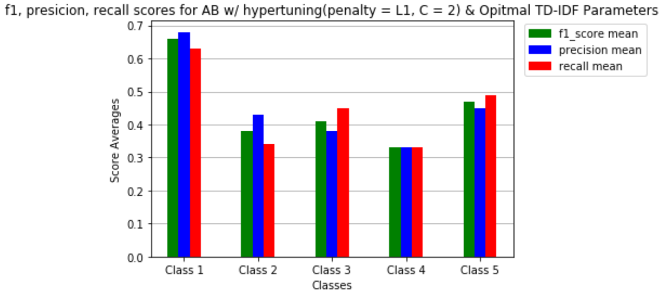
In [525]:

```python
# Graph SVM Classes(F-1, Precision, and Recall) w/ hyperparameter tuning & Optimal TD_IDF Parameters
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.63, 0.39, 0, 0.32, 0.44]
precision_mean = [0.60, 0.30, 0, 0.36, 0.41]
recall_mean = [0.66, 0.57, 0, 0.30, 0.49]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for SVM w/ hypertuning(penalty = L1, C = 2) & Opitmal TD-IDF Parameters')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



In [526]:

```python
# Graph AdaBoosting Classes(F-1, Precision, and Recall) w/ hyperparameter tuning & Optimal TD_IDF Parameters
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.66, 0.38, 0.41, 0.33, 0.47]
precision_mean = [0.68, 0.43, 0.38, 0.33, 0.45]
recall_mean = [0.63, 0.34, 0.45, 0.33, 0.49]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for AB w/ hypertuning(penalty = L1, C = 2) & Opitmal TD-IDF Parameters')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



In [530]:

```
data_test = pd.read_csv('/Users/juanrquilesjr/Documents/Machine_Learning-Spring_2019/project/test.txt',sep='\t', names = ('review', 'rating'))
data_test.head()
```

Out[530]:

	review	rating
0	Have only done tapas here. Must. Get. Ham croquettes. Those and the stuffed plantains. Lollipop chicken good, ribs good, Cuban sandwich ace, empanadas solid. Reasonably priced, service attentive, nice walkway to the restaurant.	4.0
1	I have heard of this place from multiple people over the past two or three years but somehow haven't paid a visit to it myself which I now bitterly regret. A friend took me to lunch there on a recent Thursday and I was simply amazed: from the very entrance all the way to my plate I felt transported to little Cuba in Florida. The atmosphere and decorum of the place distract you from everyday; the music, the waiters and the FOOD complete the job of taking you on vacation. The experience was so amazing that two days later I returned for dinner and the place did not disappoint. The food was dancing with flavors, attractively plated with generous portions one could share. The menu has impressive variety and the waiters know it very well and are happy to navigate it with you. I can't wait to return for another culinary excursion!	5.0
2	I came here for dinner on a weeknight around six and was seated immediately. Although the place is big it feels small and was very warm. Saying it was loud is an understatement. The music is blaring so get ready to scream if you want to be heard. Our waiter was attentive to the point of hovering. By the time we left it was packed. The food was the highlight - Bahamas coconut fish cooked well, nice sides and enough leftovers for lunch the next day. I would like to go back to dine outside, but unless I'm with a rowdy (drunk) crew the inside is not for me.	3.0
3	Food is good and the environment is very nice. It was a little loud for me but nothing I couldn't deal with. I took my dad and brother when they came to visit and they liked it as well.	4.0
4	Apparently you cannot get in to this place on weekends, but my friend and I were looking for a fun lunch spot and Yelp lead us here. It was busy for 3pm on a Saturday, but we got sat right away. After learning it's BYOB my friend went to the liquor store next door and came back with wine to they could mix us up a pitcher of Sangria. We got 4 tapas, ranging from $4-5 and 1 entree to share and there was plenty for a 3rd person or leftovers to take home. The stuffed plantains were a highlight and something I have not had at other restaurants!	4.0

In [559]:

```
print(data_test.shape)
print(data_test['review'].shape)
print(data_test['rating'].shape)
```

```
(1000, 2)
(1000,)
(1000,)
```

In [561]:

```
mnb = MultinomialNB()
mnb.fit(vectorizer.transform(data['review']), data['rating'])
```

