CSML1010 Group3 Course_Project - Milestone 2 - Baseline Machine Learning Implementation

Authors (Group3): Paul Doucet, Jerry Khidaroo

Project Repository: [https://github.com/CSML1010-3-2020/NLPCourseProject (https://github.com/CSML1010-3-2020/NLPCourseProject)](https://github.com/CSML1010-3-2020/NLPCourseProject)

Dataset:

The dataset used in this project is the **Taskmaster-1** dataset from Google. [Taskmaster-1 (https://research.google/tools/datasets/taskmaster-1/)](https://research.google/tools/datasets/taskmaster-1/)

The dataset can be obtained from: [https://github.com/google-research-datasets/Taskmaster (https://github.com/google-research-datasets/Taskmaster)](https://github.com/google-research-datasets/Taskmaster)

Workbook Setup and Data Preparation

Import Libraries

In [1]:
```
# import pandas, numpy
import pandas as pd
import numpy as np
import re
import nltk
```

Set Some Defaults

In [2]:
```
# adjust pandas display
pd.options.display.max_columns = 30
pd.options.display.max_rows = 100
pd.options.display.float_format = '{:.7f}'.format
pd.options.display.precision = 7
pd.options.display.max_colwidth = None

# Import matplotlib and seaborn and adjust some defaults
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

from matplotlib import pyplot as plt
plt.rcParams['figure.dpi'] = 100

import seaborn as sns
sns.set_style("whitegrid")

import warnings
warnings.filterwarnings('ignore')
```

Load Data

In [3]:
```
df_all = pd.read_csv('./data/dialog_norm.csv')
df_all.columns
```

Out[3]:
```
Index(['Instruction_id', 'category', 'selfdialog_norm'], dtype='object')
```

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

Out[4]:

	Instruction_id	category	selfdialog_norm
0	restaurant-table-2	0	hi im looking book table korean fod ok area thinking somewhere southern nyc maybe east village ok great theres thursday kitchen great reviews thats great need table tonight pm people dont want sit bar anywhere else fine dont availability pm times available yikes cant times ok second choice let check ok lets try boka free people yes great lets book ok great requests thats book great use account open yes please great get confirmation phone soon
1	movie-tickets-1	1	hi would like see movie men want playing yes showing would like purchase ticket yes friend two tickets please okay time moving playing today movie showing pm okay anymore movies showing around pm yes showing pm green book two men dealing racisim oh recommend anything else like well like movies funny like comedies well like action well okay train dragon playing pm okay get two tickets want cancel tickets men want yes please okay problem much cost said two adult tickets yes okay okay anything else help yes bring food theater sorry purchase food lobby okay fine thank enjoy movie
2	movie-tickets-3	2	want watch avengers endgame want watch bangkok close hotel currently staying sounds good time want watch movie oclock many tickets two use account already movie theater yes seems movie time lets watch another movie movie want watch lets watch train dragon newest one yes one dont think movie playing time either neither choices playing time want watch afraid longer interested watching movie well great day sir thank welcome

Remove NaN rows

In [5]:
```
print(df_all.shape)
df_all = df_all.dropna()
df_all = df_all.reset_index(drop=True)
print(df_all.shape)
```

```
(7708, 3)
(7705, 3)
```

Get a Sample of records.

In [6]:
```
cat_id_df = df_all[['Instruction_id', 'category']].drop_duplicates().sort_values('category')
cat_count = len(cat_id_df)
sample_size = 3000
sample_per_cat = sample_size//cat_count
print('sample_size: ', sample_size, 'sample_per_cat: ', sample_per_cat)
```

```
sample_size:  3000 sample_per_cat:  214
```

In [7]:
```
# Function to Get balanced Sample - Get a bit more than needed then down sample
def sampling_k_elements(group, k=sample_per_cat + 40):
    if len(group) < k:
        return group
    return group.sample(k, random_state=5)

#Get balanced samples
corpus_df = df_all.groupby('Instruction_id').apply(sampling_k_elements).reset_index(drop=True)

#Reduce to sample_size
corpus_df = corpus_df.sample(n=sample_size, random_state=3)
print (corpus_df.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    248
coffee-ordering-1     243
coffee-ordering-2     243
movie-finder           50
movie-tickets-1       239
movie-tickets-2       248
movie-tickets-3       187
pizza-ordering-1      238
pizza-ordering-2      241
restaurant-table-1    242
restaurant-table-2    241
restaurant-table-3     94
uber-lyft-1           249
uber-lyft-2           237
dtype: int64
```

Generate Corpus List

In [8]:
```python
doc_lst = []
for i, row in corpus_df.iterrows():
    doc_lst.append(row.selfdialog_norm)

print(len(doc_lst))
doc_lst[1:5]
```

```
3000
```

Out[8]:
```
['hey need reserve movie tickets glass sure would like purchase louisville preston crossings day wanting go to
night pm work many tickets need ok give second much tickets ok works confirm want tickets tonights showing gla
ss preston crossings thats right want send tickets phone text message yes would perfect ok tickets way got tex
t thank much youre welcome',
 'hey schedule appointment repair car yes prefer certain shop called intelligent auto solutions one location a
lright pulled need make model year car jeep wrangler seems issue might imagination feel like pulling right rec
ently sometimes drive straight highway take hands steering wheel end realign within lane within seconds theyll
want bring morning urgent want look willing dont open spots see today appointments day closing okay ill bring
morning tomorrow dont open oh work go work type job take hours would fixed time drive work anyway itll wait wo
rk oclock open book fantastic thanks asking name good contact number reach finalize appointment make maisel co
nfirm appointment set repair wheel alignment jeep wrangler pm tomorrow much cost inspection',
 'need restaurant reservation tonight sure kind restaurant interested im entirely sure feel like fish maybe su
shi seafood restaurant ok want eat cleveland yes please cleveland ok well two highest rated places alley cat o
yster bar parallax sushi oh heard never either one better yelp rating close parallax sushi restaurant rated sl
ightly higher ok parallax bar yes full service bar oh good need drink make reservation pm tonight party two re
serve table bar ok opening tonight pm take reservations bar grrr ok well take table dining room hopefully get
open table bar sit ok sounds good done ok going hate kind want oysters could never hate best boss also good ne
ws parallax also oysters get want keep reservation change oh thats great news think keep ok sounds great anyth
ing else yes said oysters mean menu online send email anything else nope cant wait look menu absolutely enjoy
dinner',
 'want reserve table per se okay date looking saturday april time pm many party people per se table availabili
ty april next availabilty may tables april good check cookshop table cookshop available april party time would
like pm pm available pm pm dont work need reservation pm times available ok dont want reserve anything today t
hanks help welcome bye goodbye']
```

Split Data into Train and Test Sets

In [9]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(doc_lst, corpus_df['category'], test_size=0.25, random_state
```

Build Vocabulary

In [10]:
```
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence

tokenizer = text.Tokenizer(lower=False)
tokenizer.fit_on_texts(X_train)
word2id = tokenizer.word_index

word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in X_train]

vocab_size = len(word2id)
embed_size = 100
window_size = 2

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Using TensorFlow backend.

Vocabulary Size: 9051
Vocabulary Sample: [('like', 1), ('would', 2), ('ok', 3), ('okay', 4), ('yes', 5), ('want', 6), ('pm', 7), ('o
rder', 8), ('thank', 9), ('please', 10)]
```

Bag of Words Feature Extraction

In [11]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1., vocabulary=word2id)
cv_matrix = cv.fit_transform(X_train, y_train)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[11]:
```
array([[0, 8, 3, ..., 0, 0, 0],
       [0, 1, 1, ..., 0, 0, 0],
       [0, 4, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 1, 1, 1],
       [0, 6, 6, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [12]:
```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

Out[12]:

	PAD	like	would	ok	okay	yes	want	pm	order	thank	please	tickets	one	time	great	...	marinare	reviewing	lawrence	ri
0	0	8	3	10	0	2	4	7	0	1	2	0	0	4	2	...	0	0	0	
1	0	1	1	9	0	2	1	2	0	0	0	0	1	0	0	...	0	0	0	
2	0	4	0	0	0	3	2	0	4	1	0	0	5	1	0	...	0	0	0	
3	0	0	0	0	3	1	0	0	1	0	0	0	6	0	2	...	0	0	0	
4	0	0	0	0	4	1	2	0	0	1	0	4	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2245	0	1	1	0	4	0	1	1	0	1	1	4	0	2	1	...	0	0	0	
2246	0	2	1	0	3	1	0	6	0	1	0	0	0	1	0	...	0	0	0	
2247	0	0	0	0	4	2	1	0	0	0	0	2	0	1	1	...	0	0	0	
2248	0	6	6	1	2	1	2	0	7	0	0	0	1	0	0	...	0	0	0	
2249	0	0	0	0	0	0	0	0	0	1	2	0	0	1	0	...	0	0	0	

2250 rows × 9051 columns

In [13]:
```
# Get BOW features
X_train_bow = cv_matrix #cv.fit_transform(X_train).toarray()
X_test_bow = cv.transform(X_test).toarray()
y_train = np.array(y_train)
y_test = np.array(y_test)
print (X_train_bow.shape)
print (X_test_bow.shape)
print (y_test.shape)
```

```
(2250, 9051)
(750, 9051)
(750,)
```

Define Model Builder Function

In [14]:
```
#from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix
from sklearn import metrics

class Result_Metrics:
    def __init__(self, predicter, cm, report, f1_score, accuracy, precision, recall):
        self.predicter = predicter
        self.cm = cm    # instance variable unique to each instance
        self.report = report
        self.f1_score = f1_score
        self.accuracy = accuracy
        self.precision = precision
        self.recall = recall

def Build_Model(model, features_train, labels_train, features_test, labels_test):
    classifier = model.fit(features_train, labels_train)

    # Predicter to output
    pred = classifier.predict(features_test)

    # Metrics to output
    cm = confusion_matrix(pred,labels_test)
    report = metrics.classification_report(labels_test, pred)
    f1 = metrics.f1_score(labels_test, pred, average='weighted')
    accuracy = cm.trace()/cm.sum()
    precision = metrics.precision_score(labels_test, pred, average='weighted')
    recall = metrics.recall_score(labels_test, pred, average='weighted')

    rm = Result_Metrics(pred, cm, report, f1, accuracy, precision, recall)

    return rm
```

Bag of Words Feature Benchmarking Baseline with Naive Bayes Classifier

In [15]:
```
from sklearn.naive_bayes import MultinomialNB

model_nb_bow = MultinomialNB()
rm_nb_bow = Build_Model(model_nb_bow, X_train_bow, y_train, X_test_bow, y_test)
```

In [16]:

```
def Save_Benchmark(descr, feat_type, b_metrics, reset_rb, reset_rb_all):
    global rows_benchmarks
    global rows_benchmarks_all
    global df_benchmarks
    global df_benchmarks_all
    if (reset_rb):
        rows_benchmarks = []

    if (reset_rb_all):
        rows_benchmarks_all = []
    rows_benchmarks.append([descr, feat_type, b_metrics.precision, b_metrics.recall, b_metrics.f1_score, b_metri
    rows_benchmarks_all.append([descr, feat_type, b_metrics.precision, b_metrics.recall, b_metrics.f1_score, b_m
    df_benchmarks = pd.DataFrame(rows_benchmarks, columns=["Features_Benchedmarked", "Feat_Type", "Precision", "
    df_benchmarks_all = pd.DataFrame(rows_benchmarks_all, columns=["Features_Benchedmarked", "Feat_Type", "Preci
```

In [17]:

```
# Save benchmark output
Save_Benchmark("BOW Naive Bayes Baseline", "BOW", rm_nb_bow, True, True)
#df_benchmarks
```

In [18]:

```
from sklearn.metrics import confusion_matrix
#rm_nb_bow.cm
```

In [19]:

