Using Text Analysis of World News Headlines to Predict Stock Market Movement

An investigation of model performance under the p >> n scenario

Yanji Du

CFRM 521

Spring 2020

1. Introduction

In the machine learning problems we've encountered this quarter so far, the number of observations (`n`) typically have been far greater than the number of dimensions/features (`p`). While this situation is common in many datasets, there also exist applications of ML on datasets where the number of dimensions is far greater than the number of observations (`p >> n`).

Problems like this are particularly common in text analysis and in genomics/computational biology, where the number of dimensions can quickly explode if words/n-grams (https://en.wikipedia.org/wiki/N-gram), or genes are one-hot encoded as features. Analysis of text data can be commonly found in financial applications; for example, the analysis of news and their correlation with stock market movements (Khadjeh Nassirtoussi et al., 2014).

In this experiment, we will use a dataset of the Top 25 News Headlines of each date from 2008-08-08 to 2016-07-01 encoded as n-grams as predictors, and the day's Dow Jones Industrial Average (https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average) index movement (i.e. increase, decrease encoded as a binary variable) as the response (Sun, 2016). With this binary classification task, we plan to investigate the performance of several classification algorithms we covered in this course in the domain where `p >> n` .

2. Problem Setup

For these classification algorithms (detailed below), we want to evaluate the following empirically:

- Which classifier algorithm (default parameters) performs the best in terms of test set AUROC?
- With hyperparameter tuning, which algorithm performs the best in terms of test set AUROC?

Classification algorithms we will evaluate (Géron, 2017):

1. Logistic Regression
    - Relatively simple classifier model to use a baseline to compare other models against.
    - Regularization parameter C will be an important hyperparameter to tune.
    - Can try both L1 and L2 regularization.
2. SVM Classifier (Linear kernel)
    - SVMs have been shown to perform well even in the `p >> n` domain (Hastie et al., 2009).
    - Regularization parameter C will be an important hyperparameter to tune.
3. SVM Classifier (RBF kernel)
    - Try to see if RBF kernel will result in better performance (if data is not linearly seperable).
4. Random Forest Classifier
    - A tree-based classifier, different from above approaches.
    - Good for handling non-linear data.
    - Prone to overfitting, so important to tune hyperparameteres like tree depth.
5. K-Nearest Neighbors Classifier
    - Non-parametric classifier (no assumptions), different from above approaches.
    - May not perform well in high dimensions.

These models were selected since they were covered in this course, for their regularization capabilities (likely important to dealing with high dimensional data), and for their popularity in ML applications in literature. We acknolwedge that there exist other classification models we covered in this course (boosting classifiers; MLPs/DNNs) and models that exist outside of this course (Naive Bayes classifier; more complex DNNs), but we have deliberately chosen to the limit the scope of this experiment.

One prior experiment we'll perform before evaluating the above is seeing which set of n-grams performs the best based on models with default parameters. Since there are many ways to create n-grams of sentences/documents (i.e. 1-grams, 2-grams, 3-grams, etc. and any combination), we want to try to limit the scope of our experiement by evaluating these scenarios first:

- Only 1-grams
- Only 2-grams
- Both 1-grams and 2-grams
- 1, 2, & 3- grams

Note: even before running the experiment, our hunch based on what we've learned so far in this course is that we will not get ground-breaking results in stock market movement prediction here, just using the Top 25 news headlines for each date. It does not feel like sufficient signal to accurately model the movement of the DJIA, which monitors stocks of 30 top companies in various industries. **Given this, we feel that if we are able to extract over 50% test set AUROC at all, this is promising in that we are able to extract at least some signal from the data**.

3a. Data Loading

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('ggplot')
```

In [2]:
```
djia = pd.read_csv('./data/upload_DJIA_table.csv', parse_dates=[0])
news = pd.read_csv('./data/RedditNews.csv', parse_dates=[0])
comb = pd.read_csv('./data/Combined_News_DJIA.csv', parse_dates=[0])
```

In [3]:
```
djia = djia.sort_values('Date', ascending=True).reset_index(drop=True)
```

Let's take a look at the data:

In [4]:
```
comb.head(3)
```

Out[4]:

	Date	Label	Top1	Top2	Top3	Top4	Top5	Top6	Top7	Top8	...	Top16	Top17	
0	2008-08-08	0	b"Georgia 'downs two Russian warplanes' as cou...	b'BREAKING: Musharraf to be impeached.'	b'Russia Today: Columns of troops roll into So...	b'Russian tanks are moving towards the capital...	b"Afghan children raped with 'impunity,' U.N. ...	b'150 Russian tanks have entered South Ossetia...	b"Breaking: Georgia invades South Ossetia, Rus...	b"The 'enemy combatent' trials are nothing but...	...	b'Georgia Invades South Ossetia - if Russia ge...	b'Al-Qaeda Faces Islamist Backlash'	b
1	2008-08-11	1	b'Why wont America and Nato help us? If they w...	b'Bush puts foot down on Georgian conflict'	b"Jewish Georgian minister: Thanks to Israeli ...	b'Georgian army flees in disarray as Russians ...	b"Olympic opening ceremony fireworks 'faked'"	b'What were the Mossad with fraudulent New Zea...	b'Russia angered by Israeli military sale to G...	b'An American citizen living in S.Ossetia blam...	...	b'Israel and the US behind the Georgian aggres...	b'"Do not believe TV, neither Russian nor Geor...	b
2	2008-08-12	0	b'Remember that adorable 9-year-old who sang a...	b"Russia 'ends Georgia operation'"	b'"If we had no sexual harassment we would hav...	b"Al-Qa'eda is losing support in Iraq because ...	b'Ceasefire in Georgia: Putin Outmaneuvers the...	b'Why Microsoft and Intel tried to kill the XO...	b'Stratfor: The Russo-Georgian War and the Bal...	b"I'm Trying to Get a Sense of This Whole Geor...	...	b'U.S. troops still in Georgia (did you know t...	b'Why Russias response to Georgia was right'	

3 rows × 27 columns

In [5]:
```
# Calculate the daily return
for index, row in djia.iterrows():
    if index == 0:
        djia.loc[index, 'DailyReturn'] = np.nan
    else:
        djia.loc[index, 'DailyReturn'] = djia.loc[index, 'Close']/djia.loc[index-1, 'Close']-1
```

In [6]:
```
# Calculate stock market movement. 1 = index increased; 0 = index decreased or stayed the same
djia['Movement'] = djia.apply(lambda x: 1 if x['DailyReturn'] > 0 else 0, axis=1)
```

In [7]:
```python
# Standardize daily returns; but don't subtract the mean
daily_return_mean = djia['DailyReturn'].mean()
daily_return_std = djia['DailyReturn'].std()

djia['DailyReturn_Norm'] = djia.apply(lambda x: (x['DailyReturn'])/daily_return_std, axis=1)
```

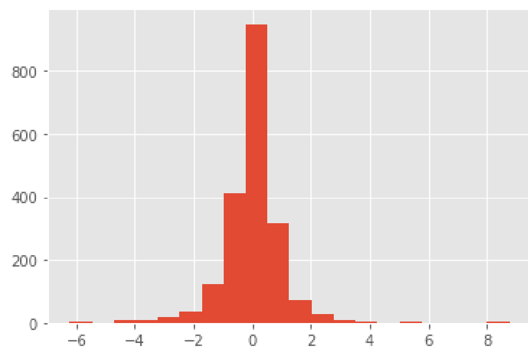
In [8]:
```python
# Join with combined dataframe
comb = comb.merge(
    djia[['Date','DailyReturn_Norm']],
    how='left', on='Date'
)

comb = comb[1:len(comb)-1].reset_index(drop=True)
```

3b. Data Exploratory Analysis

Distribution of normalized Daily Returns.

In [9]:
```python
djia['DailyReturn_Norm'].hist(bins=20)
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1192ded10>
```



In [10]:
```python
# dataset is from 8/11/2008 to 6/30/2016 (roughly 8 years)
print(comb['Date'].min())
print(comb['Date'].max())
```

```
2008-08-11 00:00:00
2016-06-30 00:00:00
```

In [11]:
```python
# 1987 dates
comb.shape
```

Out[11]:
```
(1987, 28)
```

In [12]:
```python
# slightly imbalanced dataset
# 1 means market close price rose over previous close or didn't move; 0 means decreased
comb['Label'].value_counts()
```

Out[12]:
```
1    1064
0     923
Name: Label, dtype: int64
```

3c. Data Processing

Fill NA's with blank strings.

In [13]:
```python
comb.fillna('', inplace=True)
```

In [14]:
```
# Shift label back 1 day; news today should affect stock price close tomorrow, giving markets enough time t
o react
# comb['Label_Tomorrow'] = comb['Label'].shift(-1)

# Remove last data point since there's no label after we shifted back
# comb = comb[0:len(comb)-1]
```

Create X and y matrices for modeling.

Train/test split by time.

- Train set: `2008-08-11` to `2014-12-31`
- Test set: `2015-01-01` to `2016-06-30`

In [15]:
```
y_train = comb[(comb['Date'] < pd.to_datetime('2015-01-01'))]['Label'].to_numpy()
y_test = comb[(comb['Date'] >= pd.to_datetime('2015-01-01'))]['Label'].to_numpy()
```

Note: Baseline accuracies of train and test set are not aligned, meaning in the train time period, there were relatively more daily return increases than decreases, whereas the test was pretty much balanced.

In [16]:
```
# Check the baseline accuracy for Train and Test sets:
print(np.unique(y_train, return_counts=True))
print(np.unique(y_test, return_counts=True))
```

```
(array([0, 1]), array([737, 873]))
(array([0, 1]), array([186, 191]))
```

In [17]:
```
baseline_acc_train = 873/(737+873)
baseline_acc_test = 191/(186+191)

print('Train Set baseline accuracy:', baseline_acc_train)
print('Test Set baseline accuracy:', baseline_acc_test)
```

```
Train Set baseline accuracy: 0.5422360248447204
Test Set baseline accuracy: 0.506631299734748
```

In [18]:
```
X_train_raw = comb[(comb['Date'] < pd.to_datetime('2015-01-01'))].drop(['Date','Label','DailyReturn_Norm'],
axis=1)
X_test_raw = comb[(comb['Date'] >= pd.to_datetime('2015-01-01'))].drop(['Date','Label','DailyReturn_Norm'],
axis=1)
```

In [19]:
```
# Weight feature vectors by daily return normalized.
drn_weights_train = comb[(comb['Date'] < pd.to_datetime('2015-01-01'))]['DailyReturn_Norm'].abs().to_numpy
()
drn_weights_test = comb[(comb['Date'] >= pd.to_datetime('2015-01-01'))]['DailyReturn_Norm'].abs().to_numpy
()
```

In [20]:
```
# y_train = comb[(comb['Date'] < pd.to_datetime('2015-01-01'))]['Label_Tomorrow'].to_numpy()
# y_test = comb[(comb['Date'] >= pd.to_datetime('2015-01-01'))]['Label_Tomorrow'].to_numpy()
```

In [21]:
```
# X_train_raw = comb[(comb['Date'] < pd.to_datetime('2015-01-01'))].drop(['Date','Label','Label_Tomorrow'],
axis=1)
# X_test_raw = comb[(comb['Date'] >= pd.to_datetime('2015-01-01'))].drop(['Date','Label','Label_Tomorrow'],
axis=1)
```

Pre-processing words: remove numbers, punctuation (non-alpha), stop words, [lemmatize (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20)](https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20) words, etc.

In [22]:
```python
import nltk
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

# Download required nltk resources/dictionaries/corpus
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')

# Note: below get_wordnet_pos function sourced from https://www.machinelearningplus.com/nlp/lemmatization-e
xamples-python/
# Credit to author
def get_wordnet_pos(word):
    # Map Part of Speech (POS) tag chars that lemmatize() accepts
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {'J': wordnet.ADJ,
                'N': wordnet.NOUN,
                'V': wordnet.VERB,
                'R': wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)


def process_words(headline):
    # Only keep alphabet letters
    alpha_only = re.sub("[^a-zA-Z]", " ", headline)

    # Convert to lower case and split
    words = alpha_only.lower().split()

    # Remove stop words
    stop_word_corpus = set(stopwords.words('english'))
    non_stop_words = [w for w in words if w not in stop_word_corpus]

    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    headline_lemmatized = [lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in non_stop_words]

    return(' '.join(headline_lemmatized))
```

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

In [23]:
```python
# Concatenate all headlines into one sentence for each date; then apply