Out[561]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [562]:

```
pred = mnb.predict(vectorizer.transform(data_test['review']))
print(classification_report(data_test['rating'], pred, output_dict = True))
```

```
{'1.0': {'precision': 0.5222672064777328, 'recall': 0.645, 'f1-score': 0.5771812080536913, 'support': 200}, '2.0': {'precision': 0.42268041237113
4, 'recall': 0.205, 'f1-score': 0.27609427609427606, 'support': 200}, '3.0': {'precision': 0.3804347826086957, 'recall': 0.35, 'f1-score': 0.36458
333333333337, 'support': 200}, '4.0': {'precision': 0.41935483870967744, 'recall': 0.26, 'f1-score': 0.32098765432098764, 'support': 200}, '5.0':
{'precision': 0.43103448275862066, 'recall': 0.75, 'f1-score': 0.5474452554744527, 'support': 200}, 'micro avg': {'precision': 0.442, 'recall': 0.
442, 'f1-score': 0.442, 'support': 1000}, 'macro avg': {'precision': 0.4351543445851721, 'recall': 0.442, 'f1-score': 0.41725834545534823, 'suppor
t': 1000}, 'weighted avg': {'precision': 0.4351543445851721, 'recall': 0.442, 'f1-score': 0.4172583454553481, 'support': 1000}}
```

In [553]:

```
lr = LogisticRegression(multi_class = 'multinomial', max_iter = 3000, solver = 'saga', penalty = 'l1', C = 1)
lr.fit(vectorizer.transform(data['review']), data['rating'])
```

Out[553]:

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=3000, multi_class='multinomial',
          n_jobs=None, penalty='l1', random_state=None, solver='saga',
          tol=0.0001, verbose=0, warm_start=False)
```

In [560]:

```
pred = lr.predict(vectorizer.transform(data_test['review']))
print(classification_report(data_test['rating'], pred, output_dict = True))
```

```
{'1.0': {'precision': 0.5608695652173913, 'recall': 0.645, 'f1-score': 0.6, 'support': 200}, '2.0': {'precision': 0.4634146341463415, 'recall': 0.
38, 'f1-score': 0.4175824175824176, 'support': 200}, '3.0': {'precision': 0.4260355029585799, 'recall': 0.36, 'f1-score': 0.39024390243902435, 'su
pport': 200}, '4.0': {'precision': 0.436046511627907, 'recall': 0.375, 'f1-score': 0.40322580645161293, 'support': 200}, '5.0': {'precision': 0.50
18867924528302, 'recall': 0.665, 'f1-score': 0.572043010752688, 'support': 200}, 'micro avg': {'precision': 0.485, 'recall': 0.485, 'f1-score': 0.
485, 'support': 1000}, 'macro avg': {'precision': 0.47765060128061, 'recall': 0.485, 'f1-score': 0.47661902744514856, 'support': 1000}, 'weighted
avg': {'precision': 0.47765060128061, 'recall': 0.485, 'f1-score': 0.4766190274451486, 'support': 1000}}
```

In [563]:

```
nn = MLPClassifier(hidden_layer_sizes = (100,100))
nn.fit(vectorizer.transform(data['review']), data['rating'])
```

Out[563]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 100), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

In [564]:

```
pred = nn.predict(vectorizer.transform(data_test['review']))
print(classification_report(data_test['rating'], pred, output_dict = True))
```

```
{'1.0': {'precision': 0.5245098039215687, 'recall': 0.535, 'f1-score': 0.5297029702970297, 'support': 200}, '2.0': {'precision': 0.378531073446327
7, 'recall': 0.335, 'f1-score': 0.3554376657824934, 'support': 200}, '3.0': {'precision': 0.3352272727272727, 'recall': 0.295, 'f1-score': 0.31382
978723404253, 'support': 200}, '4.0': {'precision': 0.3023255813953488, 'recall': 0.325, 'f1-score': 0.3132530120481927, 'support': 200}, '5.0':
{'precision': 0.4605263157894737, 'recall': 0.525, 'f1-score': 0.4906542056074767, 'support': 200}, 'micro avg': {'precision': 0.403, 'recall': 0.
403, 'f1-score': 0.403, 'support': 1000}, 'macro avg': {'precision': 0.40022400945599834, 'recall': 0.403, 'f1-score': 0.40057552819384695, 'suppo
rt': 1000}, 'weighted avg': {'precision': 0.40022400945599834, 'recall': 0.403, 'f1-score': 0.40057552819384706, 'support': 1000}}
```

In [566]:

```
svm = SVC(kernel = 'linear', C = 2)
svm.fit(vectorizer.transform(data['review']), data['rating'])
```

Out[566]:

```
SVC(C=2, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
In [567]: pred = svm.predict(vectorizer.transform(data_test['review']))
          print(classification_report(data_test['rating'], pred, output_dict = True))
```