```
from sklearn import metrics
#print("Label" + rm_nb_bow.report)
```

Feature Selection: BOW Features with Naive Bayes Model Using Chi-Squared Selector

Define Feature Selection Functions

In [20]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler

class Result_Metrics_selected:
    def __init__(self, x_train_sel, x_test_sel, predicter, cm, report, f1_score, accuracy, precision, recall):
        self.x_train_sel = x_train_sel
        self.x_test_sel = x_test_sel
        self.predicter = predicter
        self.cm = cm    # instance variable unique to each instance
        self.report = report
        self.f1_score = f1_score
        self.accuracy = accuracy
        self.precision = precision
        self.recall = recall

def Get_Scaled_Features(features_train, labels_train, features_test, labels_test, scaler):
    x_train_scaled = scaler.fit_transform(features_train, labels_train)
    x_test_scaled = scaler.transform(features_test)
    return x_train_scaled, x_test_scaled

def Select_Best_Features_Chi(num_feats, features_train, labels_train, features_test, labels_test):
    chi_selector = SelectKBest(chi2, k=num_feats)
    chi_selector.fit(features_train, labels_train)
    chi_support = chi_selector.get_support()
    X_train_chi = features_train[:,chi_support]
    X_test_chi = features_test[:,chi_support]
    return X_train_chi, X_test_chi

def Get_Model_Feature_Metrics(model, num_feats, features_train, labels_train, features_test, labels_test, scaler
    X_train_chi, X_test_chi = Select_Best_Features_Chi(num_feats, features_train, labels_train, features_test, l
    x_train_scaled, x_test_scaled = Get_Scaled_Features(X_train_chi, labels_train, X_test_chi, labels_test, scal
    rm_chi = Build_Model(model, x_train_scaled, labels_train, x_test_scaled, labels_test)
    return rm_chi

def SelectBestModelFeatures_Chi(model, num_feats, features_train, labels_train, features_test, labels_test, scal
    X_norm = scaler.fit_transform(features_train, labels_train)
    chi_selector = SelectKBest(chi2, k=num_feats)
    chi_selector.fit(X_norm, labels_train)
    chi_support = chi_selector.get_support()
```

```
    X_train_chi = features_train[:,chi_support]
    X_test_chi = features_test[:,chi_support]

    classifier_chi = model.fit(X_train_chi, labels_train)

    # Predicter to output
    predict_chi = classifier_chi.predict(X_test_chi)

    # Metrics to output
    cm_chi = confusion_matrix(predict_chi,labels_test)
    report_chi = metrics.classification_report(labels_test, predict_chi)
    f1_chi = metrics.f1_score(labels_test, predict_chi, average='weighted')
    accuracy_chi = cm_chi.trace()/cm_chi.sum()
    precision_chi = metrics.precision_score(labels_test, predict_chi, average='weighted')
    recall_chi = metrics.recall_score(labels_test, predict_chi, average='weighted')

    rm_chi = Result_Metrics_selected(X_train_chi, X_test_chi, predict_chi, cm_chi, report_chi, f1_chi, accuracy_

    return rm_chi
```

Iterate through number of features and get benchmark results

In [21]:
```
a = 100
tot = X_train_bow.shape[1]
b = 100 * (tot//100)
c = 100
print(a, b, c)
```

```
100 9000 100
```

In [22]:
```
rows = []

scaler_min_max = MinMaxScaler()
for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
    #rm_chi_i = Get_Model_Feature_Metrics(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_
    rm_chi_i = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_mir
    rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])

acc_df = pd.DataFrame(rows, columns=["num_of_features", "f1_score", "accuracy"])
```

Plot f1-score by number of selected features

In [24]:
```
Opt_no_of_feat = int(acc_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
Opt_no_of_feat
a = Opt_no_of_feat - 50
b = Opt_no_of_feat + 50
c = 1
print(a, b, c)
#acc_df.sort_values(by='f1_score', ascending=False).head(5)
```

```
750 850 1
```

Get a more fine-grained look at the optimal number of features region

In [25]:
```
rows = []
for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
    #rm_chi_i = Get_Model_Feature_Metrics(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_
    rm_chi_i = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_mir
    rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])

acc_df = pd.DataFrame(rows, columns=["num_of_features", "f1_score", "accuracy"])
```

In [27]:
```
Opt_no_of_feat = int(acc_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
print(Opt_no_of_feat)
#acc_df.sort_values(by='f1_score', ascending=False).head(5)
```

```
794
```

Benchmark BOW With Optimal Features Selected using Naive Bayes Model

In [28]:
```
model_nb_bow_opt = MultinomialNB()
rm_chi_opt_bow = SelectBestModelFeatures_Chi(model_nb_bow, Opt_no_of_feat, X_train_bow, y_train, X_test_bow, y_t
```

In [31]:
```
# Save benchmark output
Save_Benchmark("BOW Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "BOW", rm_chi_opt_bow, False
df_benchmarks
```

Out[31]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW Naive Bayes Baseline	BOW	0.6990870	0.6946667	0.6843059	0.6946667
1	BOW Naive Bayes Optimal Features Selected: 794	BOW	0.7698257	0.7520000	0.7431145	0.7520000

1. Benchmark Comparison

Benchmark the following four models: Logistic Regression (Multinomial) Naive Bayes Linear Support Vector Machine Random Forest

Baseline Features

In [32]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score

model_ids = ['RF', 'SVC', 'NB','LR']
models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0, max_iter=500),
]
CV = 10
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []

for model, model_id in zip(models, model_ids):
    model_name = model.__class__.__name__
    f1_scores = cross_val_score(model, X_train_bow, y_train, scoring='f1_weighted', cv=CV)
    #precisions = cross_val_score(model, X_train_bow, y_train, scoring='precision_weighted', cv=CV)
    #recalls = cross_val_score(model, X_train_bow, y_train, scoring='recall_weighted', cv=CV)

    for i in range(0, 9, 1):
        entries.append((model_id, model_name, 'baseline', 'default', '', f1_scores[i]))

cv_df = pd.DataFrame(entries, columns=['Model_Id', 'Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Score'
```

Optimised Features

In [33]:

```
models = [
    RandomForestClassifier(n_jobs=-1),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(n_jobs=-1)
]
CV = 10
cv_df = pd.DataFrame(index=range(CV * len(models)))
#entries = []

for model, model_id in zip(models, model_ids):
    model_name = model.__class__.__name__
    f1_scores = cross_val_score(model, rm_chi_opt_bow.x_train_sel, y_train, scoring='f1_weighted', cv=CV)
    #precisions = cross_val_score(model, rm_chi_opt_bow.x_train_sel, y_train, scoring='precision_weighted', cv=C
    #recalls = cross_val_score(model, rm_chi_opt_bow.x_train_sel, y_train, scoring='recall_weighted', cv=CV)

    for i in range(0, 9, 1):
        entries.append((model_id, model_name, 'optimized', 'default', '', f1_scores[i]))
        #entries.append((model_name, 'optimized', precisions[i], recalls[i], f1_scores[i]))

cv_df = pd.DataFrame(entries, columns=['Model_Id','Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Score']
```

Modeling

Four different models were verified as part of our modeling:

- Random Forest
- Linear SVC
- Multinomial Naïve Bayes
- Logistic Regression

The modeling was first done on our baseline features and using the selected optimised features identified as part of milestone 1: Naïve Bayes using Chi Squared.

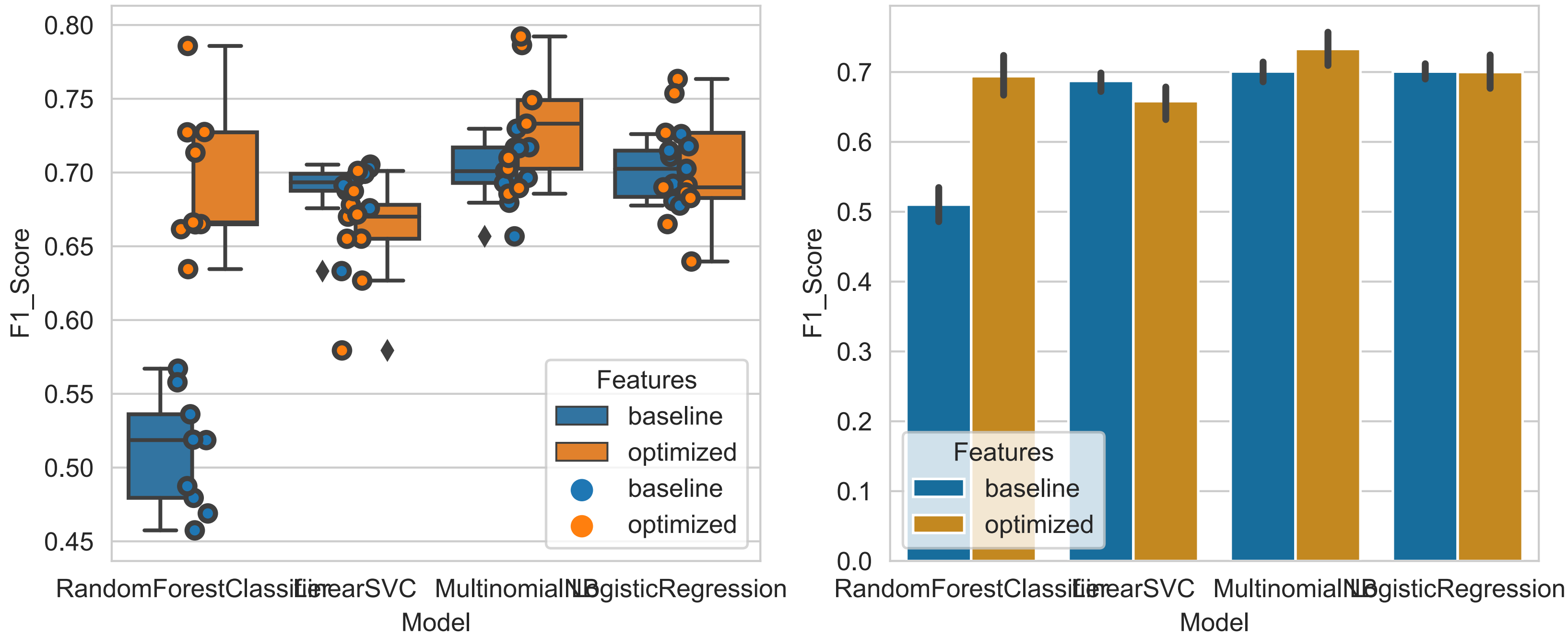
In [34]:
```python
models_df = cv_df.groupby(['Model_Id', 'Model','Features', 'Hyper_Param', 'Best_Params']).agg(['mean'])
models_df.columns = models_df.columns.map('_'.join)
models_df
```

Out[34]:

					F1_Score_mean
Model_Id	**Model**	**Features**	**Hyper_Param**	**Best_Params**	
LR	**LogisticRegression**	**baseline**	**default**		0.7006732
		optimized	**default**		0.6999498
NB	**MultinomialNB**	**baseline**	**default**		0.7007650
		optimized	**default**		0.7330550
RF	**RandomForestClassifier**	**baseline**	**default**		0.5101938
		optimized	**default**		0.6940290
SVC	**LinearSVC**	**baseline**	**default**		0.6872884
		optimized	**default**		0.6582940

In [88]:
```python
plt.subplots?
```

In [98]:
```
import seaborn as sns

fig, (ax1, ax2) = plt.subplots(figsize=(10, 4), ncols=2, sharex=True)
sns.boxplot(x='Model', y='F1_Score', data=cv_df, hue='Features', ax=ax1);
sns.stripplot(x='Model', y='F1_Score', data=cv_df, hue='Features', size=6, jitter=True, edgecolor="gray", linewi
sns.barplot(y='F1_Score', x='Model', data=cv_df, palette="colorblind", hue='Features', ax=ax2);
```



Optimize the Hyperparameters Using Grid Search

In [36]:
```python
from sklearn.model_selection import GridSearchCV

class Estimator_Parameters:
    def __init__(self, estimator, parameters, feat_type, x, y):
        self.estimator = estimator
        self.parameters = parameters
        self.feat_type = feat_type
        self.x = x
        self.y = y

def Get_Best_Parameters(est_param):
    grid_search = GridSearchCV(estimator = est_param.estimator,
                               param_grid = est_param.parameters,
                               scoring = 'f1_weighted',
                               cv= 10,
                               n_jobs = -1)
    grid_search = grid_search.fit(est_param.x, est_param.y)
    return grid_search.best_score_, grid_search.best_params_
```

In [37]:
```
from sklearn.model_selection import GridSearchCV

est_param_arr = [
    Estimator_Parameters(RandomForestClassifier(), [{'n_estimators': [50,100,150,200,250,300],'max_depth': [1, 2
    Estimator_Parameters(LinearSVC(), [{'C': [1000, 1400, 1500, 1600],'loss': ['hinge', 'squared_hinge'], 'dual'
    Estimator_Parameters(MultinomialNB(), [{'alpha': [0.1,0.2,0.3,0.4,0.42,0.44,0.46,0.48,0.5],'fit_prior': [Tru
    Estimator_Parameters(LogisticRegression(), [{'C': [1,2,3], 'penalty': ['l1', 'l2', 'elasticnet', 'none'],'du
]

for est_param, model_id in zip(est_param_arr, model_ids):
    estimator_name = est_param.estimator.__class__.__name__
    best_accuracy, best_parameters = Get_Best_Parameters(est_param)
    entries.append([model_id, estimator_name, est_param.feat_type, 'tuned', str(best_parameters), best_accuracy]
    print(estimator_name, best_accuracy, best_parameters, est_param.feat_type)
```