X_train_raw['all_headlines'] = X_train_raw.apply(lambda x: process_words(' '.join(x)), axis=1)
X_test_raw['all_headlines'] = X_test_raw.apply(lambda x: process_words(' '.join(x)), axis=1)
```

4. Feature Engineering

4a. n-gram CountVectorizer & TF-IDFVectorizer

`CountVectorizer` : Generate n-grams from world news headlines to use as features, naive method using *all* words.

`TFIDFVectorizer` : Generate n-grams from world news headlines to use as features, using TF-IDF as a means of dimensionality reduction (i.e. reducing the number of words that are vectorized to those with a higher TF-IDF score: words that are relatively more rare or more common, as compared to the rest of the words in each headline).

Experimental Setup

We will initially test each of the following 8 `vectorizer/n-gram` combinations, for each model.

Count Vectorizer	TF-IDF Vectorizer
1-grams	1-grams
2-grams	2-grams
1- & 2-grams	1- & 2-grams
1-, 2-, & 3-grams	1-, 2-, & 3-grams

As we can see below, TF-IDF vectorizer significantly reduces number of features.

# Features	Count Vectorizer	TF-IDF Vectorizer
1-grams	23,075	1,693
2-grams	333,650	119
1- & 2-grams	356,725	1,812
1-, 2-, & 3-grams	798,642	1,816

4b. Observation weighting by DailyReturn

For one more feature engineering idea, we weight observations based on the absolute value of the daily return, only for the train set. For example, if today's Daily Return is quite high or low, we will add a weight to today's feature vectors to emphasize the n-grams in today's headlines.

The idea is to create emphasis on certain n-grams associated with larger swings (up or down) in market price.

Note: we are avoiding data snooping since we only weight the observations in the train set, **NOT in the test set**. Additionally, we only weight by the absolute value of the DailyReturn, so we don't encode pos. or neg. value.

In [24]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Create 3 different CountVectorizers based on ngram_range:
# (1, 1): only 1-grams
# (2, 2): only 2-grams
# (1, 2): both 1-grams and 2-grams
# (1, 3): 1-, 2-, & 3-grams

ngrams_to_experiment = [(1, 1), (2, 2), (1, 2), (1, 3)]

count_vectorizer = {}
tfidf_vectorizer = {}
for ngram in ngrams_to_experiment:
    count_vectorizer[ngram] = CountVectorizer(analyzer='word', ngram_range=ngram)
    tfidf_vectorizer[ngram] = TfidfVectorizer(analyzer='word', ngram_range=ngram, min_df=0.03, max_df=0.95)

vectorizers = {'count':count_vectorizer, 'tfidf':tfidf_vectorizer}
```

In [25]:
```
from sklearn.preprocessing import StandardScaler

# Create 3 different X_train and X_test based on n-gram vectorizers:

scale_features = False

X_train, X_test = {}, {}
X_train_scaled, X_test_scaled = {}, {}
std_scaler = {}

for vect in ['count', 'tfidf']:
    for ngram in ngrams_to_experiment:
        X_train[(vect, ngram)] = vectorizers[vect][ngram].fit_transform(X_train_raw['all_headlines'])
        X_test[(vect, ngram)] = vectorizers[vect][ngram].transform(X_test_raw['all_headlines'])

        X_train[(vect, ngram)] = vectorizers[vect][ngram].fit_transform(X_train_raw['all_headlines'])
        X_test[(vect, ngram)] = vectorizers[vect][ngram].transform(X_test_raw['all_headlines'])

        # Weight each observation by the normalized daily return
        # e.g. on days with more extreme daily return movements, weight those news headlines more
#         X_train[(vect, ngram)] = X_train[(vect, ngram)].multiply(drn_weights_train.reshape(-1, 1))
#         X_test[(vect, ngram)] = X_test[(vect, ngram)].multiply(drn_weights_test.reshape(-1, 1))

        if scale_features:
            std_scaler[(vect, ngram)] = StandardScaler(with_mean=False) # for sparse matrices
            X_train_scaled[(vect, ngram)] = std_scaler[(vect, ngram)].fit_transform(X_train[(vect, ngram)])
            X_test_scaled[(vect, ngram)] = std_scaler[(vect, ngram)].transform(X_test[(vect, ngram)])
        else:
            X_train_scaled[(vect, ngram)] = X_train[(vect, ngram)]
            X_test_scaled[(vect, ngram)] = X_test[(vect, ngram)]
```

In [26]:
```
# Shows all feature names (1-grams and 2-grams)
# print(vectorizers[('count', (1, 2))].get_feature_names())
# print(vectorizers[('tfidf', (1, 2))].get_feature_names())
```

In [27]:
```
num_features = {}

for vect in ['count', 'tfidf']:
    for ngram in ngrams_to_experiment:
        num_features[vect+'|'+str(ngram)] = X_train[(vect, ngram)].shape[1] # store num features
        print('vectorizer:', vect, 'ngram_range:', ngram)
        print(X_train[(vect, ngram)].shape)
        print(X_test[(vect, ngram)].shape)
        print()
```

```
vectorizer: count ngram_range: (1, 1)
(1610, 23075)
(377, 23075)

vectorizer: count ngram_range: (2, 2)
(1610, 333650)
(377, 333650)

vectorizer: count ngram_range: (1, 2)
(1610, 356725)
(377, 356725)

vectorizer: count ngram_range: (1, 3)
(1610, 798642)
(377, 798642)

vectorizer: tfidf ngram_range: (1, 1)
(1610, 1693)
(377, 1693)

vectorizer: tfidf ngram_range: (2, 2)
(1610, 119)
(377, 119)

vectorizer: tfidf ngram_range: (1, 2)
(1610, 1812)
(377, 1812)

vectorizer: tfidf ngram_range: (1, 3)
(1610, 1816)
(377, 1816)
```

In [28]:
```python
# Define function to show top coefficients and ngram for each model
# to be used for diagnostic purposes later

def showTopNCoef(n, vect, model):
    coef_df = pd.DataFrame({
        'n-gram': vect.get_feature_names(),
        'coef': model.coef_[0]
    })

    return coef_df.sort_values(['coef', 'n-gram'], ascending=[False, True]).head(n), \
        coef_df.sort_values(['coef', 'n-gram'], ascending=[True, True]).head(n)
```

5. Model Training (for all 8 experimental setups)

First, we train models using default parameters for each classifier to obtain an initial indication of performance under various n-gram experiments. To limit the number of experiments we run, we want to evaluate which set of `n-grams/vectorizer` combinations appear to give our selected models the best performance, among the following experimental setups:

- Count Vectorizer
  - 1-grams
  - 2-grams
  - 1- & 2-grams
  - 1-, 2- & 3-grams
- TF-IDF Vectorizer
  - 1-grams
  - 2-grams
  - 1- & 2-grams
  - 1-, 2- & 3-grams

For each model, we choose the best experimental setup among the 8 above to further tune hyperparameters using cross-validation.

Models:

- Logistic Regression
- Linear Kernel SVC
- RBF Kernel SVC
- Random Forest Classifier
- K-Nearest Neighbors Classifier

In [29]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score

# Initialize models for each ngram experiment using default parameters

lr_clf = {}
svc_lin_clf = {}
svc_rbf_clf = {}
rf_clf = {}
knn_clf = {}

for vect in ['count','tfidf']:
    for ngram in ngrams_to_experiment:
        lr_clf[(vect, ngram)] = LogisticRegression(random_state=42, solver='lbfgs', max_iter=10000)
        svc_lin_clf[(vect, ngram)] = LinearSVC(random_state=42)
        svc_rbf_clf[(vect, ngram)] = SVC(kernel='rbf', probability=True, random_state=42)
        rf_clf[(vect, ngram)] = RandomForestClassifier(random_state=42)
        knn_clf[(vect, ngram)] = KNeighborsClassifier(n_neighbors=5)
```

In [30]:
```
# Train models based on n-gram experiments

clfs = [lr_clf, svc_lin_clf, svc_rbf_clf, rf_clf, knn_clf]

for vect in ['count','tfidf']:
    for ngram in ngrams_to_experiment:
        for estimator in clfs:
            print('Training:', vect, ngram, type(estimator[(vect, ngram)]).__name__)
            if type(estimator[(vect, ngram)]).__name__ == 'RandomForestClassifier':
                estimator[(vect, ngram)].fit(X_train[(vect, ngram)], y_train, sample_weight=drn_weights_tra
in)
            elif type(estimator[(vect, ngram)]).__name__ == 'KNeighborsClassifier':
                estimator[(vect, ngram)].fit(X_train_scaled[(vect, ngram)], y_train) #KNN doesn't take weig
hts
            else:
                estimator[(vect, ngram)].fit(X_train_scaled[(vect, ngram)], y_train, sample_weight=drn_weig
hts_train)
```

```
Training: count (1, 1) LogisticRegression
Training: count (1, 1) LinearSVC
Training: count (1, 1) SVC
Training: count (1, 1) RandomForestClassifier
Training: count (1, 1) KNeighborsClassifier
Training: count (2, 2) LogisticRegression
Training: count (2, 2) LinearSVC
Training: count (2, 2) SVC
Training: count (2, 2) RandomForestClassifier
Training: count (2, 2) KNeighborsClassifier
Training: count (1, 2) LogisticRegression
Training: count (1, 2) LinearSVC
Training: count (1, 2) SVC
Training: count (1, 2) RandomForestClassifier
Training: count (1, 2) KNeighborsClassifier
Training: count (1, 3) LogisticRegression
Training: count (1, 3) LinearSVC
Training: count (1, 3) SVC
Training: count (1, 3) RandomForestClassifier
Training: count (1, 3) KNeighborsClassifier
Training: tfidf (1, 1) LogisticRegression
Training: tfidf (1, 1) LinearSVC
Training: tfidf (1, 1) SVC
Training: tfidf (1, 1) RandomForestClassifier
Training: tfidf (1, 1) KNeighborsClassifier
Training: tfidf (2, 2) LogisticRegression
Training: tfidf (2, 2) LinearSVC
Training: tfidf (2, 2) SVC
Training: tfidf (2, 2) RandomForestClassifier
Training: tfidf (2, 2) KNeighborsClassifier
Training: tfidf (1, 2) LogisticRegression
Training: tfidf (1, 2) LinearSVC
Training: tfidf (1, 2) SVC
Training: tfidf (1, 2) RandomForestClassifier
Training: tfidf (1, 2) KNeighborsClassifier
Training: tfidf (1, 3) LogisticRegression
Training: tfidf (1, 3) LinearSVC
Training: tfidf (1, 3) SVC
Training: tfidf (1, 3) RandomForestClassifier
Training: tfidf (1, 3) KNeighborsClassifier
```

6. Evaluating Model Performance (for all initial experiments)

For this step, we evaluate model performance (with default model hyperparameters) using 5-fold Cross-Validated Train Set AUROC. We also report 5-fold Cross-Validated Train Set Accuracy for reference. We will then take the best experimental setup for each model and tune the hyperparameters in Section 7.

6a. Results

In [31]:

```python
# Define function for plotting results

# Attach a text label above each bar
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        plt.annotate(
            '{:.3f}'.format(height),
            xy=(rect.get_x() + rect.get_width() / 2, height),
            xytext=(0, 3),  # 3 points vertical offset
            textcoords="offset points",
            ha='center', va='bottom')

def plot_ngram_exp(metrics, metric_name, baseline):
    test_df = pd.DataFrame(metrics.values(), metrics.keys()).reset_index()
    test_df.columns = ['n-grams', 'Estimator', metric_name]