```
{'1.0': {'precision': 0.5304347826086957, 'recall': 0.61, 'f1-score': 0.5674418604651164, 'support': 200}, '2.0': {'precision': 0.4058823529411764
7, 'recall': 0.345, 'f1-score': 0.37297297297297294, 'support': 200}, '3.0': {'precision': 0.38285714285714284, 'recall': 0.335, 'f1-score': 0.357
3333333333333, 'support': 200}, '4.0': {'precision': 0.4340659340659341, 'recall': 0.395, 'f1-score': 0.41361256544502617, 'support': 200}, '5.0':
{'precision': 0.5349794238683128, 'recall': 0.65, 'f1-score': 0.5869074492099323, 'support': 200}, 'micro avg': {'precision': 0.467, 'recall': 0.4
67, 'f1-score': 0.467, 'support': 1000}, 'macro avg': {'precision': 0.4576439272682524, 'recall': 0.46699999999999997, 'f1-score': 0.4596536362852
762, 'support': 1000}, 'weighted avg': {'precision': 0.45764392726825237, 'recall': 0.467, 'f1-score': 0.4596536362852762, 'support': 1000}}
```

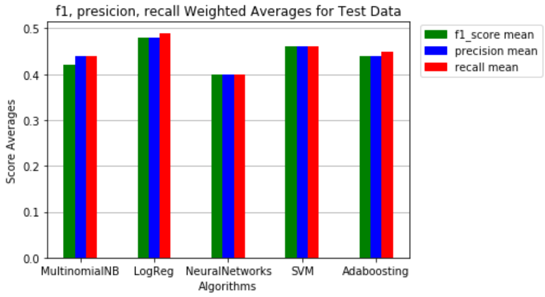
```
In [568]: abc = AdaBoostClassifier(base_estimator = MultinomialNB(), n_estimators = 250)
          abc.fit(vectorizer.transform(data['review']), data['rating'])
```

```
Out[568]: AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True),
                    learning_rate=1.0, n_estimators=250, random_state=None)
```

```
In [569]: pred = abc.predict(vectorizer.transform(data_test['review']))
          print(classification_report(data_test['rating'], pred, output_dict = True))
```

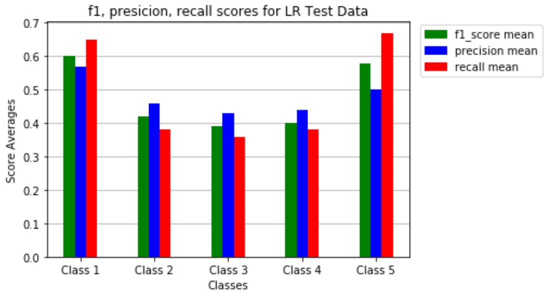
```
{'1.0': {'precision': 0.5440414507772021, 'recall': 0.525, 'f1-score': 0.5343511450381679, 'support': 200}, '2.0': {'precision': 0.370786516853932
6, 'recall': 0.33, 'f1-score': 0.3492063492063492, 'support': 200}, '3.0': {'precision': 0.37055837563451777, 'recall': 0.365, 'f1-score': 0.36775
81863979849, 'support': 200}, '4.0': {'precision': 0.4473684210526316, 'recall': 0.34, 'f1-score': 0.38636363636363635, 'support': 200}, '5.0':
{'precision': 0.475, 'recall': 0.665, 'f1-score': 0.5541666666666666, 'support': 200}, 'micro avg': {'precision': 0.445, 'recall': 0.445, 'f1-scor
e': 0.445, 'support': 1000}, 'macro avg': {'precision': 0.44155095286365686, 'recall': 0.445, 'f1-score': 0.43836919673456104, 'support': 1000},
'weighted avg': {'precision': 0.4415509528636568, 'recall': 0.445, 'f1-score': 0.438369196734561, 'support': 1000}}
```

```
In [572]: # Graph Test Data f-1, precision, recall weighted averages
          import matplotlib.pyplot as plt
          import matplotlib.patches as mpatches
          %matplotlib inline

          #Data
          f1_score_mean = [0.42, 0.48, 0.40, 0.46,0.44]
          precision_mean = [0.44, 0.48, 0.40, 0.46, 0.44]
          recall_mean = [0.44, 0.49, 0.40,0.46, 0.45]

          x_axis = np.arange(len(f1_score_mean))

          bar_width = 0.15


          plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
          plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
          plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


          #labels
          plt.xticks(x_axis + bar_width, ['MultinomialNB', 'LogReg', 'NeuralNetworks', 'SVM','Adaboosting'])
          plt.title('f1, presicion, recall Weighted Averages for Test Data')
          plt.xlabel('Algorithms')
          plt.ylabel('Score Averages')