```
RandomForestClassifier 0.6332853097716822 {'max_depth': 5, 'n_estimators': 200, 'random_state': 3} optimized
LinearSVC 0.611574550815545 {'C': 1500, 'dual': False, 'loss': 'squared_hinge', 'max_iter': 2000, 'penalty':
'l1'} optimized
MultinomialNB 0.7374644141766777 {'alpha': 0.5, 'fit_prior': True} optimized
LogisticRegression 0.6982668543965878 {'C': 1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'} optimize
d
```

Parameter Tuning

The model's hyperparameters were optimized using the GridSearchCV function from sci-kitlearn. The hyperparameters verified were:

- **Random Forest:** max_depth; n_estimators; random_state
- **Linear SVC:** C; dual; loss; max_iter; penalty
- **MultinomialNB:** alpha; fit_prior
- **Logistic Regression:** C; dual; multi_class; auto; penalty

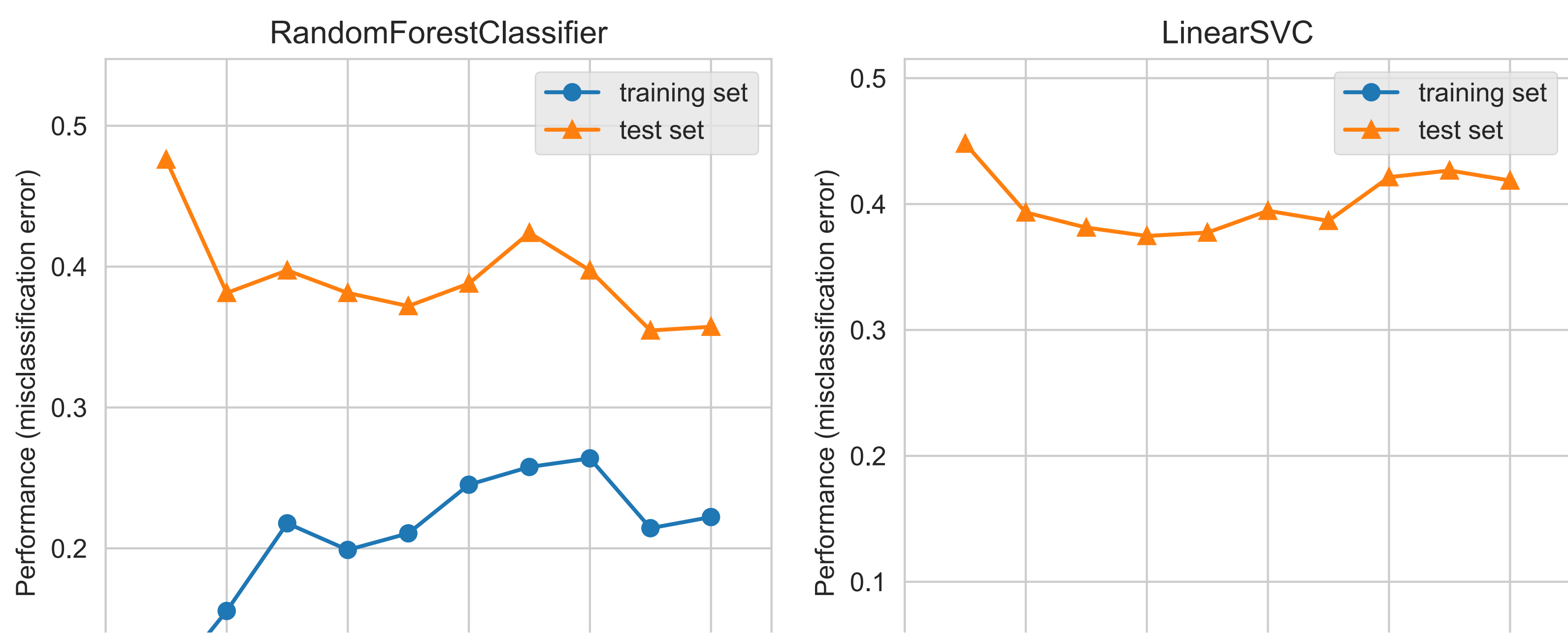
In [38]:
```
result_df = pd.DataFrame(entries, columns=['Model_Id', 'Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Sc
models_df = result_df.groupby(['Model_Id', 'Model','Features','Hyper_Param', 'Best_Params']).agg(['mean'])
models_df.columns = models_df.columns.map('_'.join)
models_df
```

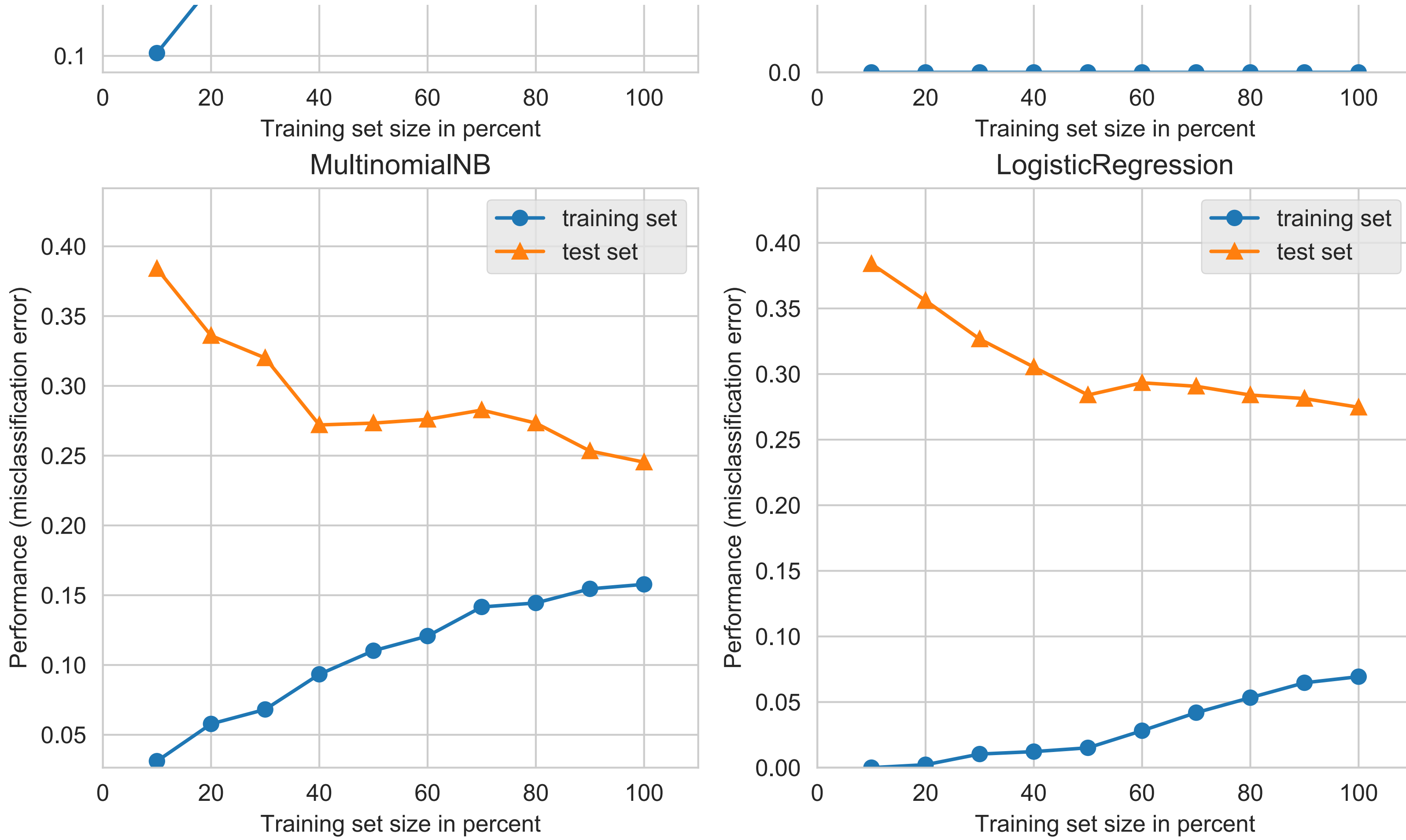
Out[38]:

Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score_mean
LR	LogisticRegression	baseline	default		0.7006732
		optimized	default		0.6999498
			tuned	{'C': 1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.6982669
NB	MultinomialNB	baseline	default		0.7007650
		optimized	default		0.7330550
			tuned	{'alpha': 0.5, 'fit_prior': True}	0.7374644
RF	RandomForestClassifier	baseline	default		0.5101938
		optimized	default		0.6940290
			tuned	{'max_depth': 5, 'n_estimators': 200, 'random_state': 3}	0.6332853
SVC	LinearSVC	baseline	default		0.6872884
		optimized	default		0.6582940
			tuned	{'C': 1500, 'dual': False, 'loss': 'squared_hinge', 'max_iter': 2000, 'penalty': 'l1'}	0.6115746

2. a. Learning Curves: Training/ Testing Errors - Optimized Hyperarameters

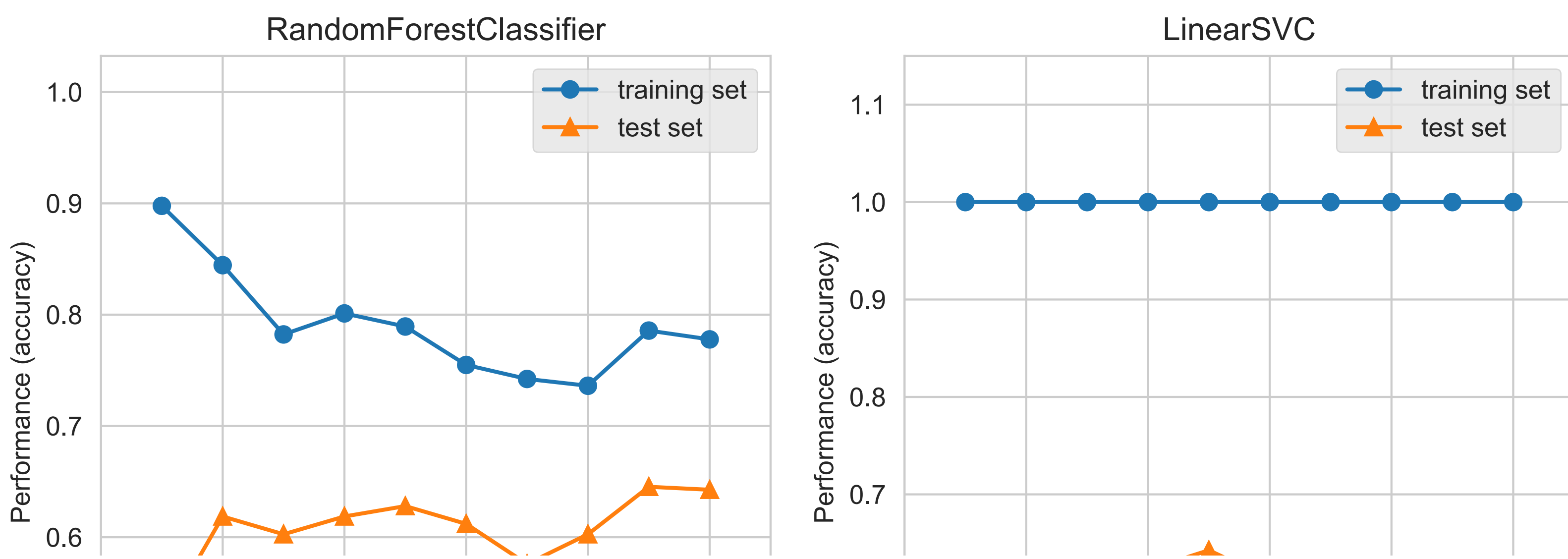
In [39]:
```
from mlxtend.plotting import plot_learning_curves
import itertools
import matplotlib.gridspec as gridspec