    N = len(test_df['n-grams'].unique())
    ind = np.arange(N)
    width = 0.18

    plt.figure(figsize=(12, 8))

    autolabel(plt.bar(ind, test_df[test_df['Estimator'] == 'LogisticRegression'][metric_name], width, label
='Logistic Regression'))
    autolabel(plt.bar(ind + width, test_df[test_df['Estimator'] == 'LinearSVC'][metric_name], width, label=
'SVC (Linear)'))
    autolabel(plt.bar(ind + width*2, test_df[test_df['Estimator'] == 'SVC'][metric_name], width, label='SVC
(RBF)'))
    autolabel(plt.bar(ind + width*3, test_df[test_df['Estimator'] == 'RandomForestClassifier'][metric_name
], width, label='Random Forest'))
    autolabel(plt.bar(ind + width*4, test_df[test_df['Estimator'] == 'KNeighborsClassifier'][metric_name],
width, label='KNN'))

    plt.title('Model '+metric_name+' by n-gram Feature Set')

    plt.grid(which='major', axis='y')
    plt.ylabel(metric_name)
    plt.ylim(0.2, 0.7)

    plt.hlines(y=baseline, xmin=0-width, xmax=N-width*0.75, colors='r', linestyles='--', lw=1)

    plt.xticks(ind + width*2, ('1-grams', '2-grams', '1- & 2-grams', '1-, 2-, & 3-grams'))
    plt.legend(loc='best')
    plt.show()
```

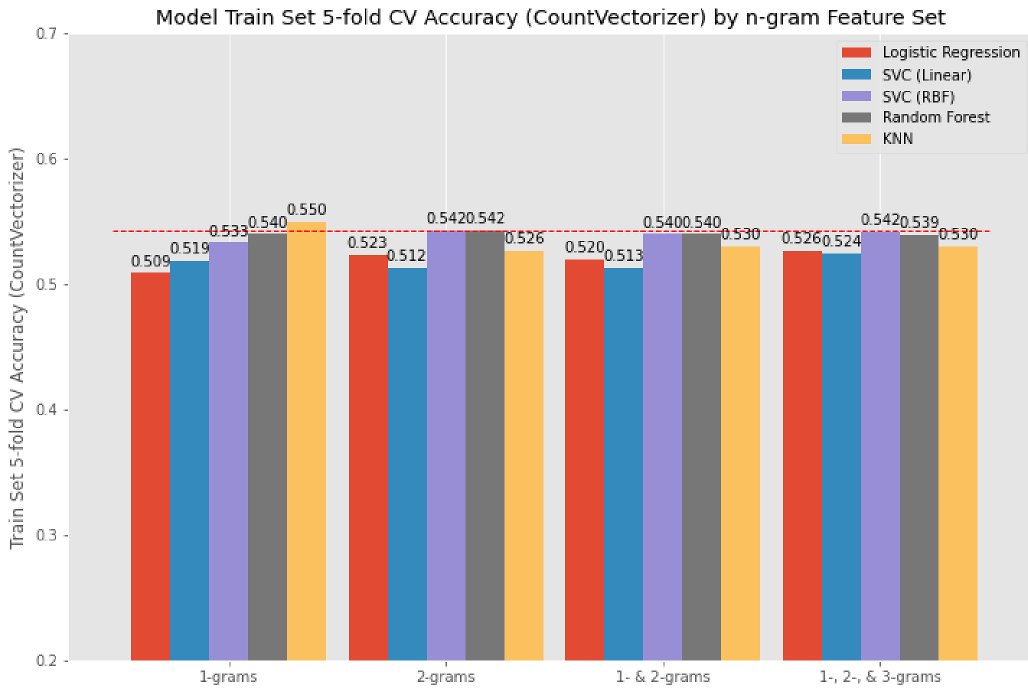
Train Set Performance

In [32]:
```
# Train Set Accuracy Scores, using 5-fold Cross Validation

from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

# two keys for y_pred dictionary: (ngram, estimator)
y_pred_train = {'count':{}, 'tfidf':{}}
acc_train = {'count':{}, 'tfidf':{}}

for vect in ['count','tfidf']:
    for ngram in ngrams_to_experiment:
        for estimator in clfs:
            estimator_name = type(estimator[(vect, ngram)]).__name__
            if estimator_name == 'RandomForestClassifier':
                y_pred_train[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_train[(vec
t, ngram)])
                acc_train[vect][(ngram, estimator_name)] = cross_val_score(
                    estimator[(vect, ngram)], X_train[(vect, ngram)], y_train, cv=5, n_jobs=-1).mean()
            else:
                y_pred_train[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_train_scal
ed[(vect, ngram)])
                acc_train[vect][(ngram, estimator_name)] = cross_val_score(
                    estimator[(vect, ngram)], X_train_scaled[(vect, ngram)], y_train, cv=5, n_jobs=-1).mean
()

            print(vect, ngram, estimator_name, acc_train[vect][(ngram, estimator_name)])
        print('---')
```

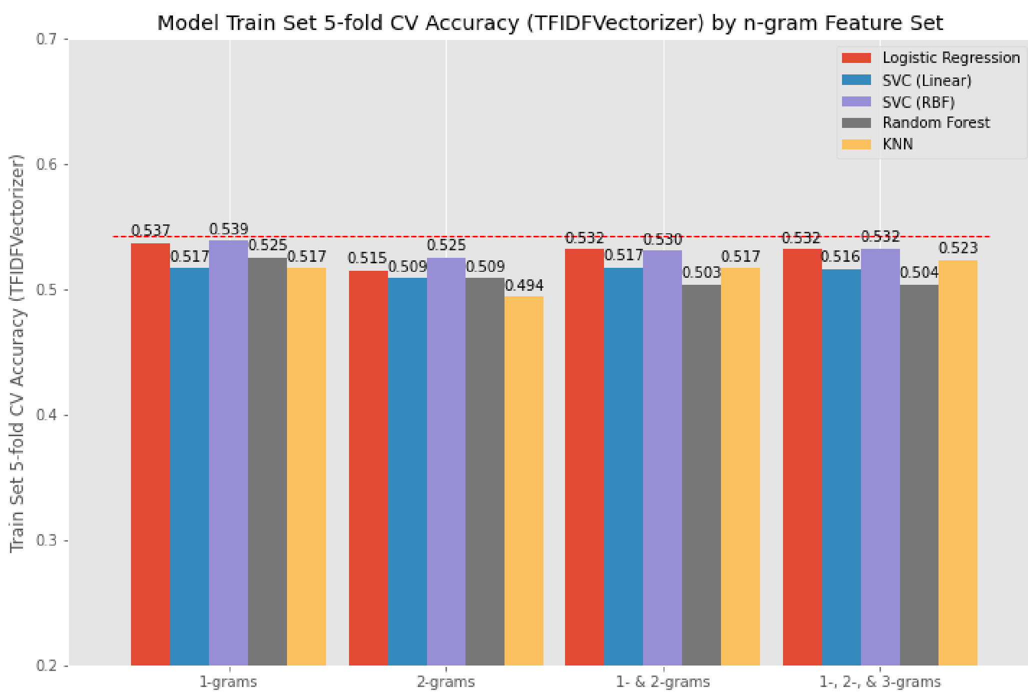
```
count (1, 1) LogisticRegression 0.508695652173913
count (1, 1) LinearSVC 0.5186335403726707
count (1, 1) SVC 0.532919254658385
count (1, 1) RandomForestClassifier 0.5397515527950311
count (1, 1) KNeighborsClassifier 0.5496894409937887
---
count (2, 2) LogisticRegression 0.5229813664596272
count (2, 2) LinearSVC 0.5124223602484472
count (2, 2) SVC 0.5422360248447206
count (2, 2) RandomForestClassifier 0.5422360248447206
count (2, 2) KNeighborsClassifier 0.5260869565217391
---
count (1, 2) LogisticRegression 0.5198757763975156
count (1, 2) LinearSVC 0.5130434782608696
count (1, 2) SVC 0.5397515527950312
count (1, 2) RandomForestClassifier 0.5397515527950311
count (1, 2) KNeighborsClassifier 0.5298136645962733
---
count (1, 3) LogisticRegression 0.5260869565217392
count (1, 3) LinearSVC 0.524223602484472
count (1, 3) SVC 0.5416149068322982
count (1, 3) RandomForestClassifier 0.5391304347826088
count (1, 3) KNeighborsClassifier 0.5298136645962733
---
tfidf (1, 1) LogisticRegression 0.5366459627329192
tfidf (1, 1) LinearSVC 0.5167701863354037
tfidf (1, 1) SVC 0.5385093167701863
tfidf (1, 1) RandomForestClassifier 0.5254658385093168
tfidf (1, 1) KNeighborsClassifier 0.5167701863354036
---
tfidf (2, 2) LogisticRegression 0.5149068322981367
tfidf (2, 2) LinearSVC 0.5086956521739131
tfidf (2, 2) SVC 0.5254658385093167
tfidf (2, 2) RandomForestClassifier 0.5086956521739131
tfidf (2, 2) KNeighborsClassifier 0.4937888198757764
---
tfidf (1, 2) LogisticRegression 0.5316770186335403
tfidf (1, 2) LinearSVC 0.5173913043478261
tfidf (1, 2) SVC 0.5304347826086957
tfidf (1, 2) RandomForestClassifier 0.5031055900621119
tfidf (1, 2) KNeighborsClassifier 0.5167701863354037
---
tfidf (1, 3) LogisticRegression 0.5316770186335403
tfidf (1, 3) LinearSVC 0.5161490683229814
tfidf (1, 3) SVC 0.5322981366459627
tfidf (1, 3) RandomForestClassifier 0.5037267080745342
tfidf (1, 3) KNeighborsClassifier 0.5229813664596273
---
```

Note: AUROC is the best metric for evaluating binary classification tasks, but we'll report accuracy just for reference.

In [33]:
```
plot_ngram_exp(acc_train['count'], 'Train Set 5-fold CV Accuracy (CountVectorizer)', baseline_acc_train)
```



In [34]:
```
plot_ngram_exp(acc_train['tfidf'], 'Train Set 5-fold CV Accuracy (TFIDFVectorizer)', baseline_acc_train)
```



In [35]:

```python
# Train Set AUROC Scores, using 5-fold Cross Validation

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score

# two keys for y_pred dictionary: (ngram, estimator)
y_pred_train = {'count':{}, 'tfidf':{}}
auroc_train = {'count':{}, 'tfidf':{}}

for vect in ['count','tfidf']:
    for ngram in ngrams_to_experiment:
        for estimator in clfs:
            estimator_name = type(estimator[(vect, ngram)]).__name__
            if estimator_name == 'RandomForestClassifier':
                y_pred_train[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_train[(vec
t, ngram)])
                auroc_train[vect][(ngram, estimator_name)] = cross_val_score(
                    estimator[(vect, ngram)], X_train[(vect, ngram)], y_train, cv=5, n_jobs=-1,
                    scoring='roc_auc').mean()
            else:
                y_pred_train[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_train_scal
ed[(vect, ngram)])
                auroc_train[vect][(ngram, estimator_name)] = cross_val_score(
                    estimator[(vect, ngram)], X_train_scaled[(vect, ngram)], y_train, cv=5, n_jobs=-1,
                    scoring='roc_auc').mean()