          #legend
          green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
          blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
          red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
          plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

          plt.grid(axis = 'y')
          plt.autoscale(enable=True, axis='x', tight=None)
```



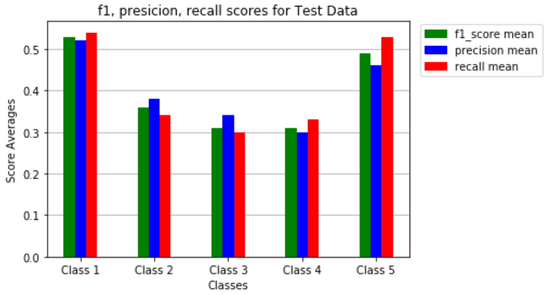
In [575]:

```
# Graph Linear Regression Classes(F-1, Precision, and Recall) Test Data
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.60, 0.42, 0.39, 0.40, 0.58]
precision_mean = [0.57, 0.46, 0.43, 0.44, 0.50]
recall_mean = [0.65, 0.38, 0.36, 0.38, 0.67]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for LR Test Data')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



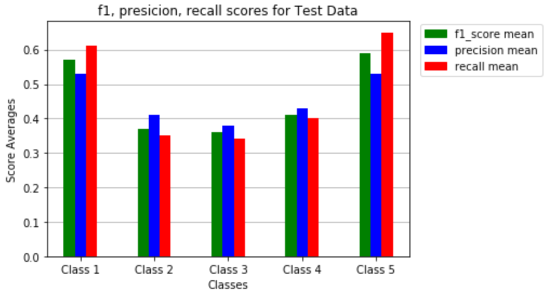
In [576]:

```
# Graph Neural Networks Classes(F-1, Precision, and Recall) Test Data
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.53, 0.36, 0.31, 0.31, 0.49]
precision_mean = [0.52, 0.38, 0.34, 0.30, 0.46]
recall_mean = [0.54, 0.34, 0.30, 0.33, 0.53]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for Test Data')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



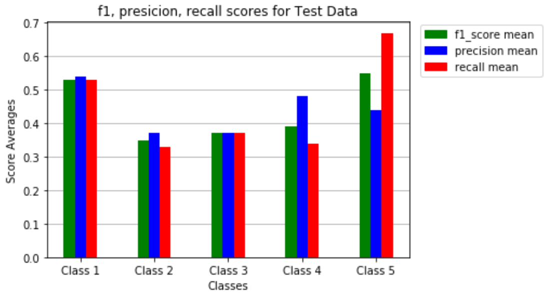
In [577]:

```
# Graph SVM Classes(F-1, Precision, and Recall) Test Data
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.57, 0.37, 0.36, 0.41, 0.59]
precision_mean = [0.53, 0.41, 0.38, 0.43, 0.53]
recall_mean = [0.61, 0.35, 0.34, 0.40, 0.65]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for Test Data')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```



In [578]:

```
# Graph AdaBoosting Classes(F-1, Precision, and Recall) Test Data
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

#Data
f1_score_mean = [0.53, 0.35, 0.37, 0.39, 0.55]
precision_mean = [0.54, 0.37, 0.37, 0.48, 0.44]
recall_mean = [0.53, 0.33, 0.37, 0.34, 0.67]

x_axis = np.arange(len(f1_score_mean))

bar_width = 0.15


plt.bar(x_axis, f1_score_mean, width = bar_width, color = 'green', zorder = 2)
plt.bar(x_axis + bar_width, precision_mean, width = bar_width, color = 'blue', zorder = 2)
plt.bar(x_axis + bar_width+0.15, recall_mean, width = bar_width, color = 'red', zorder = 2)


#labels
plt.xticks(x_axis + bar_width, ['Class 1', 'Class 2', 'Class 3', 'Class 4','Class 5'])
plt.title('f1, presicion, recall scores for Test Data')
plt.xlabel('Classes')
plt.ylabel('Score Averages')

#legend
green_bar = mpatches.Patch(color = 'green', label = 'f1_score mean')
blue_bar = mpatches.Patch(color = 'blue', label = 'precision mean')
red_bar = mpatches.Patch(color = 'red', label = 'recall mean')
plt.legend(handles=[green_bar,blue_bar, red_bar], loc = 'upper center', bbox_to_anchor=(1.2, 1.013))

plt.grid(axis = 'y')
plt.autoscale(enable=True, axis='x', tight=None)
```


f1, presicion, recall scores for Test Data
Score Averages
Class 1
Class 2
Class 3
Class 4
Class 5
Classes
🟩 f1_score mean
🟦 precision mean
🟥 recall mean