models = [
    RandomForestClassifier(n_estimators=300, max_depth=5, random_state=1),
    LinearSVC(C=1400, dual=False, loss='squared_hinge', max_iter=1500, penalty='l1'),
    MultinomialNB(alpha=0.2, fit_prior=True),
    LogisticRegression(C=1, dual=False, multi_class='ovr', penalty='l2'),
]

fig2 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for model, grd in zip(models, grid):
    model_name = model.__class__.__name__
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig2 = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, model, p
    plt.title(model_name)

plt.show()
```

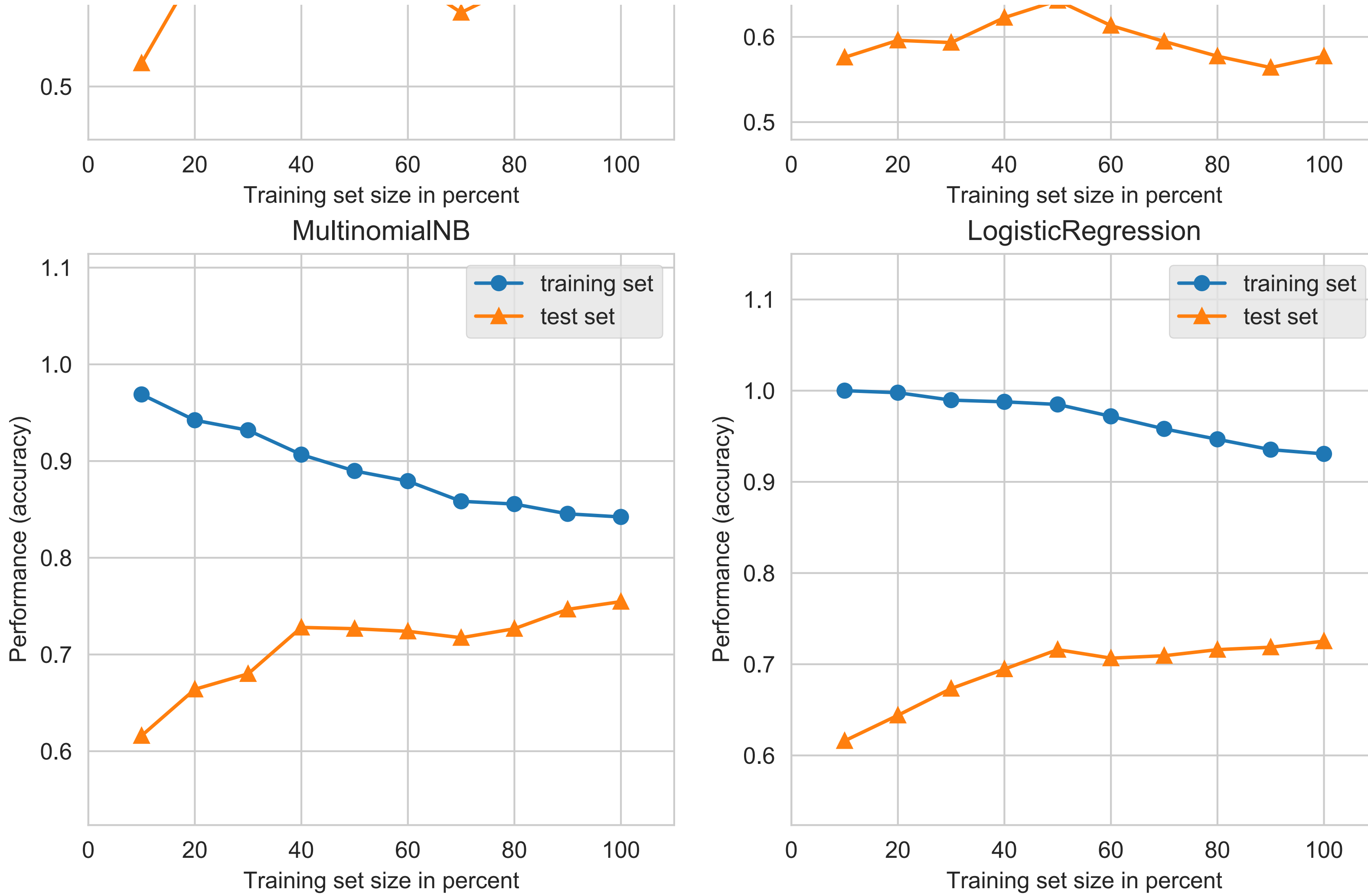




2. b. Learning Curves: Training/Testing Accuracy - Optimized Hyperarameters

In [40]:
```python
from mlxtend.plotting import plot_learning_curves
import matplotlib.gridspec as gridspec
import itertools

models = [
    RandomForestClassifier(n_estimators=300, max_depth=5, random_state=1),
    LinearSVC(C=1400, dual=False, loss='squared_hinge', max_iter=1500, penalty='l1'),
    MultinomialNB(alpha=0.2, fit_prior=True),
    LogisticRegression(C=1, dual=False, multi_class='ovr', penalty='l2'),
]

#fig2 = plt.figure(figsize=(10, 10))
fig3 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for model, grd in zip(models, grid):
    model_name = model.__class__.__name__
    ax = plt.subplot(gs[grd[0], grd[1]])
    #fig2 = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, model,
    fig3 = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, model, s
    plt.title(model_name)

plt.show()
```





Learning Curves

The learning curves for training/testing indicated the following: low error and a high gap between the training and the validation curves. This indicates:

- High variance
- Low bias

Increasing the number of samples gave us more convergence on our curves, but two of the models continue to indicate 100% validation indicating more samples are required.

3. Ensemble Learning

Initialize Models with optimized hyperparameters

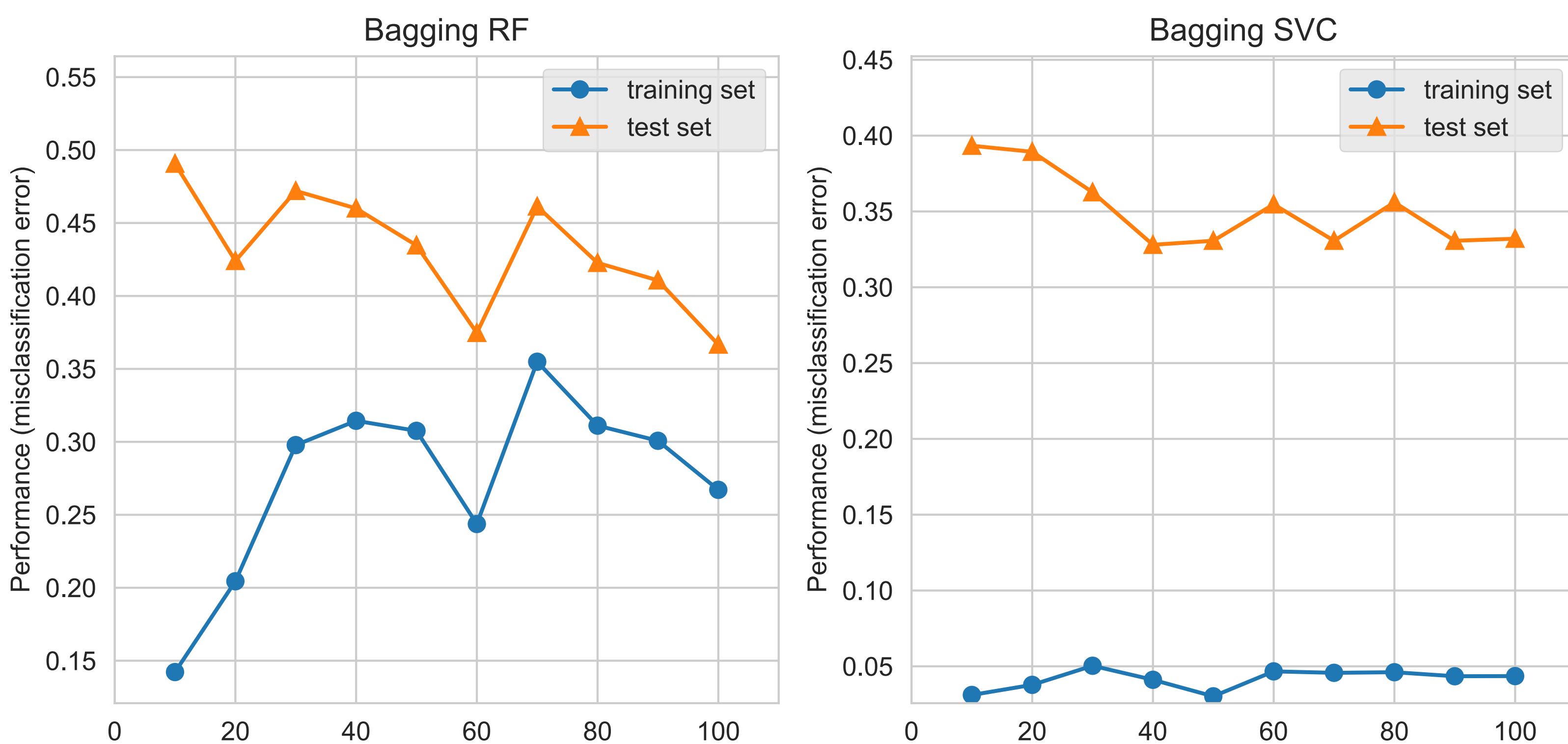
In [41]:
```
clf1 = RandomForestClassifier(criterion='entropy', max_depth=4, n_estimators = 250, random_state = 1)
clf2 = LinearSVC(C=1000, dual=False, loss='squared_hinge', max_iter=2000, penalty='l1')
clf3 = MultinomialNB(alpha=0.1, fit_prior=True)
clf4 = LogisticRegression(C=3, dual=False, multi_class='ovr', penalty='l2')
```

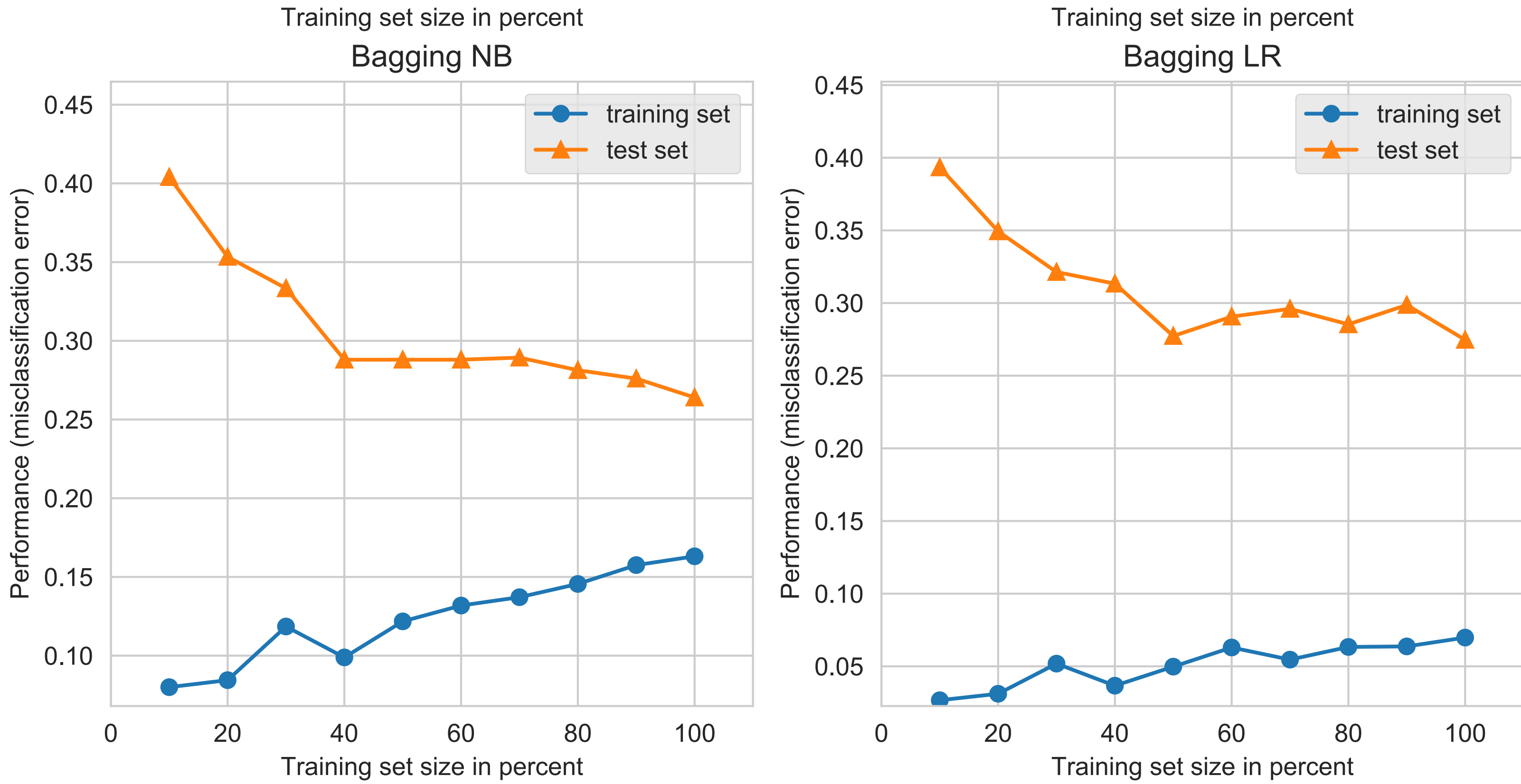
Bagging

In [42]:
```
from sklearn.ensemble import BaggingClassifier

bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=10, max_samples=0.8)
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=10, max_samples=0.8)
bagging3 = BaggingClassifier(base_estimator=clf3, n_estimators=10, max_samples=0.8)
bagging4 = BaggingClassifier(base_estimator=clf4, n_estimators=10, max_samples=0.8)
```

Learning Curves for Bagged Models

In [43]:
```
from mlxtend.plotting import plot_learning_curves

models = [
    bagging1, bagging2, bagging3, bagging4
]
labels = ['Bagging RF', 'Bagging SVC', 'Bagging NB','Bagging LR']

fig2 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for model, label, grd in zip(models, labels, grid):
    model_name = model.__class__.__name__
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig2 = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, model, p
    plt.title(label)

plt.show()
```

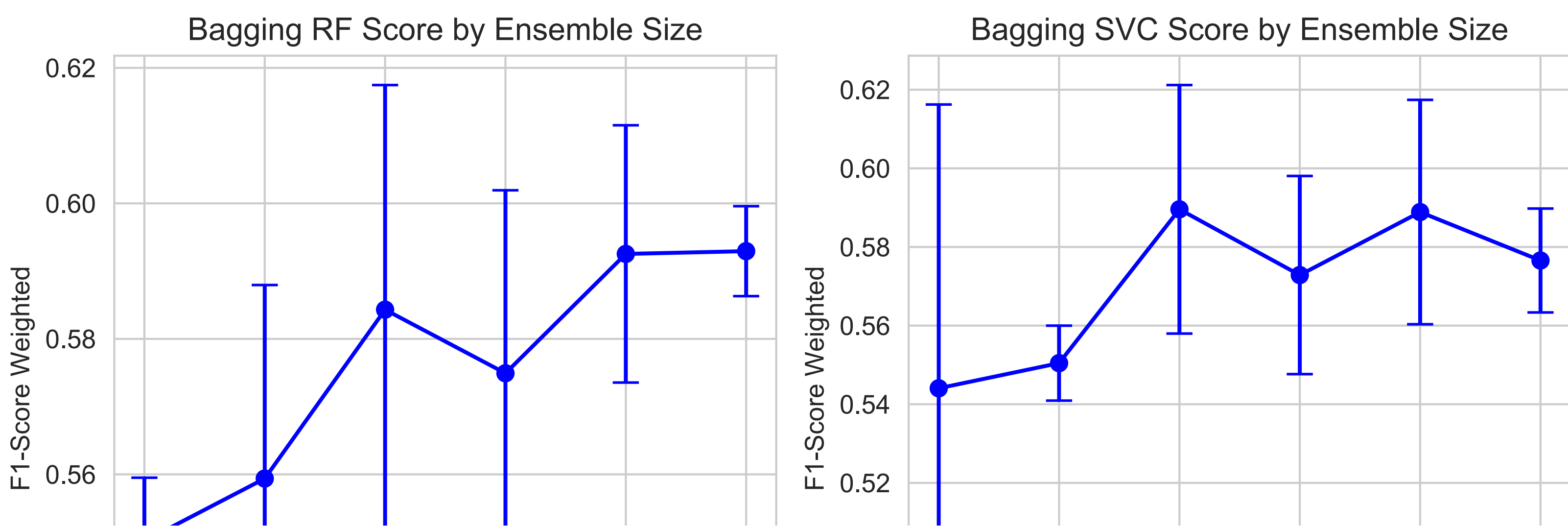


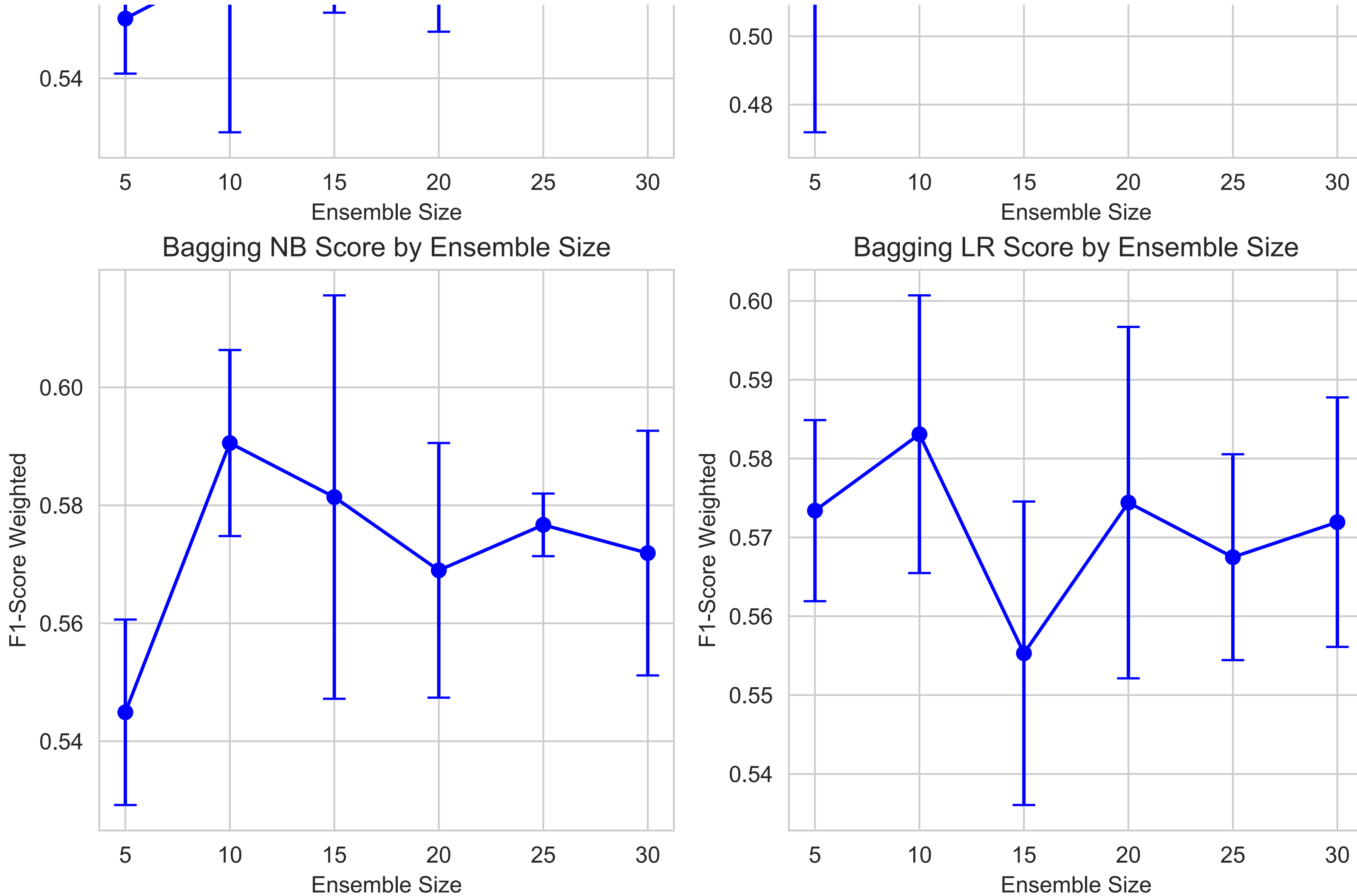


Bagging Scores Varied by Ensemble Size

In [44]:

```
clf_list = [clf1, clf2, clf3, clf4]
labels = ['Bagging RF', 'Bagging SVC', 'Bagging NB','Bagging LR']

fig2 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for clf, label, grd in zip(clf_list, labels, grid):
    num_est = map(int, np.linspace(5,30,6))
    bg_clf_cv_mean = []
    bg_clf_cv_std = []
    for n_est in num_est:
        bg_clf = BaggingClassifier(base_estimator=clf1, n_estimators=n_est, max_samples=0.8, max_features=0.8)
        scores = cross_val_score(bg_clf, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
        bg_clf_cv_mean.append(scores.mean())
        bg_clf_cv_std.append(scores.std())

    num_est = list(map(int, np.linspace(5,30,6)))
    ax = plt.subplot(gs[grd[0], grd[1]])

    (_, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue', fmt='-o', capsize=5)
    for cap in caps:
        cap.set_markeredgewidth(1)

    fig2 = plt.ylabel('F1-Score Weighted'); plt.xlabel('Ensemble Size'); plt.title(label + ' Score by Ensemble S
plt.show()
```





In [45]:
```
bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=30, max_samples=0.9)
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=10, max_samples=0.7)
bagging3 = BaggingClassifier(base_estimator=clf3, n_estimators=20, max_samples=0.8)
bagging4 = BaggingClassifier(base_estimator=clf4, n_estimators=15, max_samples=0.7)
```

In [46]:

```
from mlxtend.plotting import plot_decision_regions
import itertools
import matplotlib.gridspec as gridspec

labels = ['Bagging RF', 'Bagging SVC', 'Bagging NB','Bagging LR']
clf_list = [bagging1, bagging2, bagging3, bagging4]

fig = plt.figure(figsize=(10, 8))
gs = gridspec.GridSpec(2, 4)
grid = itertools.product([0,1],repeat=4)

for clf, label, grd, model_id in zip(clf_list, labels, grid, model_ids):
    scores = cross_val_score(clf, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
    entries.append([model_id, label, 'optimized', 'tuned', '', scores.mean()])
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
```

```
Accuracy: 0.58 (+/- 0.02) [Bagging RF]
Accuracy: 0.68 (+/- 0.01) [Bagging SVC]
Accuracy: 0.72 (+/- 0.03) [Bagging NB]
Accuracy: 0.68 (+/- 0.01) [Bagging LR]