            print(vect, ngram, estimator_name, auroc_train[vect][(ngram, estimator_name)])
        print('---')
```

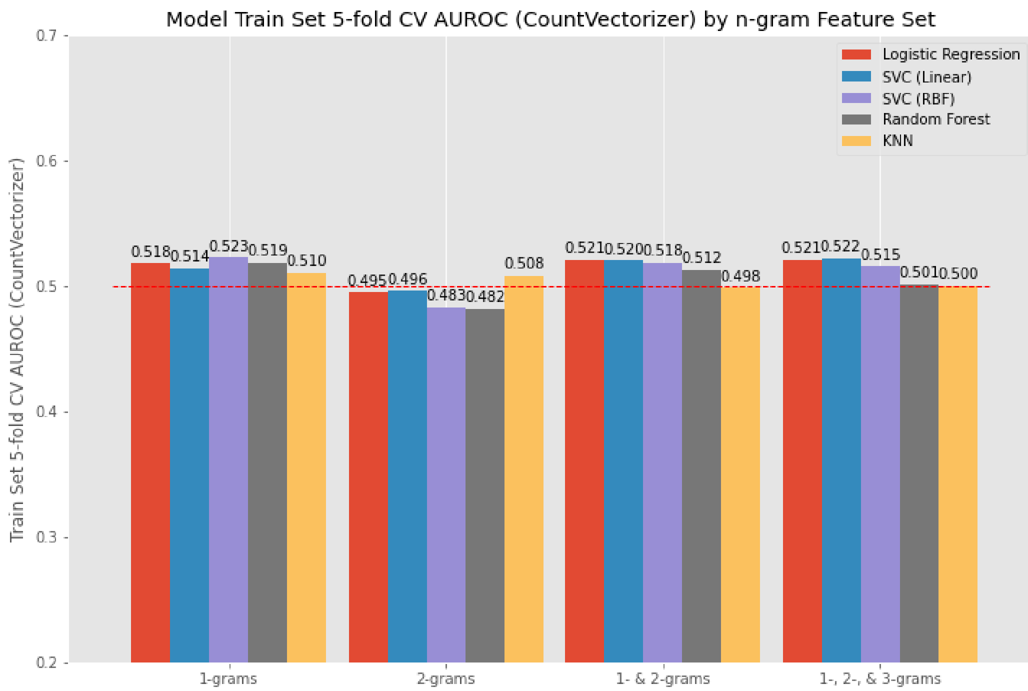
```
count (1, 1) LogisticRegression 0.5178655767459428
count (1, 1) LinearSVC 0.5139621308249035
count (1, 1) SVC 0.5225420440043945
count (1, 1) RandomForestClassifier 0.5186952963318199
count (1, 1) KNeighborsClassifier 0.5101901890832222
---
count (2, 2) LogisticRegression 0.4947205513546119
count (2, 2) LinearSVC 0.4959328658104168
count (2, 2) SVC 0.4827972280111618
count (2, 2) RandomForestClassifier 0.48168458415186066
count (2, 2) KNeighborsClassifier 0.5077796823162762
---
count (1, 2) LogisticRegression 0.5209237988365364
count (1, 2) LinearSVC 0.5201622445809637
count (1, 2) SVC 0.5184136581539819
count (1, 2) RandomForestClassifier 0.5122702046069394
count (1, 2) KNeighborsClassifier 0.4982970885331898
---
count (1, 3) LogisticRegression 0.5209554838844073
count (1, 3) LinearSVC 0.5220048835404218
count (1, 3) SVC 0.515450866167263
count (1, 3) RandomForestClassifier 0.5010326197314233
count (1, 3) KNeighborsClassifier 0.4996025253664239
---
tfidf (1, 1) LogisticRegression 0.5208174659026172
tfidf (1, 1) LinearSVC 0.5161655694097635
tfidf (1, 1) SVC 0.5206171456467023
tfidf (1, 1) RandomForestClassifier 0.522181470351421
tfidf (1, 1) KNeighborsClassifier 0.5166059466523928
---
tfidf (2, 2) LogisticRegression 0.4988421978830142
tfidf (2, 2) LinearSVC 0.5006385156645538
tfidf (2, 2) SVC 0.4979564710986245
tfidf (2, 2) RandomForestClassifier 0.492899447703459
tfidf (2, 2) KNeighborsClassifier 0.49574181223547864
---
tfidf (1, 2) LogisticRegression 0.5196913316251811
tfidf (1, 2) LinearSVC 0.5166482999537186
tfidf (1, 2) SVC 0.5198328791242233
tfidf (1, 2) RandomForestClassifier 0.47770070291463684
tfidf (1, 2) KNeighborsClassifier 0.5158258893758543
---
tfidf (1, 3) LogisticRegression 0.5195436969089327
tfidf (1, 3) LinearSVC 0.5165007630531909
tfidf (1, 3) SVC 0.5195451478421218
tfidf (1, 3) RandomForestClassifier 0.49371899988859874
tfidf (1, 3) KNeighborsClassifier 0.5224295070178251
---
```

AUROC is the best metric for evaluating binary classification.

`CountVectorizer`

- SVC (RBF kernel) had the best results with 1 grams (52.3%).
- SVC (Linear kernel) had good results for 1+2+3 grams (52.2%).
- Logistic Regression had good results for 1+2 grams and 1+2+3 grams (52.1%).
- Random Forest had decent results for 1 grams (51.9%).
- KNeighbors Classifier had decent results for 1 grams (51.0%).

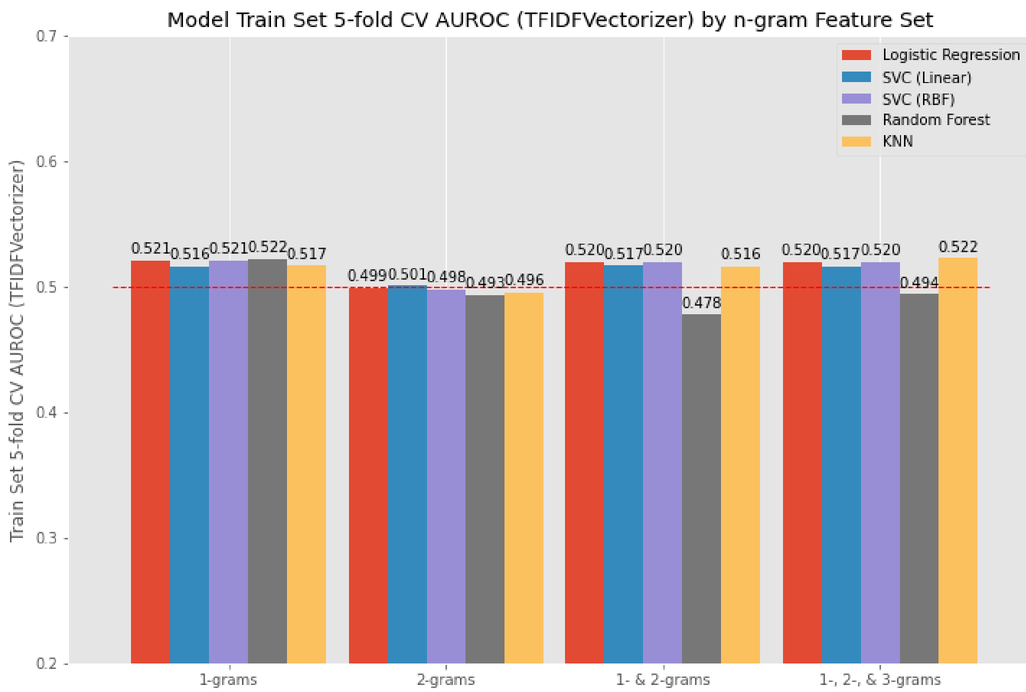
In [36]:
```
plot_ngram_exp(auroc_train['count'], 'Train Set 5-fold CV AUROC (CountVectorizer)', 0.5)
```



`TFIDFVectorizer`

- KNeighbors Classifier had the best results for 1+2+3 grams (52.2%).
- Random Forest tied for best results for 1 grams (52.2%).
- Logistic Regression had good results for 1 grams (52.1%); for 1+2 and 1+2+3 grams (52.0%).
- SVC (RBF kernel) had good results with 1 grams (52.1%); for 1+2 and 1+2+3 grams (52.0%).
- SVC (Linear kernel) had good results for 1+2 and 1+2+3 grams (51.7%).

```
In [37]: plot_ngram_exp(auroc_train['tfidf'], 'Train Set 5-fold CV AUROC (TFIDFVectorizer)', 0.5)
```

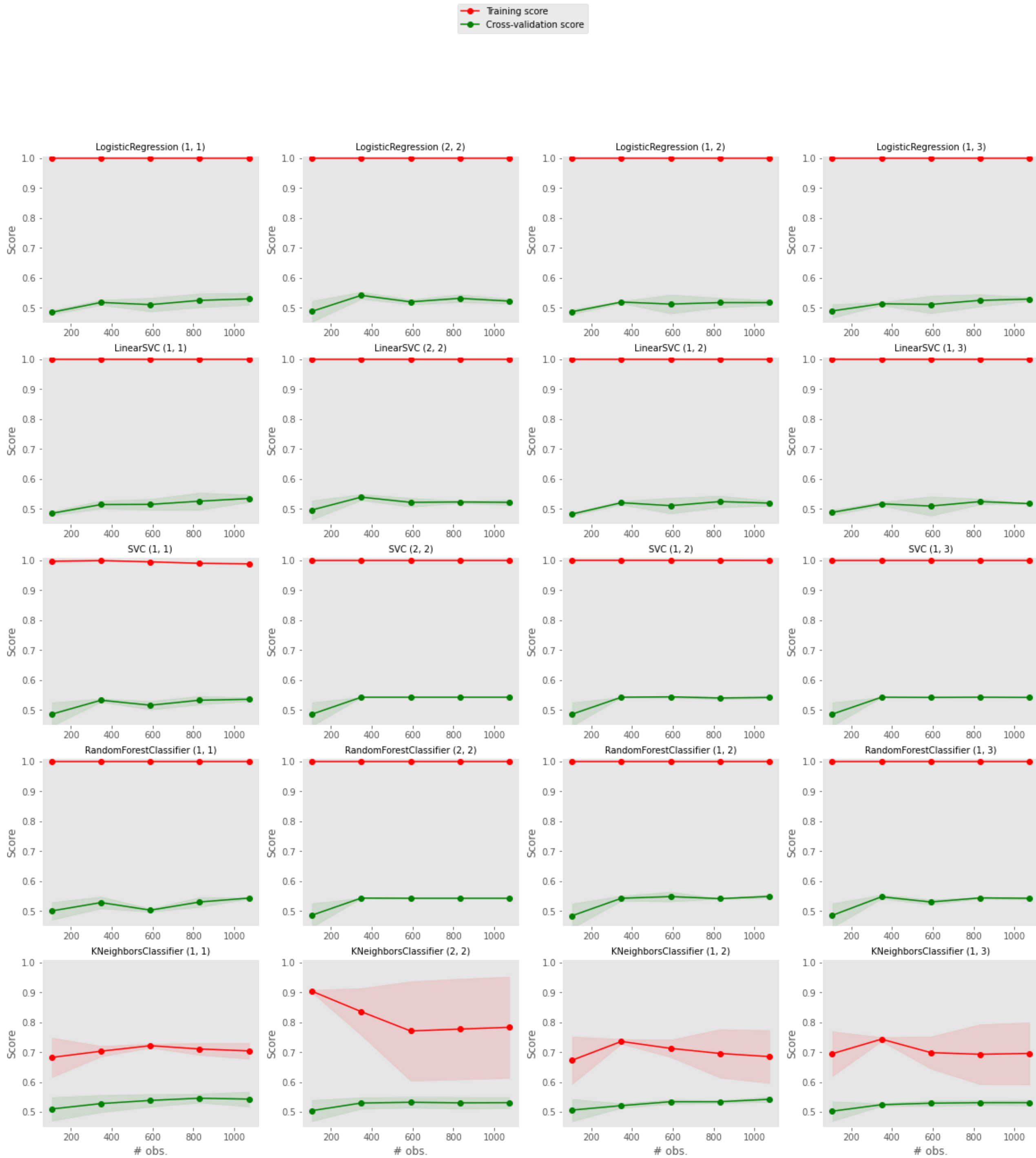


Learning Curves

```
In [38]: # Function for plotting learning curves given classifiers, vectorizer and ngrams experimental setup

from sklearn.model_selection import learning_curve

def plot_learning_curve_grid(clf, vect, ngrams, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5), xlabel=True):

    if axes is None:
        _, axes = plt.subplots(1, 4, figsize=(20, 10))

    c = 0
    for ngram in ngrams:
        axes[c].set_title(title+' '+str(ngram), fontsize=10)
        if ylim is not None:
            axes[c].set_ylim(*ylim)
        axes[c].set_ylabel("Score")
        if xlabel:
            axes[c].set_xlabel("# obs.")

        train_sizes, train_scores, test_scores, fit_times, _ = \
            learning_curve(clf[(vect, ngram)], X[(vect, ngram)], y, cv=cv, n_jobs=n_jobs,
                           train_sizes=train_sizes,
                           return_times=True)
        train_scores_mean = np.mean(train_scores, axis=1)
        train_scores_std = np.std(train_scores, axis=1)
        test_scores_mean = np.mean(test_scores, axis=1)
        test_scores_std = np.std(test_scores, axis=1)
        fit_times_mean = np.mean(fit_times, axis=1)
        fit_times_std = np.std(fit_times, axis=1)

        # Plot learning curve
        axes[c].grid()
        axes[c].fill_between(train_sizes, train_scores_mean - train_scores_std,
                             train_scores_mean + train_scores_std, alpha=0.1,
                             color="r")
        axes[c].fill_between(train_sizes, test_scores_mean - test_scores_std,
                             test_scores_mean + test_scores_std, alpha=0.1,
                             color="g")
        axes[c].plot(train_sizes, train_scores_mean, 'o-', color="r",
                     label="Training score")
        axes[c].plot(train_sizes, test_scores_mean, 'o-', color="g",
                     label="Cross-validation score")
        c += 1
```

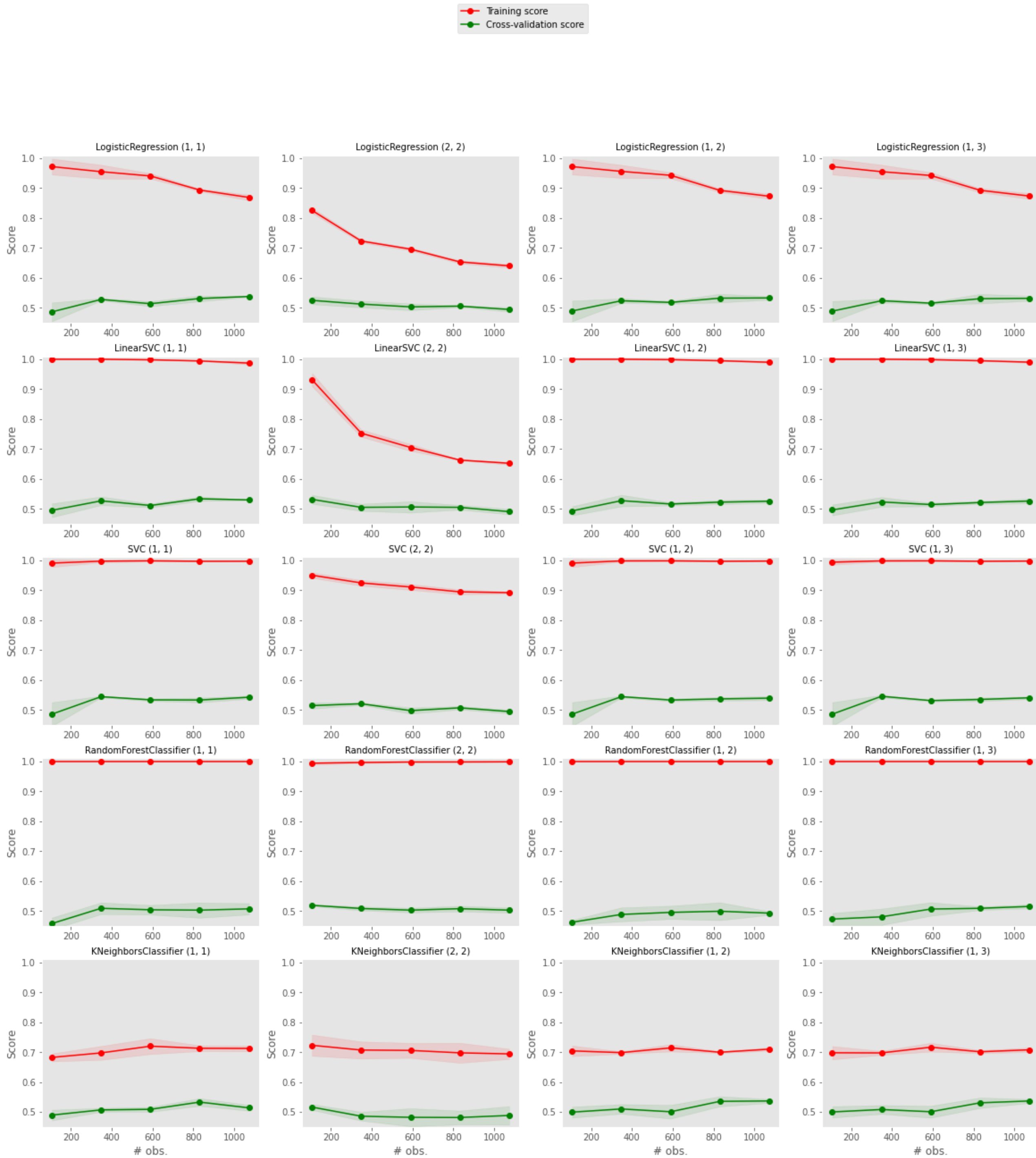
Learning Curves (CountVectorizer)

In [39]:
```
fig, axes = plt.subplots(5, 4, figsize=(20, 20))

c_lc = 0
vect_lc = 'count'
xlabel = False
for clf in clfs:
    estimator_name = type(clf[(vect, ngram)]).__name__
    if c_lc == len(clfs)-1:
        xlabel = True

    if estimator_name == 'RandomForestClassifier':
        plot_learning_curve_grid(clf, vect=vect_lc, ngrams=ngrams_to_experiment, title=str(estimator_name),
                                 X=X_train, y=y_train, axes=axes[c_lc, :],
                                 ylim=(0.45, 1.01),
                                 cv=3, n_jobs=-1, xlabel=xlabel)
    else:
        plot_learning_curve_grid(clf, vect=vect_lc, ngrams=ngrams_to_experiment, title=str(estimator_name),
                                 X=X_train_scaled, y=y_train, axes=axes[c_lc, :],
                                 ylim=(0.45, 1.01),
                                 cv=3, n_jobs=-1, xlabel=xlabel)
    c_lc += 1

handles, labels = axes[0, 0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center')
plt.show()
```



Learning Curves (TFIDF Vectorizer)

In [40]:
```
fig, axes = plt.subplots(5, 4, figsize=(20, 20))

c_lc = 0
vect_lc = 'tfidf'
xlabel = False
for clf in clfs:
    estimator_name = type(clf[(vect, ngram)]).__name__
    if c_lc == len(clfs)-1:
        xlabel = True

    if estimator_name == 'RandomForestClassifier':
        plot_learning_curve_grid(clf, vect=vect_lc, ngrams=ngrams_to_experiment, title=str(estimator_name),
                                X=X_train, y=y_train, axes=axes[c_lc, :],
                                ylim=(0.45, 1.01),
                                cv=3, n_jobs=-1, xlabel=xlabel)
    else:
        plot_learning_curve_grid(clf, vect=vect_lc, ngrams=ngrams_to_experiment, title=str(estimator_name),
                                X=X_train_scaled, y=y_train, axes=axes[c_lc, :],
                                ylim=(0.45, 1.01),
                                cv=3, n_jobs=-1, xlabel=xlabel)
    c_lc += 1

handles, labels = axes[0, 0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center')
plt.show()
```



Learning Curves, Summary of Results:

- Most models appear to be overfit (some more than others), as there is a large gap between train accuracy (red line) and cross-validation accuracy (green line).
- Unsurprisingly, TFIDF vectorizer appears to lessen the overfitting, as it essentially acts as a dimensionality reduction technique.

Test Set Performance

Note: We decided not to show the below scores during this phase (training models with default hyperparameters) and instead will evaluate using Cross-validated Train Set Accuracy and AUROC. However the code is here if you're interested.

In [41]:
```
# Note: We decided not to show the below scores during this phase (training models with default hyperparame
ters)
# and instead will evaluate using Cross-validated Train Set Accuracy and AUROC.

# Test Set Accuracy Scores

# y_pred_test = {'count':{}, 'tfidf':{}}
# y_prob_test = {'count':{}, 'tfidf':{}} # for computing AUROC
# acc_test = {'count':{}, 'tfidf':{}}

# for vect in ['count','tfidf']:
#     for ngram in ngrams_to_experiment:
#         for estimator in clfs:
#             estimator_name = type(estimator[(vect, ngram)]).__name__
#             if estimator_name == 'RandomForestClassifier':
#                 y_pred_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_test[(vec
t, ngram)])
#                 y_prob_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict_proba(X_tes
t[(vect, ngram)])
#             elif estimator_name == 'LinearSVC':
#                 y_pred_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_test_scal
ed[(vect, ngram)])
#                 y_prob_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].decision_function(X
_test_scaled[(vect, ngram)])
#             else:
#                 y_pred_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict(X_test_scal
ed[(vect, ngram)])
#                 y_prob_test[vect][(ngram, estimator_name)] = estimator[(vect, ngram)].predict_proba(X_tes
t_scaled[(vect, ngram)])

#             acc_test[vect][(ngram, estimator_name)] = accuracy_score(y_test, y_pred_test[vect][(ngram, es
timator_name)])
#             print(vect, ngram, estimator_name, acc_test[vect][(ngram, estimator_name)])
#         print('---')
```

In [42]:
```
# plot_ngram_exp(acc_test['count'], 'Test Set Accuracy (CountVectorizer)', baseline_acc_test)
```

In [43]:
```
# plot_ngram_exp(acc_test['tfidf'], 'Test Set Accuracy (TfidfVectorizer)', baseline_acc_test)
```

In [44]:
```
# Note: We decided not to show the below scores during this phase (training models with default hyperparame
ters)
# and instead will evaluate using Cross-validated Train Set Accuracy and AUROC.

# Test Set AUROC Scores
# auroc_test = {'count':{}, 'tfidf':{}}

# for vect in ['count','tfidf']:
#     for ngram in ngrams_to_experiment:
#         for estimator in clfs:
#             estimator_name = type(estimator[(vect, ngram)]).__name__
#             if estimator_name == 'LinearSVC':
#                 auroc_test[vect][(ngram, estimator_name)] = roc_auc_score(y_test, y_prob_test[vect][(ngra
m, estimator_name)])
#             else:
#                 auroc_test[vect][(ngram, estimator_name)] = roc_auc_score(y_test, y_prob_test[vect][(ngra
m, estimator_name)][:, 1])
#             print(vect, ngram, estimator_name, auroc_test[vect][(ngram, estimator_name)])
#         print('---')
```

In [45]:
```
# plot_ngram_exp(auroc_test['count'], 'Test Set AUROC (CountVectorizer)', 0.5)
```

In [46]:
```
# plot_ngram_exp(auroc_test['tfidf'], 'Test Set AUROC (TfidfVectorizer)', 0.5)
```

6b. Discussion

Summary of results for initial run with default model hyperparameters:

- Overall, performance amongst the models for each model's best experimental setup was fairly similar (around 52.0-52.3% Train CV AUROC). However, interestingly, each model's best experimental setup differed. It was somewhat reassuring that each model achieved roughly similar results using their best experimental setup. It also gives us some indication that the breadth of the 8 experimental setups appeared to have been broad enough to capture the "sweet spots" of each model.
- SVC RBF had the best results, Count 1 gram, at 52.3% Train CV AUROC.
  - TFIDF 1+2 and 1+2+3 gram performed well also.
  - Both SVC models appeared to work best with high dimensional featues (Count vectorizer), supporting the claim that they handle p >> n situation well.
- SVC Linear had the best results with Count 1+2+3 grams at 52.2%.
- KNN Classifier (1+2+3 grams; 52.2%) and Random Forest (1 grams; 52.2%) both performed better with TFIDF vectorizer, suggesting they work better with fewer features.
- Logistic Regression performed well in both Count and TFIDF vectorizers (52.1%).
- Some models were more prone to overfitting than others (see learning curves)
  - Prone to overfitting:
    - Logistic Regression
    - Linear SVC
    - Random Forest Classifier
  - Not as prone to overfitting:
    - KNN Classifier
    - SVC (RBF kernel)

Other comments:

Many of these models under all of the n-gram scenarios appear to be overfit. The evidence is in the large cap between train metrics and validation metrics in the learning curves.

This is an unsurpising effect of using high-dimensional data, especially where `p >> n`. Without explicitly specifying high regularization parameters, these models are likely "memorizing" aspects of the training data to achieve high train set performance.

While we've been taught that overfitting is generally bad as it increases model variance (makes performance on test set data poor and unreliable), it does bring up an interesting question: although the performance on test set may be unstable, we can theoretically argue that the fact that we are seeing some Train CV AUROC scores over 50% is promising. In very difficult problem with arguably poor signal, we are able to extract at least some signal out of the predictors.

In section 7 below, we'll tune the hyperparameters for each model under each model's best experimental setup.

Top Pos. and Neg. Coefficients for Linear Models

Let's look at some examples of most positive and most negative features.

Generally the negative features seem clearly correct, whereas the positive features are more mixed.

`LogisticRegression` : `CountVectorizer, 2-grams`

In [47]:
```
showTopNCoef(10, vectorizers['count'][(2, 2)], lr_clf[('count', (2, 2))])
```

Out[47]:
```
(                  n-gram      coef
 68371         court rule  0.246405
 110556        first time  0.232139
 324727       white house  0.211042
 198616       new zealand  0.200448
 262564  security council  0.198643
 293278          tear gas  0.194958
 273939      social medium  0.179430
 297963        three year  0.176776
 134914        high court  0.166944
 275830       south korea  0.165744,
                 n-gram      coef
 201860  nuclear weapon -0.240676
 17780     around world -0.226412
 275831    south korean -0.201072
 201793   nuclear plant -0.194708
 318873     wall street -0.193018
 31933        bin laden -0.189222
 153697      israel use -0.183565
 218141      phone hack -0.180755
 266703    sexual abuse -0.177923
 286134  suicide bomber -0.170099)
```

LogisticRegression : TFIDFVectorizer, 2-grams

In [48]:
```
showTopNCoef(10, vectorizers['tfidf'][(2, 2)], lr_clf[('tfidf', (2, 2))])
```

Out[48]:
```
(             n-gram      coef
 106     white house  1.309646
 12       court rule  1.124369
 79      right watch  1.118909
 24        five year  1.031474
 21   european union  1.016826
 78      right group  1.004822
 34         iraq war  0.951807
 16     drone strike  0.939294
 68      people kill  0.936983
 14    death penalty  0.920368,
             n-gram      coef
 61  nuclear reactor -1.166730
 92   suicide bomber -1.097430
 69       phone hack -1.087873
 7    billion dollar -1.033564
 4      around world -1.031501
 9   catholic church -1.030897
 62   nuclear weapon -1.021356
 40       kill least -0.966799
 87     south africa -0.954111
 70       pirate bay -0.844445)
```

LinearSVC : CountVectorizer, 2-grams

In [49]:
```
showTopNCoef(10, vectorizers['count'][(2, 2)], svc_lin_clf[('count', (2, 2))])
```

Out[49]:
```
(                  n-gram      coef
 110556        first time  0.058029
 68371         court rule  0.054839
 198616       new zealand  0.053964
 324727       white house  0.053262
 297963        three year  0.049854
 329014       world large  0.049820
 262564  security council  0.049131
 293278          tear gas  0.046318
 222403     police arrest  0.045659
 273939     social medium  0.045582,
                 n-gram      coef
 201860  nuclear weapon -0.064863
 17780     around world -0.061629
 266703    sexual abuse -0.056593
 275831    south korean -0.052397
 318873     wall street -0.050911
 31933        bin laden -0.048331
 201793   nuclear plant -0.047285
 323958     west africa -0.043480
 218141      phone hack -0.043347
 153697      israel use -0.041140)
```

LinearSVC : CountVectorizer, 2-grams

In [50]:
```
showTopNCoef(10, vectorizers['tfidf'][(2, 2)], svc_lin_clf[('tfidf', (2, 2))])
```

Out[50]:
```
(           n-gram      coef
 106   white house  1.074155
 79    right watch  0.865734
 43     last month  0.850672
 68    people kill  0.827269
 67    peace prize  0.751476
 12     court rule  0.741853
 34       iraq war  0.732812
 78    right group  0.708147
 24      five year  0.702142
 50   minister say  0.686600,
               n-gram      coef
 69       phone hack -0.832975
 61  nuclear reactor -0.819433
 9   catholic church -0.784030
 4      around world -0.751599
 40       kill least -0.748943
 7    billion dollar -0.746520
 92   suicide bomber -0.714556
 62   nuclear weapon -0.654195
 70       pirate bay -0.602958
 95     ten thousand -0.585483)
```

7. Tuning Hyperparameters via GridSearchCV

For each model, we select the experiment setup (`vectorizer/n-gram` combination) that worked the best above. Then we tune hyperparameters with cross-validation for the model. Finally we evaluate each model using the test set, plot learning curves, ROC curves, and confusion matrix.

In [51]:
```
from sklearn.model_selection import learning_curve