<Figure size 1000x800 with 0 Axes>
```

```
In [47]: result_df = pd.DataFrame(entries, columns=['Model_Id', 'Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Sc
models_df = result_df.groupby(['Model_Id','Model','Features','Hyper_Param', 'Best_Params']).agg(['mean'])
models_df.columns = models_df.columns.map('_'.join)
models_df
```

Out[47]:

					F1_Score_mean
Model_Id	**Model**	**Features**	**Hyper_Param**	**Best_Params**	
LR	**Bagging LR**	**optimized**	**tuned**		0.6831600
	LogisticRegression	**baseline**	**default**		0.7006732
		optimized	**default**		0.6999498
			tuned	**{'C': 1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}**	0.6982669
NB	**Bagging NB**	**optimized**	**tuned**		0.7245268
	MultinomialNB	**baseline**	**default**		0.7007650
		optimized	**default**		0.7330550
			tuned	**{'alpha': 0.5, 'fit_prior': True}**	0.7374644
RF	**Bagging RF**	**optimized**	**tuned**		0.5844920
	RandomForestClassifier	**baseline**	**default**		0.5101938
		optimized	**default**		0.6940290
			tuned	**{'max_depth': 5, 'n_estimators': 200, 'random_state': 3}**	0.6332853
SVC	**Bagging SVC**	**optimized**	**tuned**		0.6757373
	LinearSVC	**baseline**	**default**		0.6872884
		optimized	**default**		0.6582940
			tuned	**{'C': 1500, 'dual': False, 'loss': 'squared_hinge', 'max_iter': 2000, 'penalty': 'l1'}**	0.6115746

The Bagging ensemble did not provide any improvements on the baseline and optimized modeling.

Boosting

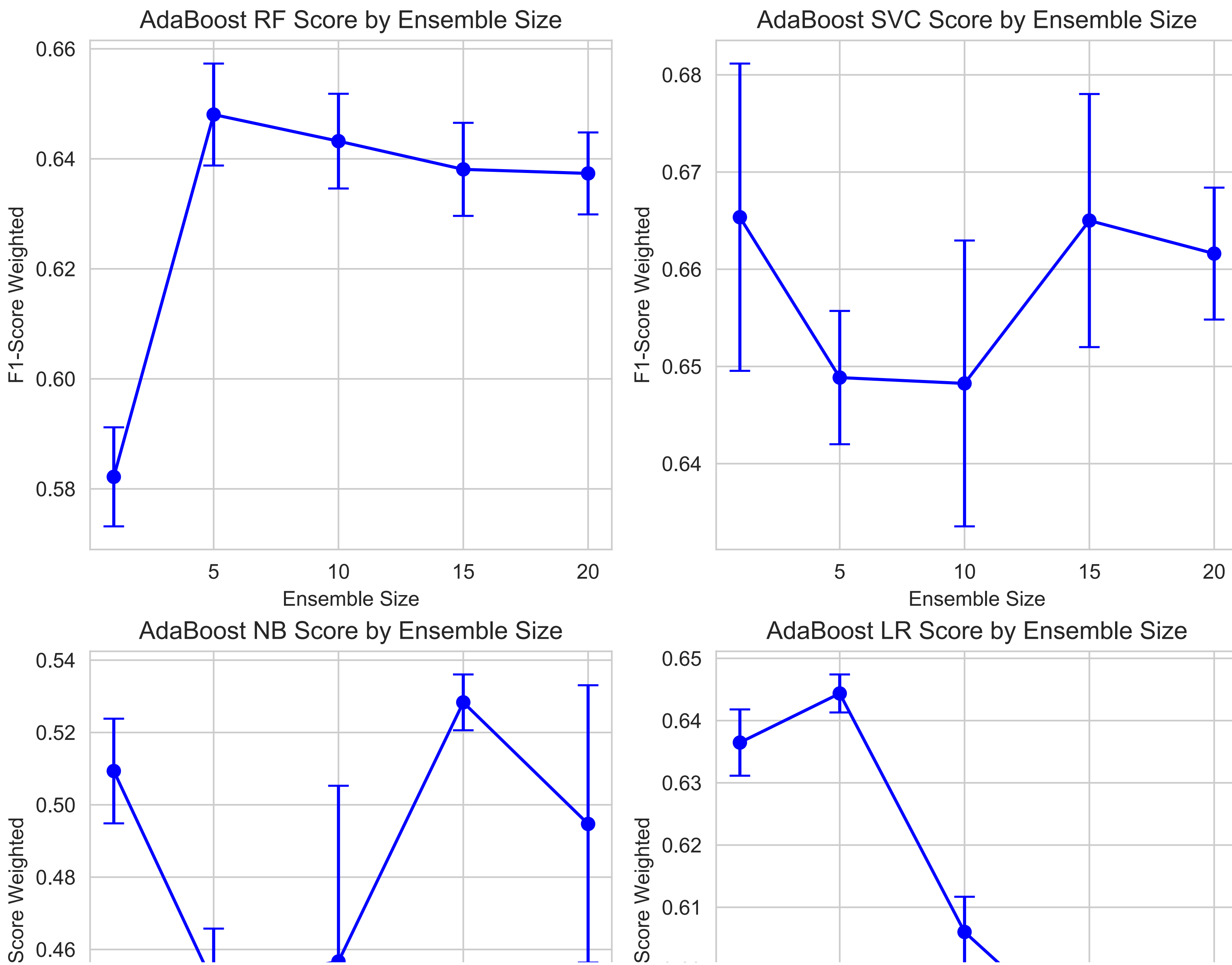
In [48]:
```
from sklearn.ensemble import AdaBoostClassifier

boosting1 = AdaBoostClassifier(base_estimator=clf1)
boosting2 = AdaBoostClassifier(base_estimator=clf2, algorithm='SAMME')
boosting3 = AdaBoostClassifier(base_estimator=clf3)
boosting4 = AdaBoostClassifier(base_estimator=clf4)
```

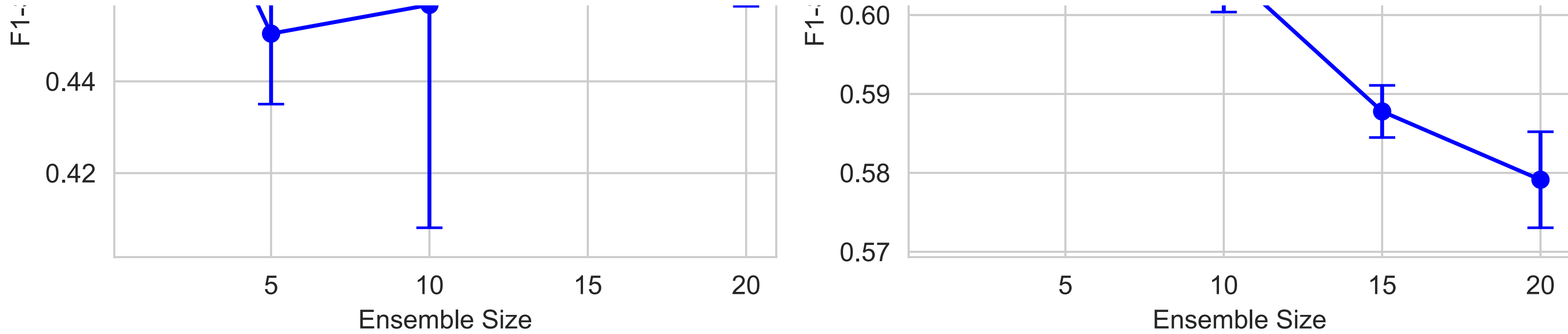
Boosting Scores Varied by Ensemble Size

```
In [49]: from sklearn.ensemble import AdaBoostClassifier

bst_list = [boosting1, boosting2, boosting3, boosting4]
labels = ['AdaBoost RF', 'AdaBoost SVC', 'AdaBoost NB','AdaBoost LR']

fig2 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for boosting, label, grd in zip(bst_list, labels, grid):
    num_est = map(int, np.linspace(1,20,5))
    bg_clf_cv_mean = []
    bg_clf_cv_std = []
    for n_est in num_est:
        boosting.set_params(n_estimators=n_est)
        scores = cross_val_score(boosting, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
        bg_clf_cv_mean.append(scores.mean())
        bg_clf_cv_std.append(scores.std())
        print("F1-Score Weighted: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))

    ax = plt.subplot(gs[grd[0], grd[1]])
    num_est = list(map(int, np.linspace(1,20,5)))
    (_, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue', fmt='-o', capsize=5)
    for cap in caps:
        cap.set_markeredgewidth(1)
    fig2 = plt.ylabel('F1-Score Weighted'); plt.xlabel('Ensemble Size'); plt.title(label + ' Score by Ensemble S
```

```
F1-Score Weighted: 0.58 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.65 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.64 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.64 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.64 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.67 (+/- 0.02) [AdaBoost SVC]
F1-Score Weighted: 0.65 (+/- 0.01) [AdaBoost SVC]
F1-Score Weighted: 0.65 (+/- 0.01) [AdaBoost SVC]
F1-Score Weighted: 0.67 (+/- 0.01) [AdaBoost SVC]
F1-Score Weighted: 0.66 (+/- 0.01) [AdaBoost SVC]
F1-Score Weighted: 0.51 (+/- 0.01) [AdaBoost NB]
F1-Score Weighted: 0.45 (+/- 0.02) [AdaBoost NB]
F1-Score Weighted: 0.46 (+/- 0.05) [AdaBoost NB]
F1-Score Weighted: 0.53 (+/- 0.01) [AdaBoost NB]
```