def plot_learning_curve(clf, vect, ngrams, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    plt.title(title+' '+str(vect)+' '+str(ngrams), fontsize=10)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.ylabel("Score")
    plt.xlabel("# obs.")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(clf, X[(vect, ngram)], y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
```

7a. LogisticRegression

In [52]:
```
best_vectorizer = 'count'
best_ngram = (2, 2)
```

In [53]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

lr_clf_init = LogisticRegression(random_state=42)

lr_param_grid = [{
        'penalty': ['l2', 'l1'],
        'solver': ['liblinear'],
        'C': [10**c for c in np.arange(1, -6, -1, dtype=float)]
    }]

lr_clf_gscv = GridSearchCV(lr_clf_init, lr_param_grid, cv=3, n_jobs=-1, scoring='roc_auc')
lr_clf_gscv.fit(X_train_scaled[(best_vectorizer, best_ngram)], y_train, sample_weight=drn_weights_train)
```

Out[53]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(random_state=42), n_jobs=-1,
             param_grid=[{'C': [10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05],
                          'penalty': ['l2', 'l1'], 'solver': ['liblinear']}],
             scoring='roc_auc')
```

In [54]:
```
from sklearn.model_selection import cross_val_score

print('Best hyperparameters:', lr_clf_gscv.best_params_)

lr_clf_gscv_cv_score = cross_val_score(
    lr_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1)
print('5-fold Cross Validation Train Accuracy', lr_clf_gscv_cv_score.mean())

lr_clf_gscv_cv_score_auroc = cross_val_score(
    lr_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1,
    scoring='roc_auc')
print('5-fold Cross Validation Train AUROC', lr_clf_gscv_cv_score_auroc.mean())
```

```
Best hyperparameters: {'C': 1e-05, 'penalty': 'l2', 'solver': 'liblinear'}
5-fold Cross Validation Train Accuracy 0.5422360248447206
5-fold Cross Validation Train AUROC 0.5125185283805973
```

Note: hyperparameter tuning didn't appear to help here (model with default parameters performed better), so we'll use the model with default hyperparameters going forward.

In [55]:
```
best_lr_clf = clfs[0][(best_vectorizer, best_ngram)]
# best_lr_clf = lr_clf_gscv.best_estimator_
```

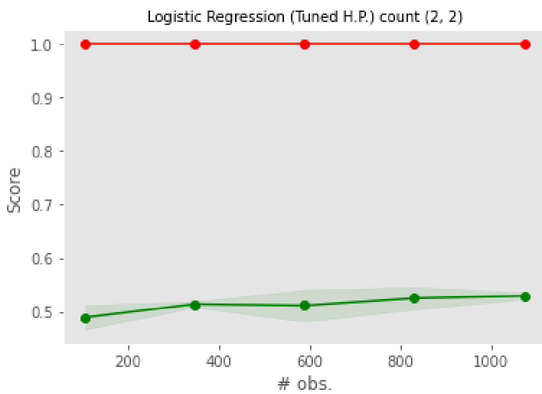
In [56]:
```
y_pred_lr_clf_gscv_best = best_lr_clf.predict(X_test_scaled[(best_vectorizer, best_ngram)])
y_prob_lr_clf_gscv_best = best_lr_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])[:, 1]

print('Test Accuracy:', best_lr_clf.score(X_test_scaled[(best_vectorizer, best_ngram)], y_test))

roc_auc = roc_auc_score(y_test, y_prob_lr_clf_gscv_best)
print('Test AUROC:', roc_auc)
```

```
Test Accuracy: 0.5119363395225465
Test AUROC: 0.5464167088892642
```

Learning Curve

Not as overfit as before.

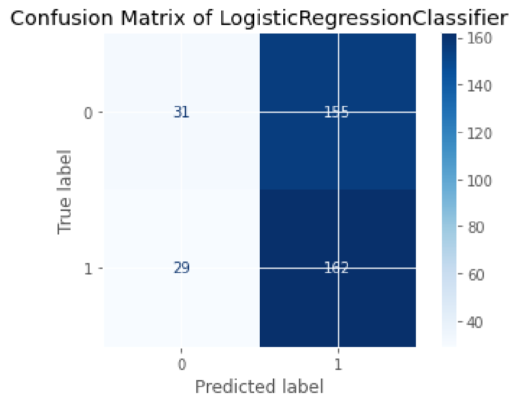
In [57]:
```
plot_learning_curve(best_lr_clf, vect=best_vectorizer, ngrams=best_ngram,
                    title='Logistic Regression (Tuned H.P.)',
                    X=X_train_scaled, y=y_train,
                    cv=3, n_jobs=-1)
```



Confusion Matrix

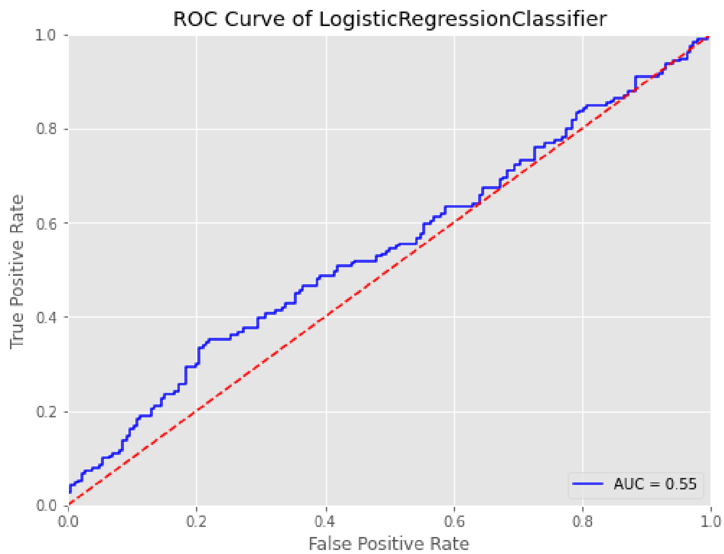
In [58]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(best_lr_clf, X_test_scaled[(best_vectorizer, best_ngram)], y_test,
                             display_labels=[0, 1],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title("Confusion Matrix of LogisticRegressionClassifier")

print("Confusion Matrix of LogisticRegressionClassifier")
print(disp.confusion_matrix)

plt.show()
```

```
Confusion Matrix of LogisticRegressionClassifier
[[ 31 155]
 [ 29 162]]
```



ROC Curve

In [59]:
```python
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

y_scores_lr_best = best_lr_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])
fpr, tpr, threshold = roc_curve(y_test, y_scores_lr_best[:, 1])

plt.figure(figsize=(8, 6))
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of LogisticRegressionClassifier')
plt.show()
```



AUROC over 50%! Good result using default hyperparameters.

Top Pos. and Neg. n-grams:

In [60]:
```python
showTopNCoef(10, vectorizers[best_vectorizer][best_ngram], lr_clf_gscv.best_estimator_)
```

Out[60]:
```
(                n-gram      coef
 200576       north korea  0.000142
 68371        court rule  0.000136
 158197    julian assange  0.000125
 110556        first time  0.000125
 228967    prime minister  0.000124
 198616       new zealand  0.000124
 108704  financial crisis  0.000115
 262564  security council  0.000107
 311163      united state  0.000106
 139418       human right  0.000104,
                n-gram      coef
 31933         bin laden -0.000113
 8574            al qaeda -0.000094
 241978        red cross -0.000087
 201793    nuclear plant -0.000081
 93463     embassy yemen -0.000080
 329107      world order -0.000078
 286134   suicide bomber -0.000076
 14488       anti terror -0.000076
 168217    leave country -0.000075
 140653   icelandic bank -0.000073)
```

7b. LinearSVC

In [61]:
```
best_vectorizer = 'count'
best_ngram = (2, 2)
```

In [62]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC

svc_lin_clf_init = LinearSVC(random_state=42)

svc_lin_param_grid = [
    {
        'penalty': ['l1'],
        'loss': ['squared_hinge'],
        'dual': [False],
        'max_iter': [10000],
        'C': [10**c for c in np.arange(1, -8, -1, dtype=float)]
    },
    {
        'penalty': ['l2'],
        'loss': ['squared_hinge'],
        'C': [10**c for c in np.arange(1, -8, -1, dtype=float)]
    }
]

svc_lin_clf_gscv = GridSearchCV(svc_lin_clf_init, svc_lin_param_grid, cv=3, n_jobs=-1, scoring='roc_auc')
svc_lin_clf_gscv.fit(X_train_scaled[(best_vectorizer, best_ngram)], y_train, sample_weight=drn_weights_trai
n)
```

Out[62]:
```
GridSearchCV(cv=3, estimator=LinearSVC(random_state=42), n_jobs=-1,
             param_grid=[{'C': [10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05,
                                1e-06, 1e-07],
                          'dual': [False], 'loss': ['squared_hinge'],
                          'max_iter': [10000], 'penalty': ['l1']},
                         {'C': [10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05,
                                1e-06, 1e-07],
                          'loss': ['squared_hinge'], 'penalty': ['l2']}],
             scoring='roc_auc')
```

In [63]:
```
from sklearn.model_selection import cross_val_score

print('Best hyperparameters:', svc_lin_clf_gscv.best_params_)

svc_lin_clf_gscv_cv_score = cross_val_score(
    svc_lin_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=
-1)
print('5-fold Cross Validation Train Accuracy', svc_lin_clf_gscv_cv_score.mean())

svc_lin_clf_gscv_cv_score_auroc = cross_val_score(
    svc_lin_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=
-1,
    scoring='roc_auc')
print('5-fold Cross Validation Train AUROC', svc_lin_clf_gscv_cv_score_auroc.mean())
```

```
Best hyperparameters: {'C': 1e-07, 'loss': 'squared_hinge', 'penalty': 'l2'}
5-fold Cross Validation Train Accuracy 0.5422360248447206
5-fold Cross Validation Train AUROC 0.512526302918984
```

Note: hyperparameter tuning didn't appear to help here (model with default parameters performed better), so we'll use the model with default hyperparameters going forward.

In [64]:
```
best_svc_lin_clf = clfs[1][(best_vectorizer, best_ngram)]
# best_svc_lin_clf = lr_clf_gscv.best_estimator_
```

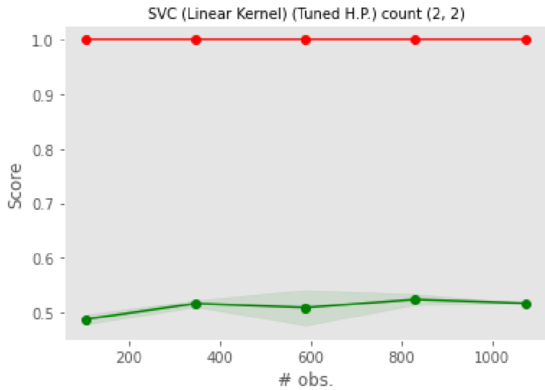
In [65]:
```
y_pred_svc_lin_clf_gscv_best = best_svc_lin_clf.predict(X_test_scaled[(best_vectorizer, best_ngram)])
y_prob_svc_lin_clf_gscv_best = best_svc_lin_clf.decision_function(X_test_scaled[(best_vectorizer, best_ngra
m)])

print('Test Accuracy:', best_svc_lin_clf.score(X_test_scaled[(best_vectorizer, best_ngram)], y_test))

roc_auc = roc_auc_score(y_test, y_prob_svc_lin_clf_gscv_best)
print('Test AUROC:', roc_auc)
```

```
Test Accuracy: 0.5092838196286472
Test AUROC: 0.5390981253166695
```

Learning Curve

Not as overfit as before.

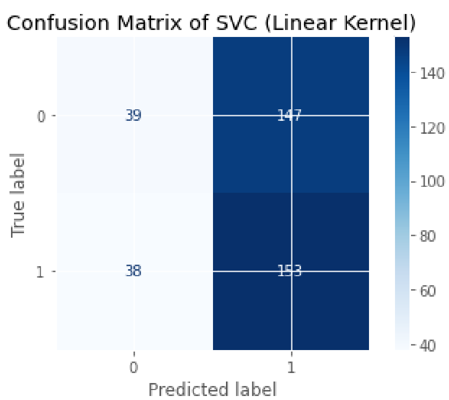
In [66]:
```
plot_learning_curve(best_svc_lin_clf, vect=best_vectorizer, ngrams=best_ngram,
                    title='SVC (Linear Kernel) (Tuned H.P.)',
                    X=X_train_scaled, y=y_train,
                    cv=3, n_jobs=-1)
```



Confusion Matrix

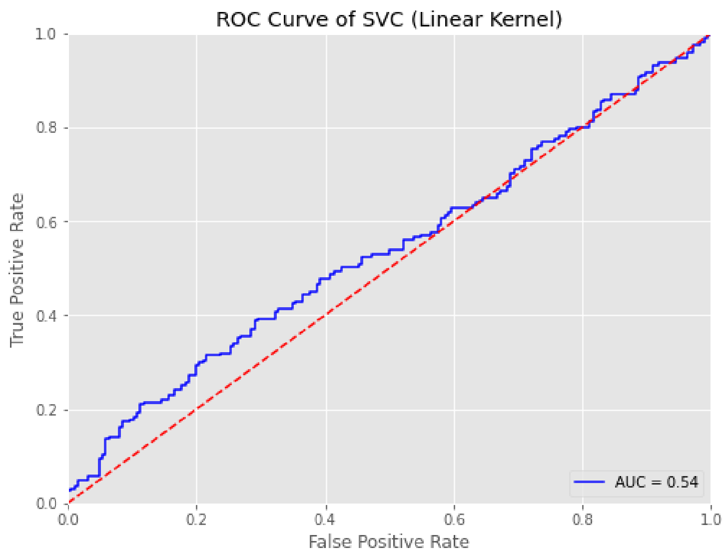
In [67]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(best_svc_lin_clf, X_test_scaled[(best_vectorizer, best_ngram)], y_test,
                             display_labels=[0, 1],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title("Confusion Matrix of SVC (Linear Kernel)")

print("Confusion Matrix of SVC (Linear Kernel)")
print(disp.confusion_matrix)

plt.show()
```

```
Confusion Matrix of SVC (Linear Kernel)
[[ 39 147]
 [ 38 153]]
```



ROC Curve

In [68]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

y_scores_svc_lin_best = best_svc_lin_clf.decision_function(X_test_scaled[(best_vectorizer, best_ngram)])
fpr, tpr, threshold = roc_curve(y_test, y_scores_svc_lin_best)

plt.figure(figsize=(8, 6))
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of SVC (Linear Kernel)')
plt.show()
```



AUROC over 50%! Good result using default hyperparameters.

Top Pos. and Neg. n-grams:

In [69]:
```
showTopNCoef(10, vectorizers[best_vectorizer][best_ngram], svc_lin_clf_gscv.best_estimator_)
```

Out[69]:
```
(                 n-gram      coef
 200576       north korea  0.000006
 68371         court rule  0.000005
 110556        first time  0.000005
 158197    julian assange  0.000005
 228967    prime minister  0.000005
 198616        new zealand  0.000005
 108704   financial crisis  0.000005
 262564  security council  0.000004
 311163       united state  0.000004
 139418        human right  0.000004,
                 n-gram      coef
 31933          bin laden -0.000005
 8574            al qaeda -0.000004
 241978         red cross -0.000003
 201793     nuclear plant -0.000003
 93463      embassy yemen -0.000003
 329107       world order -0.000003
 286134    suicide bomber -0.000003
 14488        anti terror -0.000003
 168217     leave country -0.000003
 140653    icelandic bank -0.000003)
```

7c. SVC (RBF)

```
In [70]: best_vectorizer = 'count'
         best_ngram = (1, 1)
```

```
In [71]: from sklearn.model_selection import GridSearchCV
         from sklearn.svm import SVC

         svc_rbf_clf_init = SVC(kernel='rbf', random_state=42, probability=True)

         svc_rbf_param_grid = [
             {
                 'degree': [3, 4, 5],
                 'gamma': ['scale'],
                 'C': [10**c for c in np.arange(1, -8, -1, dtype=float)]
             }
         ]

         svc_rbf_clf_gscv = GridSearchCV(svc_rbf_clf_init, svc_rbf_param_grid, cv=3, n_jobs=-1, scoring='roc_auc')
         svc_rbf_clf_gscv.fit(X_train_scaled[(best_vectorizer, best_ngram)], y_train, sample_weight=drn_weights_trai
         n)
```

```
Out[71]: GridSearchCV(cv=3, estimator=SVC(probability=True, random_state=42), n_jobs=-1,
                      param_grid=[{'C': [10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05,
                                         1e-06, 1e-07],
                                   'degree': [3, 4, 5], 'gamma': ['scale']}],
                      scoring='roc_auc')
```

```
In [72]: from sklearn.model_selection import cross_val_score

         print('Best hyperparameters:', svc_rbf_clf_gscv.best_params_)

         svc_rbf_clf_gscv_cv_score = cross_val_score(
             svc_rbf_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=
         -1)
         print('5-fold Cross Validation Train Accuracy', svc_rbf_clf_gscv_cv_score.mean())

         svc_rbf_clf_gscv_cv_score_auroc = cross_val_score(
             svc_rbf_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=
         -1,
             scoring='roc_auc')
         print('5-fold Cross Validation Train AUROC', svc_rbf_clf_gscv_cv_score_auroc.mean())
```

```
Best hyperparameters: {'C': 10.0, 'degree': 3, 'gamma': 'scale'}
5-fold Cross Validation Train Accuracy 0.5180124223602485
5-fold Cross Validation Train AUROC 0.526265787049742
```

Note: hyperparameter tuning didn't appear to help here (model with default parameters performed better), so we'll use the model with default hyperparameters going forward.

```
In [73]: best_svc_rbf_clf = clfs[2][(best_vectorizer, best_ngram)]
         # best_svc_rbf_clf = lr_clf_gscv.best_estimator_
```

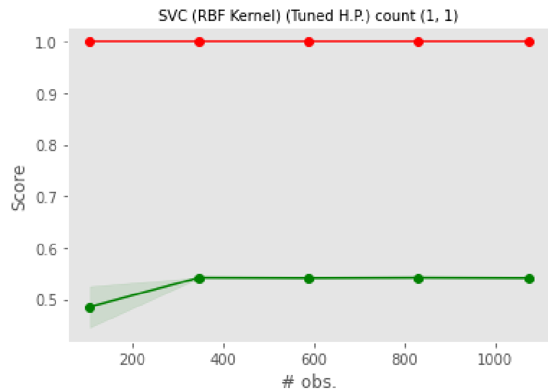
```
In [74]: y_pred_svc_rbf_clf_gscv_best = best_svc_rbf_clf.predict(X_test_scaled[(best_vectorizer, best_ngram)])
         y_prob_svc_rbf_clf_gscv_best = best_svc_rbf_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])
         [:, 1]

         print('Test Accuracy:', best_svc_rbf_clf.score(X_test_scaled[(best_vectorizer, best_ngram)], y_test))

         roc_auc = roc_auc_score(y_test, y_prob_svc_rbf_clf_gscv_best)
         print('Test AUROC:', roc_auc)
```

```
Test Accuracy: 0.5039787798408488
Test AUROC: 0.5070933963857456
```

Learning Curve

Still appears to be somewhat overfit.

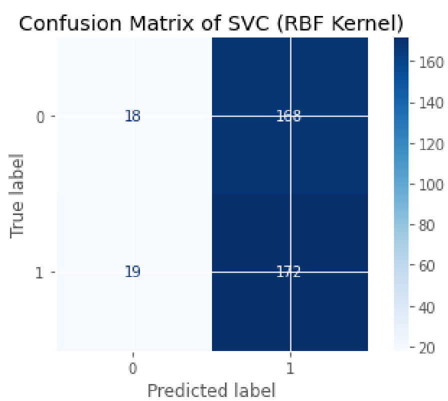
In [75]:
```
plot_learning_curve(best_svc_rbf_clf, vect=best_vectorizer, ngrams=best_ngram,
                    title='SVC (RBF Kernel) (Tuned H.P.)',
                    X=X_train_scaled, y=y_train,
                    cv=3, n_jobs=-1)
```



Confusion Matrix

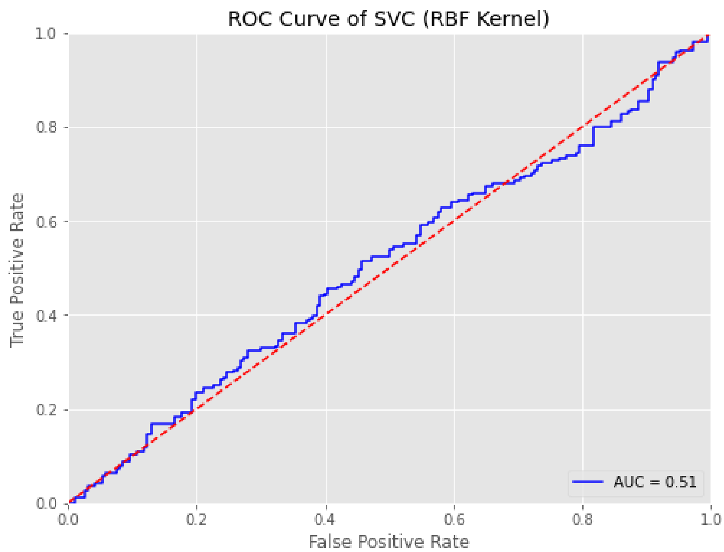
In [76]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(best_svc_rbf_clf, X_test_scaled[(best_vectorizer, best_ngram)], y_test,
                             display_labels=[0, 1],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title("Confusion Matrix of SVC (RBF Kernel)")

print("Confusion Matrix of SVC (RBF Kernel)")
print(disp.confusion_matrix)

plt.show()
```

```
Confusion Matrix of SVC (RBF Kernel)
[[ 18 168]
 [ 19 172]]
```



ROC Curve

In [77]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

y_scores_svc_rbf_best = best_svc_rbf_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])
fpr, tpr, threshold = roc_curve(y_test, y_scores_svc_rbf_best[:, 1])

plt.figure(figsize=(8, 6))
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of SVC (RBF Kernel)')
plt.show()
```



We got a test AUROC over 50%! Not bad.

7d. RandomForestClassifier

In [78]:
```
best_vectorizer = 'count'
best_ngram = (1, 1)
```

In [79]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_clf_init = RandomForestClassifier(random_state=42)

rf_param_grid = [{
        'max_depth': [10**c for c in np.arange(2, 6, dtype=float)]+[None],
#         'max_leaf_nodes':
        'n_estimators': [10, 50, 100, 500, 1000, 5000]
    }]

rf_clf_gscv = GridSearchCV(rf_clf_init, rf_param_grid, cv=3, n_jobs=-1, scoring='roc_auc')
rf_clf_gscv.fit(X_train[(best_vectorizer, best_ngram)], y_train, sample_weight=drn_weights_train)
```

Out[79]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid=[{'max_depth': [100.0, 1000.0, 10000.0, 100000.0, None],
                          'n_estimators': [10, 50, 100, 500, 1000, 5000]}],
             scoring='roc_auc')
```

In [80]:
```python
from sklearn.model_selection import cross_val_score

print('Best hyperparameters:', rf_clf_gscv.best_params_)

rf_clf_gscv_cv_score = cross_val_score(
    rf_clf_gscv.best_estimator_, X_train[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1)
print('5-fold Cross Validation Train Accuracy', rf_clf_gscv_cv_score.mean())

rf_clf_gscv_cv_score_auroc = cross_val_score(
    rf_clf_gscv.best_estimator_, X_train[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1,
    scoring='roc_auc')
print('5-fold Cross Validation Train AUROC', rf_clf_gscv_cv_score_auroc.mean())
```

```
Best hyperparameters: {'max_depth': 100.0, 'n_estimators': 10}
5-fold Cross Validation Train Accuracy 0.515527950310559
5-fold Cross Validation Train AUROC 0.5134871245529233
```

Note: hyperparameter tuning didn't appear to help here (model with default parameters performed better), so we'll use the model with default hyperparameters going forward.

In [81]:
```python
best_rf_clf = clfs[3][(best_vectorizer, best_ngram)]
# best_rf_clf = lr_clf_gscv.best_estimator_
```

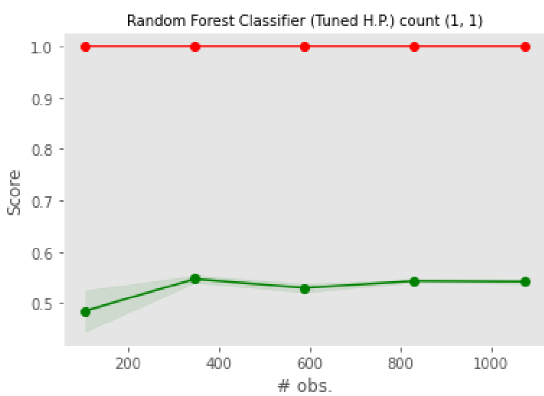
In [82]:
```python
y_pred_rf_clf_gscv_best = best_rf_clf.predict(X_test[(best_vectorizer, best_ngram)])
y_prob_rf_clf_gscv_best = best_rf_clf.predict_proba(X_test[(best_vectorizer, best_ngram)])[:, 1]

print('Test Accuracy:', best_rf_clf.score(X_test[(best_vectorizer, best_ngram)], y_test))

roc_auc = roc_auc_score(y_test, y_prob_rf_clf_gscv_best)
print('Test AUROC:', roc_auc)
```

```
Test Accuracy: 0.5358090185676393
Test AUROC: 0.5212238923605246
```

Learning Curve

Still appears to be overfit.

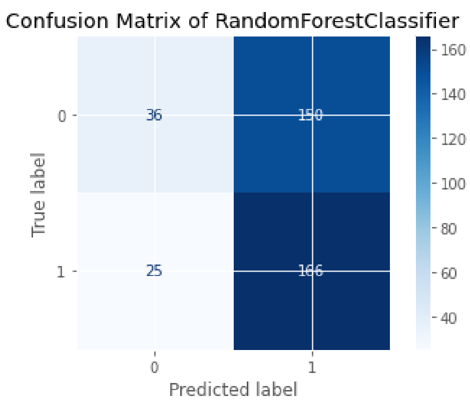
In [83]:
```python
plot_learning_curve(best_rf_clf, vect=best_vectorizer, ngrams=best_ngram,
                    title='Random Forest Classifier (Tuned H.P.)',
                    X=X_train, y=y_train,
                    cv=3, n_jobs=-1)
```



Confusion Matrix

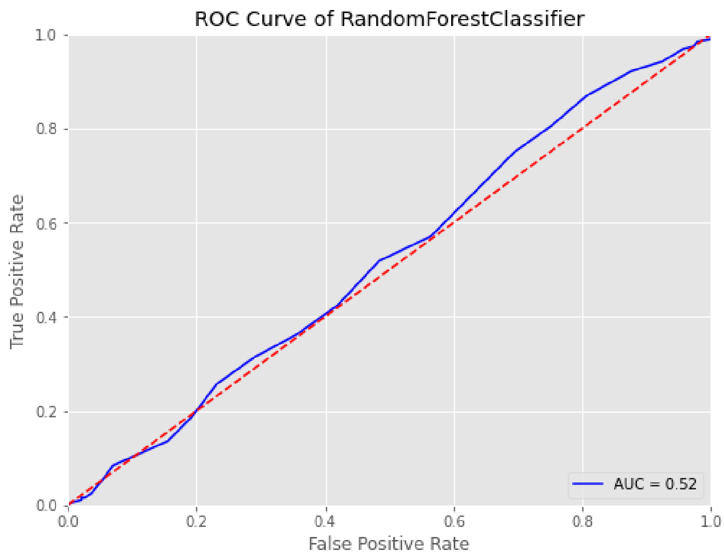
In [84]:
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(best_rf_clf, X_test[(best_vectorizer, best_ngram)], y_test,
                             display_labels=[0, 1],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title("Confusion Matrix of RandomForestClassifier")

print("Confusion Matrix of RandomForestClassifier")
print(disp.confusion_matrix)

plt.show()
```

```
Confusion Matrix of RandomForestClassifier
[[ 36 150]
 [ 25 166]]
```



ROC Curve

In [85]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

y_scores_rf_best = best_rf_clf.predict_proba(X_test[(best_vectorizer, best_ngram)])
fpr, tpr, threshold = roc_curve(y_test, y_scores_rf_best[:, 1])

plt.figure(figsize=(8, 6))
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of RandomForestClassifier')
plt.show()
```



AUROC over 50%! Not bad.

7e. `KNeighborsClassifier`

In [86]:
```
best_vectorizer = 'tfidf'
best_ngram = (1, 1)
```

In [87]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn_clf_init = KNeighborsClassifier()

knn_param_grid = [{
        'n_neighbors': [5*i for i in range(1, 50)],
    }]

knn_clf_gscv = GridSearchCV(knn_clf_init, knn_param_grid, cv=3, n_jobs=-1, scoring='roc_auc')
knn_clf_gscv.fit(X_train_scaled[(best_vectorizer, best_ngram)], y_train)
```

Out[87]:
```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid=[{'n_neighbors': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50,
                                          55, 60, 65, 70, 75, 80, 85, 90, 95,
                                          100, 105, 110, 115, 120, 125, 130,
                                          135, 140, 145, 150, ...]}],
             scoring='roc_auc')
```

In [88]:
```python
from sklearn.model_selection import cross_val_score

print('Best hyperparameters:', knn_clf_gscv.best_params_)

knn_clf_gscv_cv_score = cross_val_score(
    knn_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1)
print('5-fold Cross Validation Train Accuracy', knn_clf_gscv_cv_score.mean())

knn_clf_gscv_cv_score_auroc = cross_val_score(
    knn_clf_gscv.best_estimator_, X_train_scaled[(best_vectorizer, best_ngram)], y_train, cv=5, n_jobs=-1,
    scoring='roc_auc')
print('5-fold Cross Validation Train AUROC', knn_clf_gscv_cv_score_auroc.mean())
```

```
Best hyperparameters: {'n_neighbors': 225}
5-fold Cross Validation Train Accuracy 0.5422360248447206
5-fold Cross Validation Train AUROC 0.5114024741037408
```

Note: hyperparameter tuning didn't appear to help here (model with default parameters performed better), so we'll use the model with default hyperparameters going forward.

In [89]:
```python
best_knn_clf = clfs[4][(best_vectorizer, best_ngram)]
# best_knn_clf = lr_clf_gscv.best_estimator_
```

In [90]:
```python
y_pred_knn_clf_gscv_best = best_knn_clf.predict(X_test_scaled[(best_vectorizer, best_ngram)])
y_prob_knn_clf_gscv_best = best_knn_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])[:, 1]

print('Test Accuracy:', best_knn_clf.score(X_test_scaled[(best_vectorizer, best_ngram)], y_test))

roc_auc = roc_auc_score(y_test, y_prob_knn_clf_gscv_best)
print('Test AUROC:', roc_auc)
```

```
Test Accuracy: 0.53315649867374
Test AUROC: 0.526332826662163
```

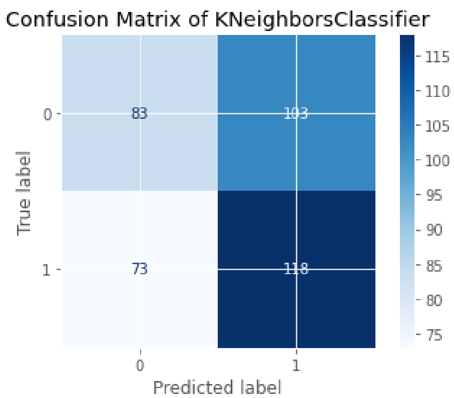
Learning Curve

In [91]:
```python
# KNeighborsClassifier learning curve could not be plotted
# plot_learning_curve(best_knn_clf, vect=best_vectorizer, ngrams=best_ngram,
#                     title='KNeighborsClassifier (Tuned H.P.)',
#                     X=X_train_scaled, y=y_train,
#                     cv=3, n_jobs=-1)
```

Confusion Matrix

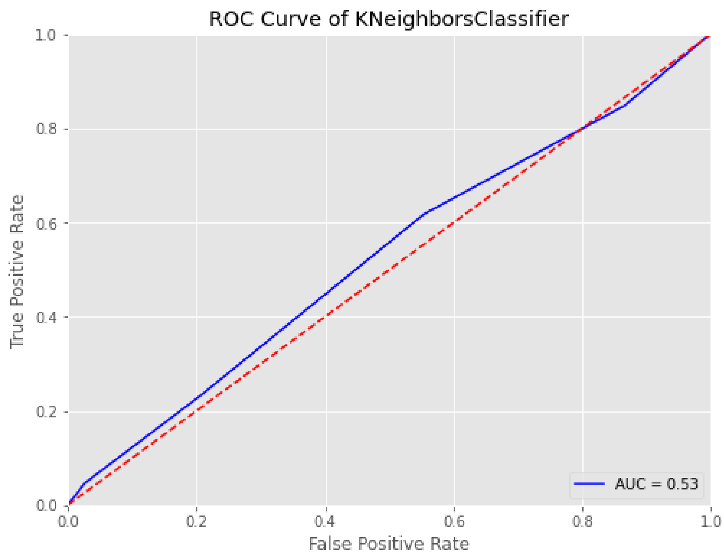
In [92]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(best_knn_clf, X_test_scaled[(best_vectorizer, best_ngram)], y_test,
                             display_labels=[0, 1],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title("Confusion Matrix of KNeighborsClassifier")

print("Confusion Matrix of KNeighborsClassifier")
print(disp.confusion_matrix)

plt.show()
```

```
Confusion Matrix of KNeighborsClassifier
[[ 83 103]
 [ 73 118]]
```



ROC Curve

In [93]:
```python
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

y_scores_knn_best = best_knn_clf.predict_proba(X_test_scaled[(best_vectorizer, best_ngram)])
fpr, tpr, threshold = roc_curve(y_test, y_scores_knn_best[:, 1])

plt.figure(figsize=(8, 6))
plt.title('Receiver Operating Characteristic (ROC)')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNeighborsClassifier')
plt.show()
```



AUROC over 50%! Good result using default hyperparameters.

Discussion

Summary of Results:

After hyperparameter tuning with cross validation, and evaluation on test set, our top 3 best results were:

- Logistic Regression (Count 2 gram), Test AUROC = 54.6%
- SVC (Linear) (Count 2 gram), Test AUROC = 53.9%
- KNN Classifier (TFIDF 1 gram), Test AUROC = 52.6%

Interestingly our models had poor performance after tuning hyperparameters with CV, most likely due simplistic (low capacity) model that were overfitting. Regularization or other hyperparameters could not help these models achieve good performance. Default hyperparameters for our models appeared to work best.

Count vs. TF-IDF Vectorizer

We see that our models generally preferred the features generated by TF-IDF vectorizer. Unsurprising, since TF-IDF Vectorizer greatly reduces the number of features. Our models perhaps preferred the greater regularization/dimensionality reduction provided by TF-IDF Vectorizer. As future work, it would be interesting to test this with an extremely high capacity model, like a deep neural network.

Overall Conclusions

- Surprisingly, our simplest models (LogisticRegression, SVC Linear, & KNeighborsClassifier) ended up performing the best on the test set AUROC. Perhaps the models are slightly better built to generalize better than more advanced models, which may have tended to overfit the high dimensional feature space.
- The more sophisticated models, SVC (RBF Kernel) and (to a lessser extent) Random Forest seemed to not perform as well.
  - Random Forest Classifier seems to be overfit quite easily, even with hyperparameter tuning limiting tree depth and n_estimators, but did better than SVC RBF.
  - SVC (RBF) had a large number of hyperparameters to tune; perhaps we didn't choose the right ranges to properly maximize it's potential. Perhaps another kernel might have worked better.
- When working in high dimensional space, regularization & dimensionality reduction helps. In our case we leveraged TF-IDF scores to help focus on n-grams that were not too common or too rare, relatively to other n-grams.
- Predicting stock market movements with just the top 25 headlines is quite difficult! If we actually did this in practice, our work here would likely just be one factor used in combination with many other predictors.

End of Presentation. Thank you! Questions?

References

[1] Sun, J. (2016). Daily News for Stock Market Prediction, Version 1. Retrieved 23/04/20 from https://www.kaggle.com/aaron7sun/stocknews (https://www.kaggle.com/aaron7sun/stocknews).

[2] Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 1st. ed. O'Reilly Media, Inc.

[3] Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer.

[4] Khadjeh Nassirtoussi, A., Aghabozorgi, S., Ying Wah, T., & Ngo, D. (2014). Text mining for market prediction: A systematic review. Expert Systems with Applications, 41 (16), 7653–7670. https://doi.org/10.1016/j.eswa.2014.06.009 (https://doi.org/10.1016/j.eswa.2014.06.009).