```
F1-Score Weighted: 0.49 (+/- 0.04) [AdaBoost NB]
F1-Score Weighted: 0.64 (+/- 0.01) [AdaBoost LR]
F1-Score Weighted: 0.64 (+/- 0.00) [AdaBoost LR]
F1-Score Weighted: 0.61 (+/- 0.01) [AdaBoost LR]
F1-Score Weighted: 0.59 (+/- 0.00) [AdaBoost LR]
F1-Score Weighted: 0.58 (+/- 0.01) [AdaBoost LR]
```



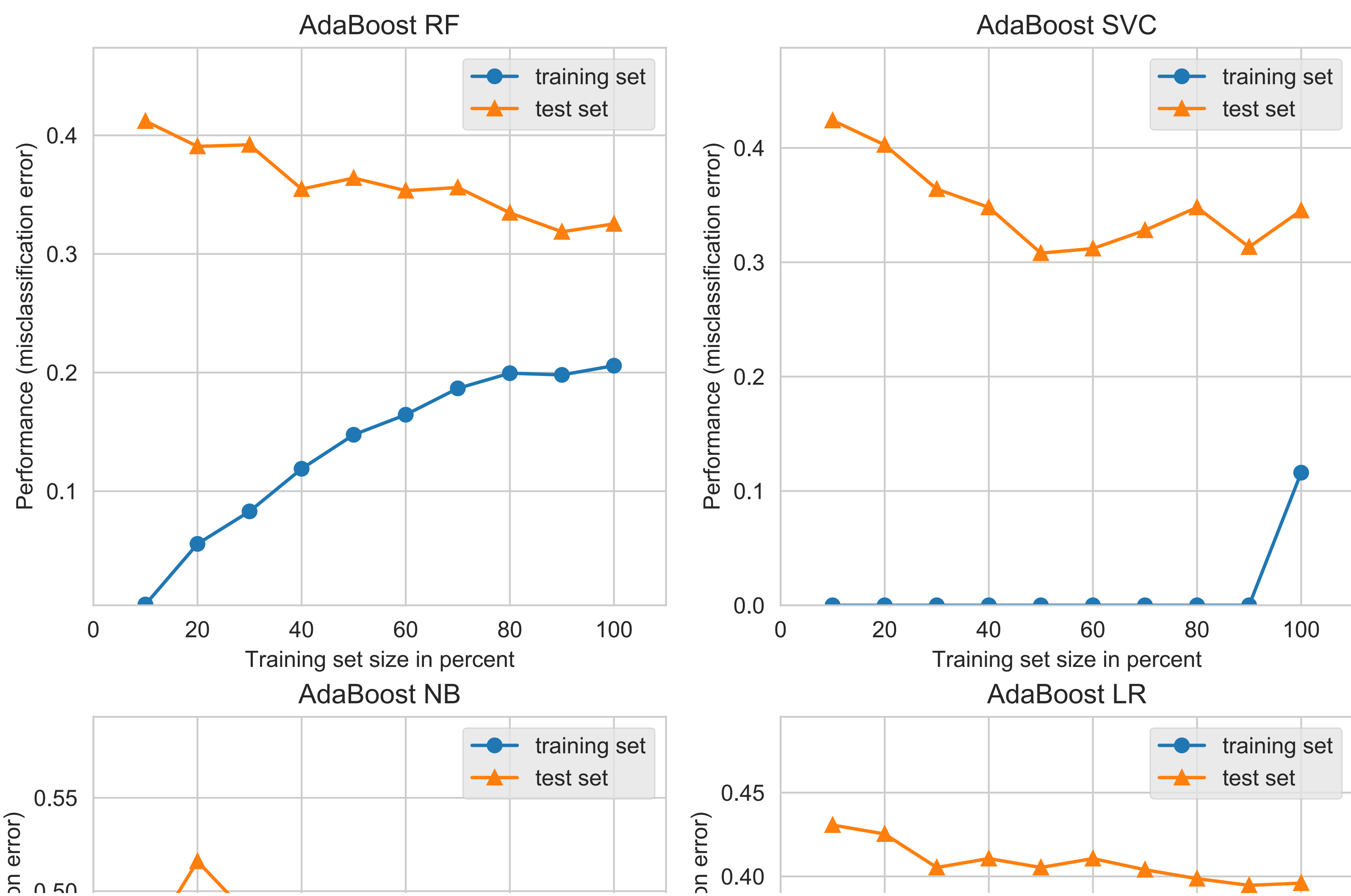


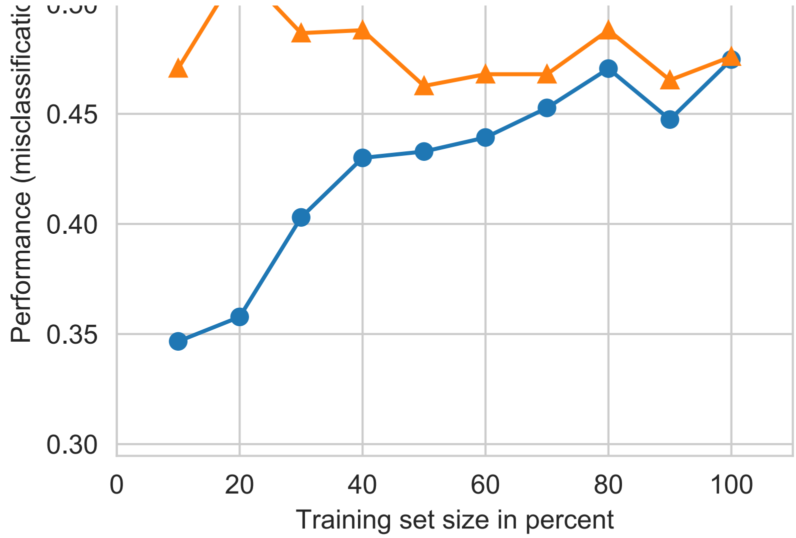
In [50]:
```
plt.show()
```

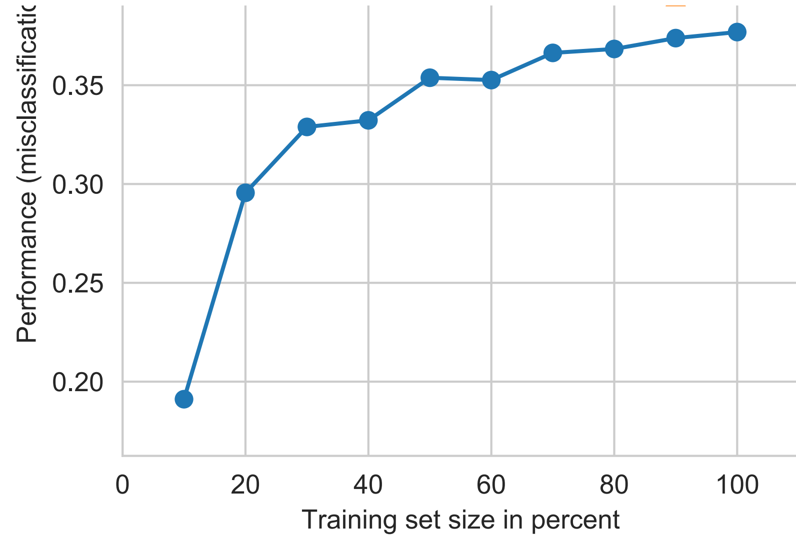
Learning Curves for Boosted Models

In [51]:

```
#plot Boosting learning curve
fig_bst = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for boosting, label, grd in zip(bst_list, labels, grid):
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig_bst = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test,\
                                   boosting, print_model=False, style='ggplot')
    plt.title(label)
plt.show()
```







In [52]:
```
boosting1 = AdaBoostClassifier(base_estimator=clf1, n_estimators=10)
boosting2 = AdaBoostClassifier(base_estimator=clf2, n_estimators=3, algorithm='SAMME')
boosting3 = AdaBoostClassifier(base_estimator=clf3, n_estimators=1)
boosting4 = AdaBoostClassifier(base_estimator=clf4, n_estimators=2)
boost_list = [boosting1, boosting2, boosting3, boosting4]
labels_bst = ['AdaBoost RF', 'AdaBoost SVC', 'AdaBoost NB','AdaBoost LR']
```

In [53]:
```
from sklearn.ensemble import AdaBoostClassifier

labels = ['AdaBoost RF', 'AdaBoost SVC', 'AdaBoost NB','AdaBoost LR']
bst_list = [boosting1, boosting2, boosting3, boosting4]

for boosting, label, model_id in zip(bst_list, labels, model_ids):

    scores = cross_val_score(boosting, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
    entries.append([model_id, label, 'optimized', 'tuned', '', scores.mean()])
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
```

```
Accuracy: 0.65 (+/- 0.01) [AdaBoost RF]
Accuracy: 0.65 (+/- 0.00) [AdaBoost SVC]
Accuracy: 0.51 (+/- 0.01) [AdaBoost NB]
Accuracy: 0.66 (+/- 0.01) [AdaBoost LR]
```

In [54]:
```
result_df = pd.DataFrame(entries, columns=['Model_Id', 'Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Sc
models_df = result_df.groupby(['Model_Id','Model','Features','Hyper_Param', 'Best_Params']).agg(['mean'])
models_df.columns = models_df.columns.map('_'.join)
models_df
```

Out[54]:

Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score_mean
LR	AdaBoost LR	optimized	tuned		0.6626194
	Bagging LR	optimized	tuned		0.6831600
	LogisticRegression	baseline	default		0.7006732
		optimized	default		0.6999498
			tuned	{'C': 1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.6982669
NB	AdaBoost NB	optimized	tuned		0.5093467
	Bagging NB	optimized	tuned		0.7245268
	MultinomialNB	baseline	default		0.7007650
		optimized	default		0.7330550
			tuned	{'alpha': 0.5, 'fit_prior': True}	0.7374644
RF	AdaBoost RF	optimized	tuned		0.6489358
	Bagging RF	optimized	tuned		0.5844920
	RandomForestClassifier	baseline	default		0.5101938
		optimized	default		0.6940290
			tuned	{'max_depth': 5, 'n_estimators': 200, 'random_state': 3}	0.6332853
SVC	AdaBoost SVC	optimized	tuned		0.6452123
	Bagging SVC	optimized	tuned		0.6757373
	LinearSVC	baseline	default		0.6872884
		optimized	default		0.6582940
			tuned	{'C': 1500, 'dual': False, 'loss': 'squared_hinge', 'max_iter': 2000, 'penalty': 'l1'}	0.6115746

The Boosting ensemble did not provide any improvements on the baseline and optimized modeling.

Stacking

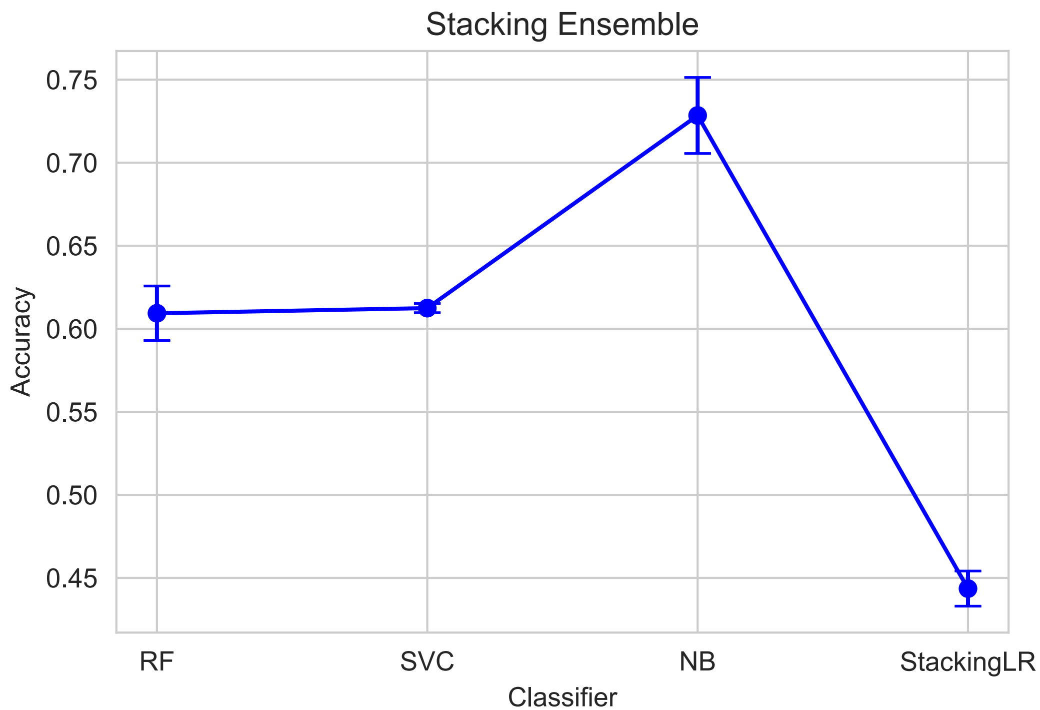
In [55]:

```
from mlxtend.classifier import StackingClassifier

sclf = StackingClassifier(classifiers=[clf1, clf2, clf3], meta_classifier=clf4)

labels = ['Random Forest', 'LinearSVC', 'MultinomialNB', 'Stacking LR']
clf_list = [clf1, clf2, clf3, sclf]

fig = plt.figure(figsize=(10,8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

clf_cv_mean = []
clf_cv_std = []
for clf, label, grd in zip(clf_list, labels, grid):

    scores = cross_val_score(clf, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='accuracy')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    if (label == 'Stacking LR'):
        entries.append(['Stack', label, 'optimized', 'tuned', '', scores.mean()])
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())
```

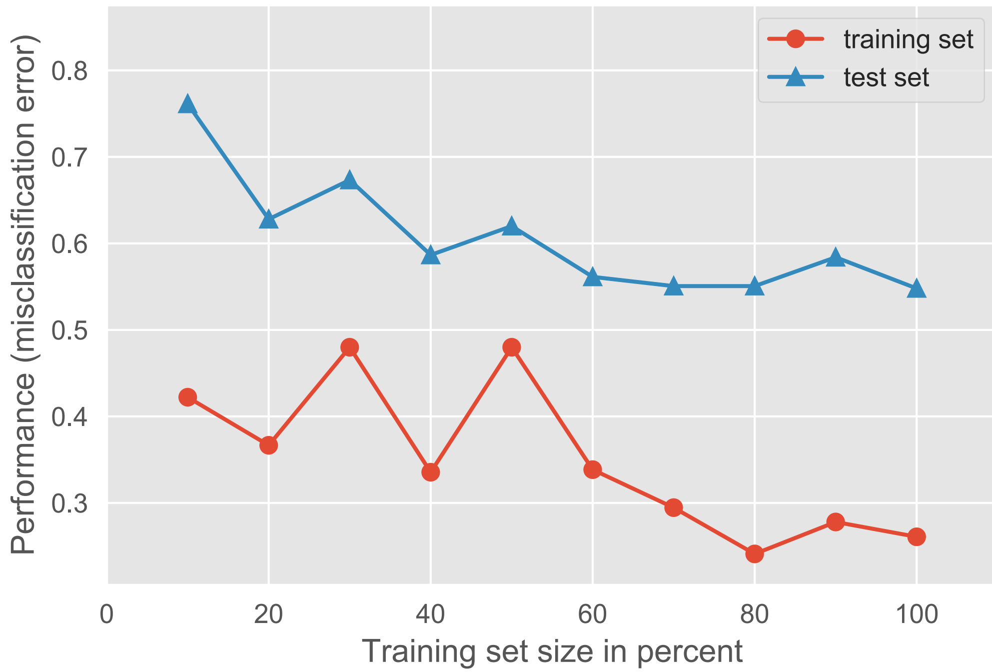
```
Accuracy: 0.61 (+/- 0.02) [Random Forest]
Accuracy: 0.61 (+/- 0.00) [LinearSVC]
Accuracy: 0.73 (+/- 0.02) [MultinomialNB]
Accuracy: 0.44 (+/- 0.01) [Stacking LR]
```

```
<Figure size 1000x800 with 0 Axes>
```

In [56]:
```
#plot classifier accuracy
plt.figure()
(_, caps, _) = plt.errorbar(range(4), clf_cv_mean, yerr=clf_cv_std, c='blue', fmt='-o', capsize=5)
for cap in caps:
    cap.set_markeredgewidth(1)
plt.xticks(range(4), ['RF', 'SVC', 'NB', 'StackingLR'])
plt.ylabel('Accuracy'); plt.xlabel('Classifier'); plt.title('Stacking Ensemble');
plt.show()
```



In [57]:
```
#plot Stacking learning curve
plt.figure()
plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, sclf, print_model=F
plt.show()
```



In [58]:
```
from mlxtend.classifier import StackingClassifier

sclf_bst = StackingClassifier(classifiers=[boosting1, boosting2, boosting3], meta_classifier=clf4)

labels = ['Boosted RF', 'Boosted SVC', 'Boosted NB', 'Stacking Boosted LR']
#clf_list = [clf1, clf2, clf3, sclf]
bst_list = [boosting1, boosting2, boosting3, sclf_bst]

fig = plt.figure(figsize=(10,8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

clf_cv_mean = []
clf_cv_std = []
for clf, label, grd in zip(bst_list, labels, grid):

    scores = cross_val_score(clf, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='accuracy')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    if (label == 'Stacking Boosted LR'):
        entries.append(['Boost_Stack', label, 'optimized', 'tuned', '', scores.mean()])
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())
```

```
Accuracy: 0.66 (+/- 0.01) [Boosted RF]
Accuracy: 0.65 (+/- 0.00) [Boosted SVC]
Accuracy: 0.57 (+/- 0.01) [Boosted NB]
Accuracy: 0.45 (+/- 0.04) [Stacking Boosted LR]

<Figure size 1000x800 with 0 Axes>
```

In [59]:
```
result_df = pd.DataFrame(entries, columns=['Model_Id', 'Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Sc
models_df = result_df.groupby(['Model_Id', 'Model','Features','Hyper_Param', 'Best_Params']).agg(['mean'])
models_df.columns = models_df.columns.map('_'.join)
models_df
```

Out[59]:

Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score_mean
Boost_Stack	Stacking Boosted LR	optimized	tuned		0.4546667
LR	AdaBoost LR	optimized	tuned		0.6626194
	Bagging LR	optimized	tuned		0.6831600
	LogisticRegression	baseline	default		0.7006732
		optimized	default		0.6999498
			tuned	{'C': 1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.6982669
NB	AdaBoost NB	optimized	tuned		0.5093467
	Bagging NB	optimized	tuned		0.7245268
	MultinomialNB	baseline	default		0.7007650
		optimized	default		0.7330550
			tuned	{'alpha': 0.5, 'fit_prior': True}	0.7374644
RF	AdaBoost RF	optimized	tuned		0.6489358
	Bagging RF	optimized	tuned		0.5844920
	RandomForestClassifier	baseline	default		0.5101938
		optimized	default		0.6940290
			tuned	{'max_depth': 5, 'n_estimators': 200, 'random_state': 3}	0.6332853
SVC	AdaBoost SVC	optimized	tuned		0.6452123
	Bagging SVC	optimized	tuned		0.6757373
	LinearSVC	baseline	default		0.6872884
		optimized	default		0.6582940
			tuned	{'C': 1500, 'dual': False, 'loss': 'squared_hinge', 'max_iter': 2000, 'penalty': 'l1'}	0.6115746

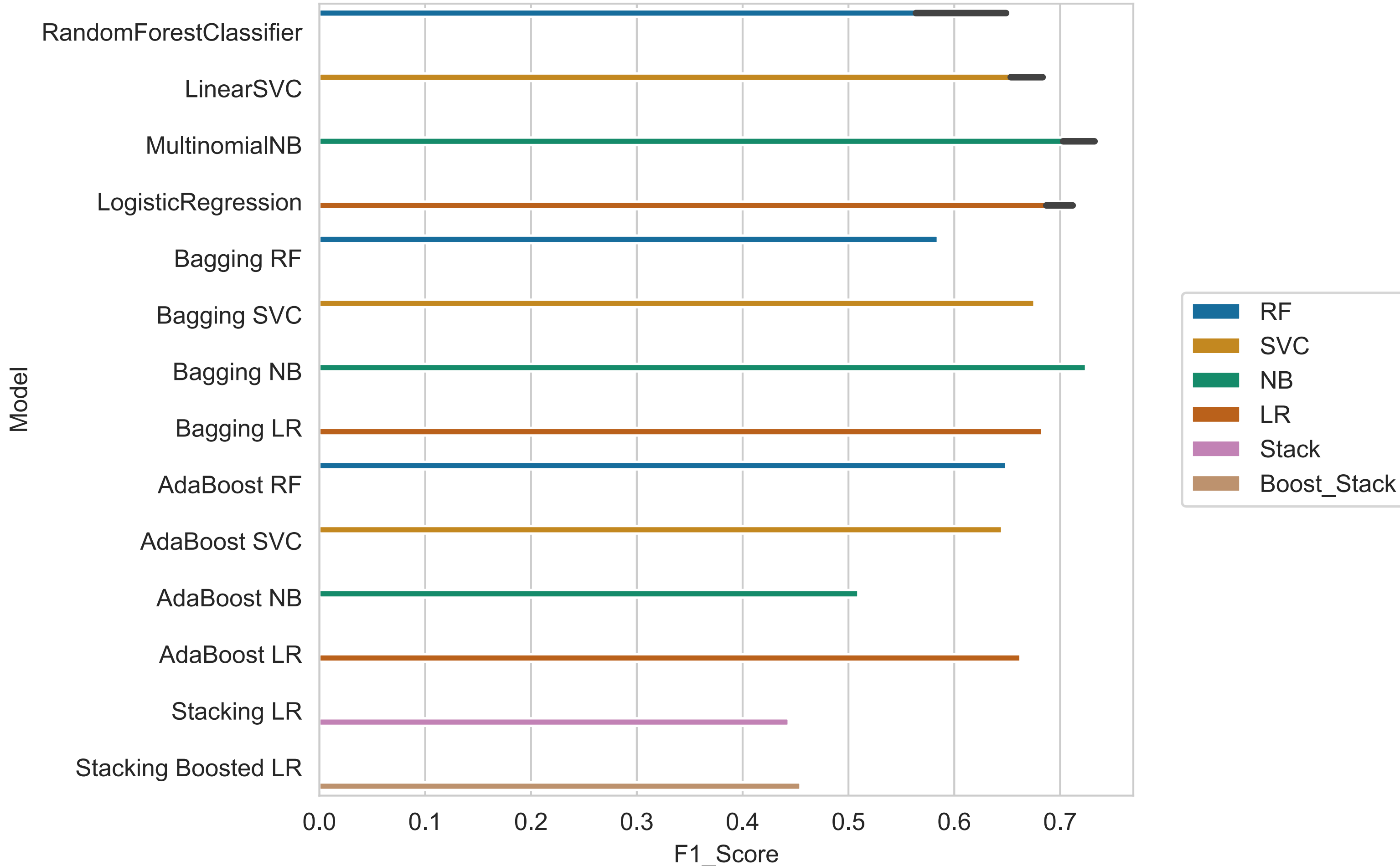
Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score_mean
Stack	**Stacking LR**	**optimized**	**tuned**		0.4435556

The Stacking performed poorly on our modeling.

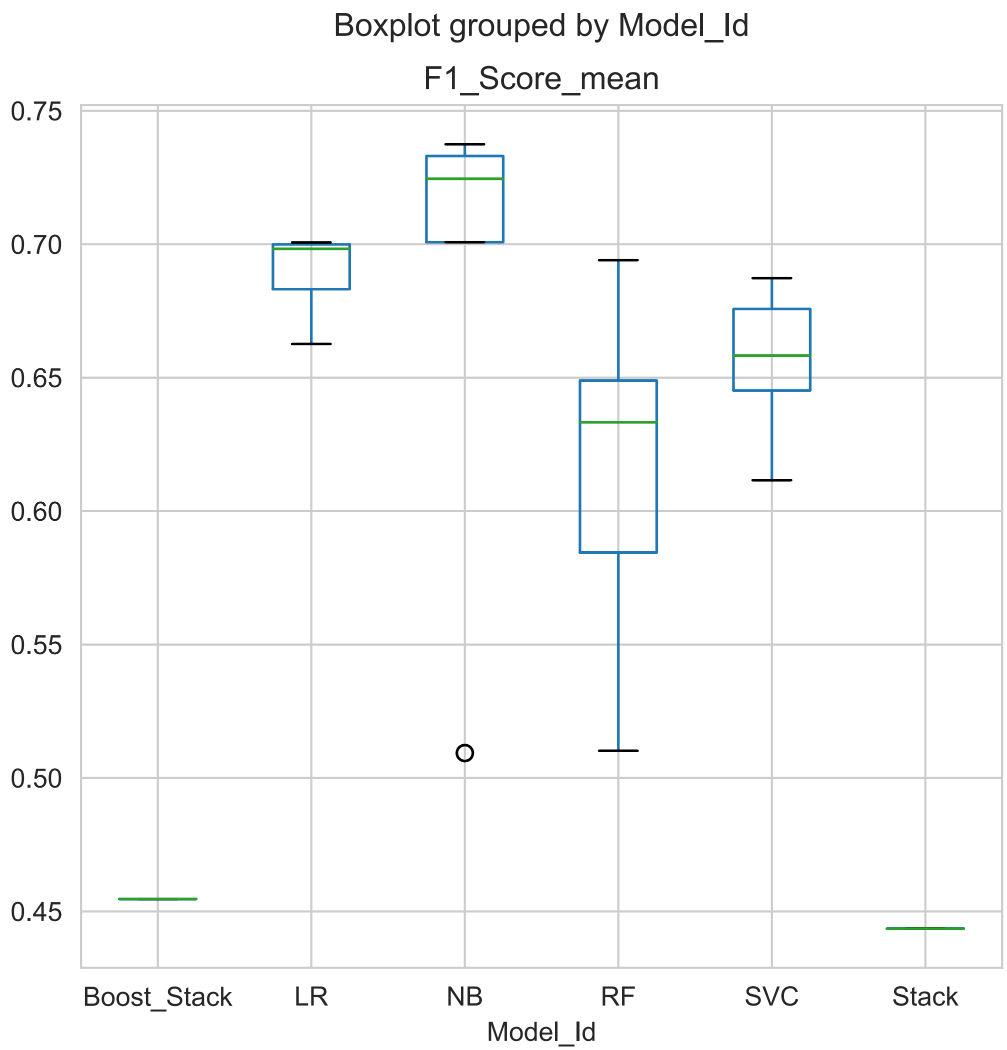
Summary of Findings

Benchmarking of F1 Scores of Models for Baseline, Bagging, Boosting and Stacking

```
In [105]: fig = plt.subplots(figsize=(6, 6))
g = sns.barplot(x='F1_Score', y='Model', data=result_df, palette="colorblind", hue='Model_Id')
g.legend(loc='center right', bbox_to_anchor=(1.35, 0.5), ncol=1);
```



In [99]:
```
models_df.boxplot(column=['F1_Score_mean'], by='Model_Id', figsize=(6, 6));
```



In [ ]: