Project Name: CSML1010 NLP Course Project - Part 1 - Proposal): Problem, Dataset, and Exploratory Data Analysis

Authors (Group3): Paul Doucet, Jerry Khidaroo

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

1. Problem Definition and Data Preparation Notebook:

This notebook will review the following sections as part of our project proposal:

Problem Definition

The problem we will be analysing is supervised text classification. The goal is to investigate which supervised mahine learning methods will give the best results in classifying the texts from our dataset into the pre-defined categories. This is a multi-class text classification problem. The input will be the text elements of each conversation concatenated together. The output will be the instruction_id.

Dataset Description

The dataset we will be using for our project is the **Taskmaster-1** dataset from Google. 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)

> The dataset consists of 13,215 task-based dialogs, including 5,507 spoken and 7,708 written dialogs created with two distinct procedures. Each conversation falls into one of six domains: ordering pizza, creating auto repair appointments, setting up ride service, ordering movie tickets, ordering coffee drinks and making restaurant reservations. Our initial data exploration will use the written dialog file with 7,708 records.

Roadmap

As part of our study, we will be consider the following steps to find the ideal classifier for incoming texts.

- **Feature Engineering:**
  - Count Vectors: Count Vector is a matrix notation of the dataset in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document. These provide no context, nor any consideration of the words in relation to other words or position in the sentence.
    - Bag-of-words
    - Bag of n-grams
  - TF-IDF Vectors: TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed by two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.
    - Word level Tfidf
    - N-gram Level TF-IDF
  - Word Embeddings: A word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. These generate a context free representation of each word in the vocabulary.
    - Word2vec
    - Glove
  - NLP Based features: An example of this would be Frequency distribution of Part of Speech Tags.
    - Noun, Verb, Adjective, Adverb, Pronoun Counts
  - Language Models: These are recent breakthroughs that provide context and generate a representation of each word based on other words in the sentence.
    - BERT or FLAIR
- **Model Training:**
  - Naive Bayes (multinominal): the one most suitable for word counts is multinominal.
  - logistic regression.
  - support vector machine.

    - decision tree (random forest).
    - Ensemble: Bagging, Boosting
- **Model Evaluation:**
    - Confusion Matrix
    - Metrics: Presicion, Recall, F1 Score

Import the Dataset

Two JSON format file we will be using from the **Taskmaster-1** dataset is the following:

- **self-dialogs.json** contains all the one-person dialogs.

This file can be divided into train/dev/test sets by matching the dialog IDs from the following files:

- train.csv
- dev.csv
- test.csv

Supplementary information is provided to describe the data structure and annotation schema.

- **sample.json** - A sample conversation describing the format of the data.
- **ontology.json** - Schema file describing the annotation ontology.

The structure of the conversations in the data files is as follows:

- **conversationId:** A universally unique identifier with the prefix 'dlg-'. The ID has no meaning.
- **utterances:** An array of utterances that make up the conversation.
- **instructionId:** A reference to the file(s) containing the user (and, if applicable, agent) instructions for this conversation.

The **utterances** category, has the following sub-categories of which we will be using the **text** to perform our analysis:

- **index:** A 0-based index indicating the order of the utterances in the conversation.
- **speaker:** Either USER or ASSISTANT, indicating which role generated this utterance.
- **text:** The raw text of the utterance. In case of self dialogs, this is written by the crowdsourced worker. In case of the WOz dialogs, 'ASSISTANT' turns are written and 'USER' turns are transcribed from the spoken recordings of crowdsourced workers.
- **segments:** An array of various text spans with semantic annotations.

Import the libraries

In [1]:
```
import json
import pandas as pd
from pandas.io.json import json_normalize
```

Open the `self-dialogs.json` file and view the entire content

In [2]:
```
with open(r'./data/self-dialogs.json') as f:
    data = json.load(f)
```

Extract the `utterances` column and normalize it to view all individual text fields.
This will increase the dataframe rows from 7708 to 169469 as each text field is now available

In [3]:
```
tt = pd.json_normalize(data, 'utterances', ['conversation_id','instruction_id'])
```

View the dataframe with the text field visible outside the dictionary

In [4]:
```
tt
```

Out[4]:

	index	speaker	text	segments	conversation_id	instruction_id
0	0	USER	Hi, I'm looking to book a table for Korean fod.	NaN	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
1	1	ASSISTANT	Ok, what area are you thinking about?	NaN	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
2	2	USER	Somewhere in Southern NYC, maybe the East Vill...	[{'start_index': 13, 'end_index': 49, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
3	3	ASSISTANT	Ok, great. There's Thursday Kitchen, it has g...	[{'start_index': 20, 'end_index': 35, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
4	4	USER	That's great. So I need a table for tonight at...	[{'start_index': 26, 'end_index': 31, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
...	...	...	...	...	...	...
169464	15	ASSISTANT	Ok.	NaN	dlg-fffa6565-32bb-4592-8d30-fff66df29633	movie-tickets-3
169465	16	USER	I think we'll pass for tonight. Thanks anyhow.	NaN	dlg-fffa6565-32bb-4592-8d30-fff66df29633	movie-tickets-3
169466	17	ASSISTANT	Ok. Just let me know if you change your mind.	NaN	dlg-fffa6565-32bb-4592-8d30-fff66df29633	movie-tickets-3
169467	18	USER	I will. Thanks	NaN	dlg-fffa6565-32bb-4592-8d30-fff66df29633	movie-tickets-3
169468	19	ASSISTANT	No problem!	NaN	dlg-fffa6565-32bb-4592-8d30-fff66df29633	movie-tickets-3

169469 rows × 6 columns

Remove all columns but the `text` and `conversation_id` from the dataframe and view

tt.drop('index', axis=1, inplace=True) tt.drop('segments', axis=1, inplace=True) tt.drop('speaker', axis=1, inplace=True) tt

View the columns of the dataframe

In [5]:
```
tt.columns
```

Out[5]:
```
Index(['index', 'speaker', 'text', 'segments', 'conversation_id',
       'instruction_id'],
      dtype='object')
```

View the content of the `text` column, then the `conversation_id`

In [6]:
```
tt['text']
```

Out[6]:
```
0            Hi, I'm looking to book a table for Korean fod.
1                      Ok, what area are you thinking about?
2          Somewhere in Southern NYC, maybe the East Vill...
3          Ok, great.  There's Thursday Kitchen, it has g...
4          That's great. So I need a table for tonight at...
                                ...                        
169464                                                   Ok.
169465        I think we'll pass for tonight. Thanks anyhow.
169466         Ok. Just let me know if you change your mind.
169467                                        I will. Thanks
169468                                           No problem!
Name: text, Length: 169469, dtype: object
```

In [7]:
```
tt['conversation_id']
```
Out[7]:
```
0         dlg-00055f4e-4a46-48bf-8d99-4e477663eb23
1         dlg-00055f4e-4a46-48bf-8d99-4e477663eb23
2         dlg-00055f4e-4a46-48bf-8d99-4e477663eb23
3         dlg-00055f4e-4a46-48bf-8d99-4e477663eb23
4         dlg-00055f4e-4a46-48bf-8d99-4e477663eb23
                            ...
169464    dlg-fffa6565-32bb-4592-8d30-fff66df29633
169465    dlg-fffa6565-32bb-4592-8d30-fff66df29633
169466    dlg-fffa6565-32bb-4592-8d30-fff66df29633
169467    dlg-fffa6565-32bb-4592-8d30-fff66df29633
169468    dlg-fffa6565-32bb-4592-8d30-fff66df29633
Name: conversation_id, Length: 169469, dtype: object
```

View of one line of the dataframe filtered by `conversation_id`

In [8]:
```
tt[tt.conversation_id == 'dlg-00055f4e-4a46-48bf-8d99-4e477663eb23']
```
Out[8]:

	index	speaker	text	segments	conversation_id	instruction_id
0	0	USER	Hi, I'm looking to book a table for Korean fod.	NaN	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
1	1	ASSISTANT	Ok, what area are you thinking about?	NaN	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
2	2	USER	Somewhere in Southern NYC, maybe the East Vill...	[{'start_index': 13, 'end_index': 49, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
3	3	ASSISTANT	Ok, great. There's Thursday Kitchen, it has g...	[{'start_index': 20, 'end_index': 35, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
4	4	USER	That's great. So I need a table for tonight at...	[{'start_index': 26, 'end_index': 31, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
5	5	ASSISTANT	They don't have any availability for 7 pm.	[{'start_index': 37, 'end_index': 41, 'text': ...	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2
6	6	USER	What times are available?	NaN	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	restaurant-table-2

Categorize the `conversation_id` TODO: confirm this step is necessary

In [9]:
```
tt2 = tt.conversation_id.unique()
```
In [10]:
```
tt2
```
Out[10]:
```
array(['dlg-00055f4e-4a46-48bf-8d99-4e477663eb23',
       'dlg-0009352b-de51-474b-9f13-a2b0b2481546',
       'dlg-00123c7b-15a0-4f21-9002-a2509149ee2d', ...,
       'dlg-ffcd1d53-c080-4acf-897d-48236513bc58',
       'dlg-ffd9db94-36e3-4534-b99d-89f7560db17c',
       'dlg-fffa6565-32bb-4592-8d30-fff66df29633'], dtype=object)
```

Verify the length of the `tt2` array to confirm the number of conversations: note that it should match initial dataframe length of 7708

In [11]:
```
len(tt2)
```
Out[11]:
```
7708
```

Loop thru the entire `tt2` dataframe and combine all the text based on the conversation_id

In [12]:
```
# Loop thru all the conversation_id unique values
#df = pd.DataFrame(columns=['Conversation', 'ident'])
conversation_id = []
conv_text = []
instr_id = []
for i in tt2:
    conv2 = ''
    tti = tt[tt.conversation_id == i]
    conv = ''
    conv2 = ''
    instr3 = tti['instruction_id']
    instr_id.append(instr3.iloc[0])
    for j in tti:
        conv = tti['text']
    for k in conv:
        conv2 = conv2 + k + " "
    conversation_id.append(i)
    conv_text.append(conv2)
```

In [13]:
```
# View the content of the concatenated conversation list, created by combining all 'text' fields per conversation_id
conv_text[0:5]
```

Out[13]:
```
["Hi, I'm looking to book a table for Korean fod. Ok, what area are you thinking about? Somewhere in Southern NYC, maybe t
he East Village? Ok, great.  There's Thursday Kitchen, it has great reviews. That's great. So I need a table for tonight a
t 7 pm for 8 people. We don't want to sit at the bar, but anywhere else is fine. They don't have any availability for 7 p
m. What times are available? 5 or 8. Yikes, we can't do those times. Ok, do you have a second choice? Let me check. Ok. Le
ts try Boka, are they free for 8 people at 7? Yes. Great, let's book that. Ok great, are there any other requests? No, tha
t's it, just book. Great, should I use your account you have open with them? Yes please. Great. You will get a confirmatio
n to your phone soon. ",
 "Hi I would like to see if the Movie What Men Want is playing here. Yes it's showing here would you like to purchase a ti
cket? Yes, for me and a friend so two tickets please Okay. What time is that moving playing today? That movie is showing a
t 4, 5, and 8pm. Okay. Is there anymore movies showing around 8pm Yes , showing at 8pm is Green Book. What is that about?
It's about two men dealing with racisim. Oh, no can you recommend anything else? What do you like? Well I like movies that
are funny. Like comedies? Well no I like action as well. Okay. How to train your dragon is playing at 8pm. Okay can i get
two tickets for that ? So you want me to cancel the tickets for What men want ? Yes please. Okay, no problem. How much wil
l this cost. You said two adult tickets? Yes. Okay, that will be $20.80 Okay. Anything else I can help you with ? Yes can
i bring my own food to theater. No, sorry you have to purchase food in the lobby. Okay that is fine. Thank you enjoy your
movie ",
 "I want to watch avengers endgame where do you want to watch it at? at bangkok close the hotel I a currently staying soun
ds good, what time do you want to watch the movie? 8 o'clock how many tickets? two and should we use the account we alread
y have with the movie theater? yes It seems they do not have any movie at that time let's watch another movie then what ot
```

In [14]:
```
# View the content of the conversation_id list, which will be used to merge with original dataframe to match up topics
conversation_id[0:5]
```

Out[14]:
```
['dlg-00055f4e-4a46-48bf-8d99-4e477663eb23',
 'dlg-0009352b-de51-474b-9f13-a2b0b2481546',
 'dlg-00123c7b-15a0-4f21-9002-a2509149ee2d',
 'dlg-0013673c-31c6-4565-8fac-810e173a5c53',
 'dlg-001d8bb1-6f25-4ecd-986a-b7eeb5fa4e19']
```

In [15]:
```
instr_id[0:5]
```

Out[15]:
```
['restaurant-table-2',
 'movie-tickets-1',
 'movie-tickets-3',
 'pizza-ordering-2',
 'pizza-ordering-2']
```

In [16]:
```
# Create a dictionary to store the conversation_id and text lists, which will be stored to a dataframe
ex_dict = {'id':conversation_id, 'conv':conv_text, 'instr':instr_id}
```

```
In [17]: # Create a dataframe with the conversation id and conversation
         df = pd.DataFrame(ex_dict)
         df.columns = ['id', 'Conversation','Instruction_id']
         df
```

Out[17]:

	id	Conversation	Instruction_id
0	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	Hi, I'm looking to book a table for Korean fod...	restaurant-table-2
1	dlg-0009352b-de51-474b-9f13-a2b0b2481546	Hi I would like to see if the Movie What Men W...	movie-tickets-1
2	dlg-00123c7b-15a0-4f21-9002-a2509149ee2d	I want to watch avengers endgame where do you ...	movie-tickets-3
3	dlg-0013673c-31c6-4565-8fac-810e173a5c53	I want to order a pizza from Bertuccis in Chel...	pizza-ordering-2
4	dlg-001d8bb1-6f25-4ecd-986a-b7eeb5fa4e19	Hi I'd like to order two large pizzas. Sure, w...	pizza-ordering-2
...	...	...	...
7703	dlg-ffc0c5fb-573f-40e0-b739-0e55d84100e8	I feel like eating at a nice restaurant tonigh...	restaurant-table-1
7704	dlg-ffc87550-389a-432e-927e-9a9438fc4f1f	Hi Sally, I need a Grande iced Americano with ...	coffee-ordering-2
7705	dlg-ffcd1d53-c080-4acf-897d-48236513bc58	Good afternoon. I would like to order a pizza ...	pizza-ordering-2
7706	dlg-ffd9db94-36e3-4534-b99d-89f7560db17c	Hey. I'm thinking of seeing What Men Want toni...	movie-tickets-1
7707	dlg-fffa6565-32bb-4592-8d30-fff66df29633	Hello. Can you help me purchase a couple of mo...	movie-tickets-3

```
In [18]: # View first three rows of the data frame conversation columns
         df['Conversation'][0:3]

Out[18]: 0    Hi, I'm looking to book a table for Korean fod...
         1    Hi I would like to see if the Movie What Men W...
         2    I want to watch avengers endgame where do you ...
         Name: Conversation, dtype: object

In [19]: # Export the dataframe to csv to confirm content
         df.to_csv(r'./data/DF_selfDialogs.csv', index=False)

In [ ]:
```

Project Name: CSML1010 NLP Course Project - Part 1 - Proposal): Problem, Dataset, and Exploratory Data Analysis

Authors (Group3): Paul Doucet, Jerry Khidaroo

2. Data Clean-up and NLP Notebook

This notebook will review the Data Cleaning tasks performed as part of our project proposal:

Categorize Groups

In [17]:
```
import pandas as pd
```

In [18]:
```
# Import CSV
df = pd.read_csv("./data/DF_selfDialogs.csv")
```

In [19]:
```
print (df.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    1161
coffee-ordering-1      735
coffee-ordering-2      641
movie-finder            54
movie-ticket-1          37
movie-tickets-1        642
movie-tickets-2        377
movie-tickets-3        195
pizza-ordering-1       257
pizza-ordering-2      1211
restaurant-table-1     704
restaurant-table-2     494
restaurant-table-3     102
uber-lyft-1            646
uber-lyft-2            452
dtype: int64
```

We need to fix the 37 movie-ticket-1 instruction_ids

In [4]:
```
df = df.replace(['movie-ticket-1'], 'movie-tickets-1')
```

In [5]:
```
print (df.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    1161
coffee-ordering-1      735
coffee-ordering-2      641
movie-finder            54
movie-tickets-1        679
movie-tickets-2        377
movie-tickets-3        195
pizza-ordering-1       257
pizza-ordering-2      1211
restaurant-table-1     704
restaurant-table-2     494
restaurant-table-3     102
uber-lyft-1            646
uber-lyft-2            452
dtype: int64
```

Add the Service Type as a column (i.e. auto, coffee, movie, etc.)

In [6]:
```
df['service_type'] = df['Instruction_id'].str.split('-',expand=True)[0]
print (df.groupby('service_type').size())
```

```
service_type
auto          1161
coffee        1376
movie         1305
pizza         1468
restaurant    1300
uber          1098
dtype: int64
```

In [7]:
```
df
```

Out[7]:

	id	Conversation	Instruction_id	service_type
0	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	Hi, I'm looking to book a table for Korean fod...	restaurant-table-2	restaurant
1	dlg-0009352b-de51-474b-9f13-a2b0b2481546	Hi I would like to see if the Movie What Men W...	movie-tickets-1	movie
2	dlg-00123c7b-15a0-4f21-9002-a2509149ee2d	I want to watch avengers endgame where do you ...	movie-tickets-3	movie
3	dlg-0013673c-31c6-4565-8fac-810e173a5c53	I want to order a pizza from Bertuccis in Chel...	pizza-ordering-2	pizza
4	dlg-001d8bb1-6f25-4ecd-986a-b7eeb5fa4e19	Hi I'd like to order two large pizzas. Sure, w...	pizza-ordering-2	pizza
...	...	...	...	...
7703	dlg-ffc0c5fb-573f-40e0-b739-0e55d84100e8	I feel like eating at a nice restaurant tonigh...	restaurant-table-1	restaurant
7704	dlg-ffc87550-389a-432e-927e-9a9438fc4f1f	Hi Sally, I need a Grande iced Americano with ...	coffee-ordering-2	coffee
7705	dlg-ffcd1d53-c080-4acf-897d-48236513bc58	Good afternoon. I would like to order a pizza ...	pizza-ordering-2	pizza
7706	dlg-ffd9db94-36e3-4534-b99d-89f7560db17c	Hey. I'm thinking of seeing What Men Want toni...	movie-tickets-1	movie
7707	dlg-fffa6565-32bb-4592-8d30-fff66df29633	Hello. Can you help me purchase a couple of mo...	movie-tickets-3	movie

7708 rows × 4 columns

Connect to Database

In [8]:
```
import sqlite3
con = sqlite3.connect('selfdialogs.db')
```

Cleaning the Dataset for NLP

Cleaning Function

In [9]:
```
import re
def clean(s):
    s = s.replace(r'<lb>', "\n")
    s = s.replace(r'<tab>', "\i")
    s = re.sub(r'<br */*>', "\n", s)
    s = s.replace("&lt;", "<").replace("&gt;", ">").replace("&amp;", "&")
    s = s.replace("&amp;", "&")
    # markdown urls
    s = re.sub(r'\(https*://[^\)]*\)', "", s)
    # normal urls
    s = re.sub(r'https*://[^\s]*', "", s)
    s = re.sub(r'_+', ' ', s)
    s = re.sub(r'"+', '"', s)
    return str(s)
```

In [10]:
```
df["selfdialog_clean"] = ''
```

Iterate and Clean

In [11]:
```
for i, row in df.iterrows():
    df.at[i, "selfdialog_clean"] = clean(row.Conversation)
```

In [12]:
```
df.head()
```
Out[12]:

	id	Conversation	Instruction_id	service_type	selfdialog_clean
0	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	Hi, I'm looking to book a table for Korean fod...	restaurant-table-2	restaurant	Hi, I'm looking to book a table for Korean fod...
1	dlg-0009352b-de51-474b-9f13-a2b0b2481546	Hi I would like to see if the Movie What Men W...	movie-tickets-1	movie	Hi I would like to see if the Movie What Men W...
2	dlg-00123c7b-15a0-4f21-9002-a2509149ee2d	I want to watch avengers endgame where do you ...	movie-tickets-3	movie	I want to watch avengers endgame where do you ...
3	dlg-0013673c-31c6-4565-8fac-810e173a5c53	I want to order a pizza from Bertuccis in Chel...	pizza-ordering-2	pizza	I want to order a pizza from Bertuccis in Chel...
4	dlg-001d8bb1-6f25-4ecd-986a-b7eeb5fa4e19	Hi I'd like to order two large pizzas. Sure, w...	pizza-ordering-2	pizza	Hi I'd like to order two large pizzas. Sure, w...

NLP

In [13]:
```
import spacy
nlp = spacy.load('en')
```

Iterate and Perform NLP

In [14]:
```
for i, row in df.iterrows():
    if i % 1000 == 0:
        print(i)
    if(row["selfdialog_clean"] and len(str(row["selfdialog_clean"])) < 1000000):
        doc = nlp(str(row["selfdialog_clean"]))
        adjectives = []
        nouns = []
        verbs = []
        lemmas = []

        for token in doc:
            lemmas.append(token.lemma_)
            if token.pos_ == "ADJ":
                adjectives.append(token.lemma_)
            if token.pos_ == "NOUN" or token.pos_ == "PROPN":
                nouns.append(token.lemma_)
            if token.pos_ == "VERB":
                verbs.append(token.lemma_)

        df.at[i, "selfdialog_lemma"] = " ".join(lemmas)
        df.at[i, "selfdialog_nouns"] = " ".join(nouns)
        df.at[i, "selfdialog_adjectives"] = " ".join(adjectives)
        df.at[i, "selfdialog_verbs"] = " ".join(verbs)
        df.at[i, "selfdialog_nav"] = " ".join(nouns+adjectives+verbs)
        df.at[i, "no_tokens"] = len(lemmas)
```

```
0
1000
2000
3000
4000
5000
6000
7000
```

In [15]:
```python
df.head()
```

Out[15]:

	id	Conversation	Instruction_id	service_type	selfdialog_clean	selfdialog_lemma	selfdialog_nouns	selfdialog_adjectives	selfdialog_verbs	se
0	dlg-00055f4e-4a46-48bf-8d99-4e477663eb23	Hi, I'm looking to book a table for Korean fod...	restaurant-table-2	restaurant	Hi, I'm looking to book a table for Korean fod...	hi , -PRON- be look to book a table for korean...	table fod area southern nyc east village thurs...	korean what great great great fine available s...	be look book be think have be need do want sit...	t
1	dlg-0009352b-de51-474b-9f13-a2b0b2481546	Hi I would like to see if the Movie What Men W...	movie-tickets-1	movie	Hi I would like to see if the Movie What Men W...	hi -PRON- would like to see if the movie what ...	movie what men ticket friend ticket time today...	what that funny -PRON- much -PRON- own fine -P...	would like see want be play be show would like...	
2	dlg-00123c7b-15a0-4f21-9002-a2509149ee2d	I want to watch avengers endgame where do you ...	movie-tickets-3	movie	I want to watch avengers endgame where do you ...	-PRON- want to watch avenger endgame where do ...	avenger endgame bangkok hotel time movie o'clo...	good what many other -PRON- new afraid interes...	want watch do want watch close stay sound do w...	b
3	dlg-0013673c-31c6-4565-8fac-810e173a5c53	I want to order a pizza from Bertuccis in Chel...	pizza-ordering-2	pizza	I want to order a pizza from Bertuccis in Chel...	-PRON- want to order a pizza from bertuccis in...	pizza bertuccis chelmsford ma what type pizza ...	what large different what large -PRON- large g...	want order would like understand will do have ...	pi ch wh
4	dlg-001d8bb1-6f25-4ecd-986a-b7eeb5fa4e19	Hi I'd like to order two large pizzas. Sure, w...	pizza-ordering-2	pizza	Hi I'd like to order two large pizzas. Sure, w...	hi -PRON- would like to order two large pizza ...	pizza kind pizza mind please anything meat lov...	large what hawaiian large sorry sure what -PRO...	would like order have will have be can get wou...	a

Store to Database

In [16]:
```python
df.to_sql('posts_nlp', con, if_exists='replace')
```

In [ ]:
```python

```

Project Name: CSML1010 NLP Course Project - Part 1 - Proposal): Problem, Dataset, and Exploratory Data Analysis

Authors (Group3): Paul Doucet, Jerry Khidaroo

3. Data Exploration Notebook

This notebook will review the Data Exploration tasks performed as part of our project proposal:

Load the Dataframe

In [1]:
```
# filter warnings on depreciation etc.
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# import pandas, numpy
import pandas as pd
import numpy as np

# adjust pandas display
pd.options.display.max_columns = 30
pd.options.display.max_rows = 100
pd.options.display.float_format = '{:.2f}'.format
pd.options.display.precision = 2
pd.options.display.max_colwidth = -1
```

In [3]:
```
# 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")
```

Load Data

In [4]:
```
import sqlite3

sql = """
SELECT p.*
FROM posts_nlp p
"""

with sqlite3.connect('selfdialogs.db') as con:
    df = pd.read_sql_query(sql, con)
```

Data Exploration

```
In [5]: # list column names and datatypes
        df.dtypes
```

```
Out[5]: index                     int64
        id                       object
        Conversation             object
        Instruction_id           object
        service_type             object
        selfdialog_clean         object
        selfdialog_lemma         object
        selfdialog_nouns         object
        selfdialog_adjectives    object
        selfdialog_verbs         object
        selfdialog_nav           object
        no_tokens               float64
        dtype: object
```

```
In [6]: # select a sample of some data frame columns
        df[['id', 'Conversation', 'Instruction_id','service_type']] \
          .sample(2, random_state=42)
```

Out[6]:

	id	Conversation	Instruction_id	service_type
6482	dlg-d601e9c1-f9b4-4778-ae20-29f5ab6edd72	Hello can you please book a reservation at the crawling crab for tonight at 7:30 for 3 people There is no table available at that time how about 8:00 pm Ok That table is outside is that ok No that won't work Ok is there another restaurant that you would like to try Yes how about the crying tree Same Time and party? Yes Ok I will book the table would you like the restaurant to send u a text confirmation Yes Ok what phone number 867 5309 Ok they will text u around 15 minutes before the table is ready to confirm the table Ok Do you want to order drinks to have ready when you arrive. Yes please order 2 glasses of red wine Ok they will have your drinks ready Ok To confirm I have a table for 3 ready at the crying tree for today at 7:30 wine will be ordered prior to arriving they will text to confirm is this correct Yes Ok your good to go	restaurant-table-2	restaurant
6872	dlg-e3797e80-a033-47d3-be9f-3235ea00f09c	Hi, I would like for you to order a car for me From where would you like to leave? Bank of America Stadium, South Mint Street, Charlotte, NC And where will you be going? Romare Bearden Park, South Church Street, Charlotte, NC An UberX is 6.65IsUberXLavailable?UberXLisavailablefor7.75.Arethereanyotheroptions?Blackisavailablefor15.00andBlackSUVisavailablefor25.00IwouldliketobookUberXL.YouwouldliketobookUberXLfor7.75?Yes, that'scorrect.OkIambookingyourUberXLnow.Thankyou.Doyouhaveanyotherrequests?Howmuchwasthetotalintheend?Itwas 7.75. When can I expect my UberXL to arrive? Your ride is on the way and you can check your status on your phone. Thanks!	uber-lyft-1	uber

```
In [7]: # length of a dataframe
        len(df)
```

```
Out[7]: 7708
```

```
In [8]: # number of values per column
        df.count()
```

```
Out[8]: index                    7708
        id                       7708
        Conversation             7708
        Instruction_id           7708
        service_type             7708
        selfdialog_clean         7708
        selfdialog_lemma         7708
        selfdialog_nouns         7708
        selfdialog_adjectives    7708
        selfdialog_verbs         7708
        selfdialog_nav           7708
        no_tokens                7708
        dtype: int64
```

In [9]:
```
# size info, including memory consumption
df.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7708 entries, 0 to 7707
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   index                  7708 non-null   int64
 1   id                     7708 non-null   object
 2   Conversation           7708 non-null   object
 3   Instruction_id         7708 non-null   object
 4   service_type           7708 non-null   object
 5   selfdialog_clean       7708 non-null   object
 6   selfdialog_lemma       7708 non-null   object
 7   selfdialog_nouns       7708 non-null   object
 8   selfdialog_adjectives  7708 non-null   object
 9   selfdialog_verbs       7708 non-null   object
 10  selfdialog_nav         7708 non-null   object
 11  no_tokens              7708 non-null   float64
dtypes: float64(1), int64(1), object(10)
memory usage: 36.8 MB
```

Column Exploration

In [10]:
```
columns = [col for col in df.columns if not col.startswith('self')]
columns
```

Out[10]:
```
['index', 'id', 'Conversation', 'Instruction_id', 'service_type', 'no_tokens']
```

In [11]:
```
# describe categorical columns of type np.object
df[['service_type','Instruction_id']] \
  .describe(include=np.object) \
  .transpose()
```

Out[11]:

	count	unique	top	freq
service_type	7708	6	pizza	1468
Instruction_id	7708	14	pizza-ordering-2	1211

In [12]:
```
df['Instruction_id'].value_counts()[:10]
```

Out[12]:
```
pizza-ordering-2      1211
auto-repair-appt-1    1161
coffee-ordering-1     735
restaurant-table-1    704
movie-tickets-1       679
uber-lyft-1           646
coffee-ordering-2     641
restaurant-table-2    494
uber-lyft-2           452
movie-tickets-2       377
Name: Instruction_id, dtype: int64
```

In [13]:
```
# describe numerical columns
df.describe().transpose()
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
index	7708.00	3853.50	2225.25	0.00	1926.75	3853.50	5780.25	7707.00
no_tokens	7708.00	228.51	80.57	20.00	175.00	215.00	267.00	1336.00

In [14]:
```python
# group by service_type, count distinct Instruction_id
cat_df = df.groupby('service_type') \
           .agg({'Instruction_id': pd.Series.nunique,
                 'id': pd.Series.count}) \
           .rename(columns={'Instruction_id': 'num_Instruction_ids',
                            'id': 'num_posts'}) \
           .sort_values('num_Instruction_ids', ascending=False)

# show top 5 records
cat_df.head(5)
```

Out[14]:

service_type	num_Instruction_ids	num_posts
movie	4	1305
restaurant	3	1300
coffee	2	1376
pizza	2	1468
uber	2	1098

In [15]:
```python
# group by service_type, count distinct Instruction_id
cat_id_df = df.groupby('Instruction_id') \
           .agg({'id': pd.Series.count}) \
           .rename(columns={'id': 'num_posts'}) \
           .sort_values('num_posts', ascending=False)

# show top 5 records
cat_id_df.head(5)
```

Out[15]:

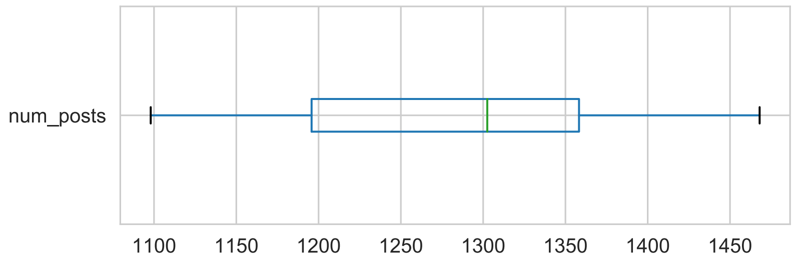
Instruction_id	num_posts
pizza-ordering-2	1211
auto-repair-appt-1	1161
coffee-ordering-1	735
restaurant-table-1	704
movie-tickets-1	679

In [16]:
```python
cat_df.describe()
```

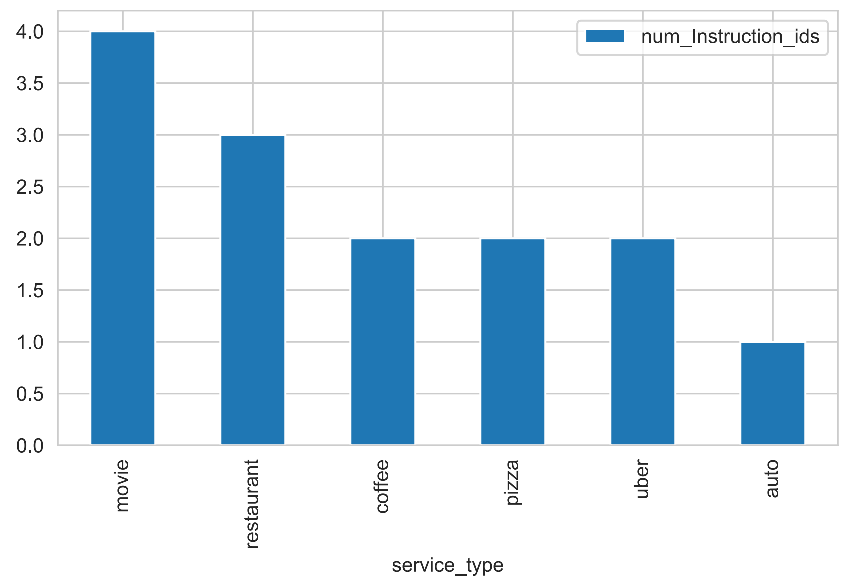
Out[16]:

	num_Instruction_ids	num_posts
count	6.00	6.00
mean	2.33	1284.67
std	1.03	136.19
min	1.00	1098.00
25%	2.00	1195.75
50%	2.00	1302.50
75%	2.75	1358.25
max	4.00	1468.00

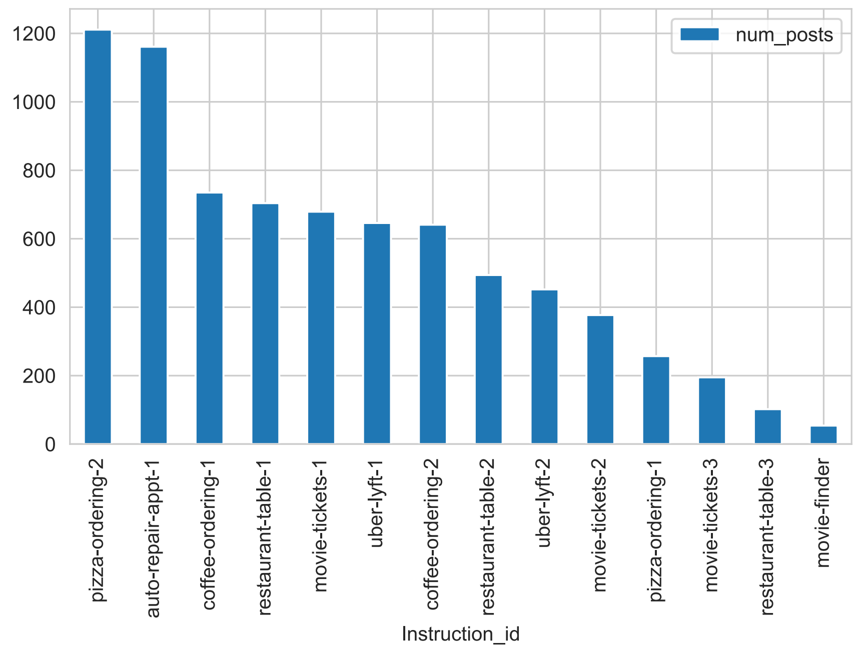
In [17]:
```python
# horizontal boxplot of a dataframe column
cat_df[['num_posts']].plot(kind='box', vert=False, figsize=(6, 2));
```



In [18]:
```
# bar chart of a dataframe column
cat_df[['num_Instruction_ids']].plot(kind='bar', figsize=(7,4));
```



In [19]:
```
# bar chart of a dataframe column
cat_id_df[['num_posts']].plot(kind='bar', figsize=(7,4));
```



Word Exploration

```
In [20]: # create a data frame slice
         sub_df = df[df['Instruction_id']=='movie-finder']

         # sample cleaned text and tokens tagged as nouns
         sub_df[['selfdialog_clean', 'selfdialog_nouns']].sample(2)
```

Out[20]:

	selfdialog_clean	selfdialog_nouns
2157	I want a movie to watch What are you in a mood for? I don't know. Maybe something with a lot of explosion So action genre suits you tonight? Yeah, action or adventure Any preferences on the actors or directors? Not really, but something recent. Recent being from the last decade? Yeah, nothing too old. How about from the past 10 years? Yes, that's good. I have many movies that fit the criteria. I need some more information Find me something Keanu Reeves did in the past 10 years. He has made many action movies in the past ten years. The most recent being John Wick 2. I don't know I didn't like John Wick, the original one How about 47 ronin. Its ratings is not too high, around 6.3 Ok. That's not my typical preference, but let's try that Ok. 47 ronin it is	movie what mood something lot explosion action genre suit tonight action adventure preference actor director something decade nothing year movie criterion information something keanu reeves year action movie year john wick john wick one ronin rating preference ronin
5910	I'm looking for a movie to watch tonight. Certainly, what genre are you looking for? Either action or comedy. Well theres a few movies that include both. Really? Which ones? Have you ever seen Rush Hour? No, I can't say that I have. Who stars in it. It has Jackie Chan and Chris Tucker. Does it have good reviews? Yes, it is considered a cult classic. Is it just one movie? No, Rush Hour as 2 sequels you can watch too. Are they as good as the first? Based on your preferences, I would say you wouldn't like them as much. Is there any dvd extras to Rush Hour? Yes, the dvd comes packed with extras. Great! Guess I know what I'm watching tonight. Thanks! You're welcome! Happy to help.	movie tonight genre action comedy movie one rush hour who jackie chan chris tucker review cult movie rush hour sequel preference dvd rush hour dvd extra what tonight thank

Creating Token List

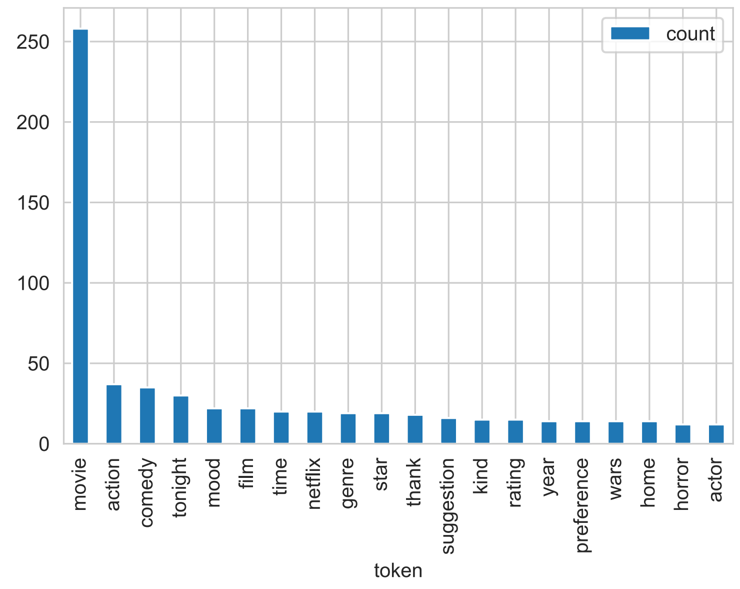
```
In [21]: def my_tokenizer(text):
             return text.split() if text != None else []

In [22]: # transform list of documents into a single list of tokens
         tokens = sub_df.selfdialog_nouns.map(my_tokenizer).sum()

In [23]: from collections import Counter

         counter = Counter(tokens)
         counter.most_common(20)

Out[23]: [('movie', 258),
          ('what', 82),
          ('something', 57),
          ('action', 37),
          ('comedy', 35),
          ('tonight', 30),
          ('one', 26),
          ('mood', 22),
          ('film', 22),
          ('time', 20),
          ('netflix', 20),
          ('genre', 19),
          ('star', 19),
          ('anything', 18),
          ('thank', 18),
          ('suggestion', 16),
          ('kind', 15),
          ('rating', 15),
          ('year', 14),
          ('preference', 14)]

In [24]: df.service_type.unique()

Out[24]: array(['restaurant', 'movie', 'pizza', 'coffee', 'auto', 'uber'],
               dtype=object)
```

In [25]:
```python
print([t[0] for t in counter.most_common(200)])
```

```
['movie', 'what', 'something', 'action', 'comedy', 'tonight', 'one', 'mood', 'film', 'time', 'netflix', 'genre', 'star', 'an
ything', 'thank', 'suggestion', 'kind', 'rating', 'year', 'preference', 'wars', 'home', 'horror', 'actor', 'hour', 'ticket',
'assistant', 'problem', 'sci', 'fi', 'sir', 'list', 'who', 'theater', 'recommendation', 'day', 'lot', 'scifi', 'world', 'typ
e', 'drama', 'imdb', 'fan', 'mind', 'tom', 'y', 'jedi', 'night', 'john', 'black', 'minute', 'trailer', 'episode', 'alien',
'romance', 'thriller', 'documentary', 'choice', 'place', 'help', 'nothing', 'book', 'amazon', 'name', 'return', 'wick', 'pan
ther', 'popcorn', 'rush', 'director', 'blade', 'runner', 'release', 'master', 'jurassic', 'hanks', 'show', 'marvel', 'missio
n', 'classic', 'quiet', 'cast', 'box', 'review', 'showing', 'text', 'adam', 'option', 'way', 'adventure', 'empire', 'galax
y', 'war', 'rosemary', 'baby', 'demand', 'kung', 'fu', 'man', 'avengers', 'x', 'men', 'weather', 'fantasy', 'theme', 'spac
e', 'hero', 'crime', 'wife', 'today', 'christmas', 'airplane', 'month', 'incredibles', 'thing', 'table', 'ready', 'player',
'people', 'matrix', 'bit', 'superhero', 'sandler', 'deadpool', 'plotline', 'alright', 'question', 'service', 'mystery', 'sci
ence', 'other', 'thor', 'art', 'jackie', 'chan', 'yoga', 'kingdom', 'great', 'title', 'infinity', 'link', 'watch', 'chris',
'fiction', 'got', 'mail', 'ryan', 'travel', 'angry', 'story', 'wave', 'idea', 'kong', 'island', 'crazy', 'rich', 'asians',
'jon', 'bernthal', 'matter', 'sequel', 'category', 'family', 'earth', 'big', 'violence', 'mrs.', 'stadium', 'style', 'seatin
g', 'guy', 'body', 'part', '1990', 'true', 'lies', 'hobbit', 'laura', 'sure', 'second', "o'clock", 'half', 'actress', 'set
h', 'rogan', 'acting', 'lead', 'buddy', 'cop', 'drug', 'keanu', 'ronin', 'iv', 'new', 'hope', 'harrison', 'ford', 'back', 'j
ames', 'cameron']
```

In [26]:
```python
from spacy.lang.en.stop_words import STOP_WORDS

def remove_stopwords(tokens):
    """Remove stopwords from a list of tokens."""
    return [t for t in tokens if t not in STOP_WORDS]

# rebuild counter
counter = Counter(remove_stopwords(tokens))
```

In [27]:
```python
# convert list of tuples into data frame
freq_df = pd.DataFrame.from_records(counter.most_common(20),
                                    columns=['token', 'count'])

# create bar plot
freq_df.plot(kind='bar', x='token');
```

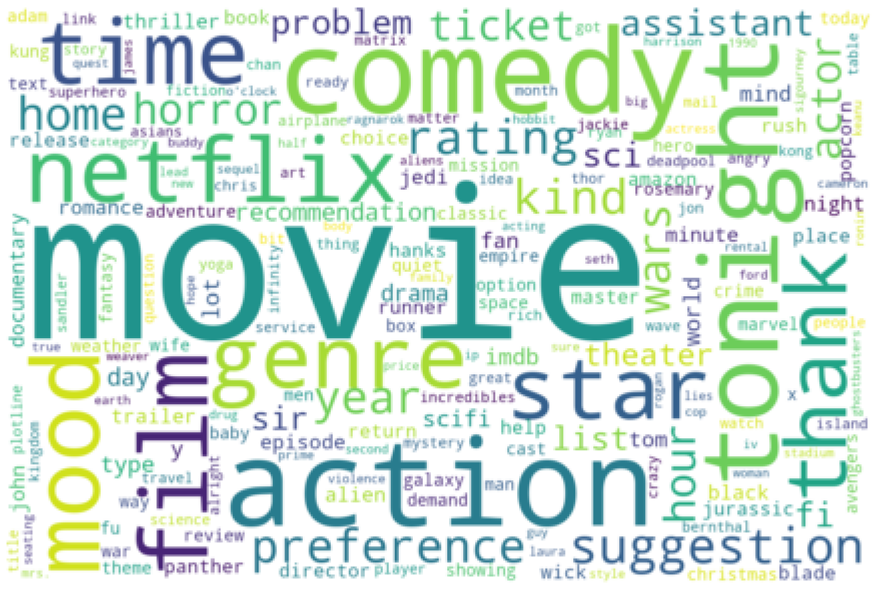


Exploring Word Clouds

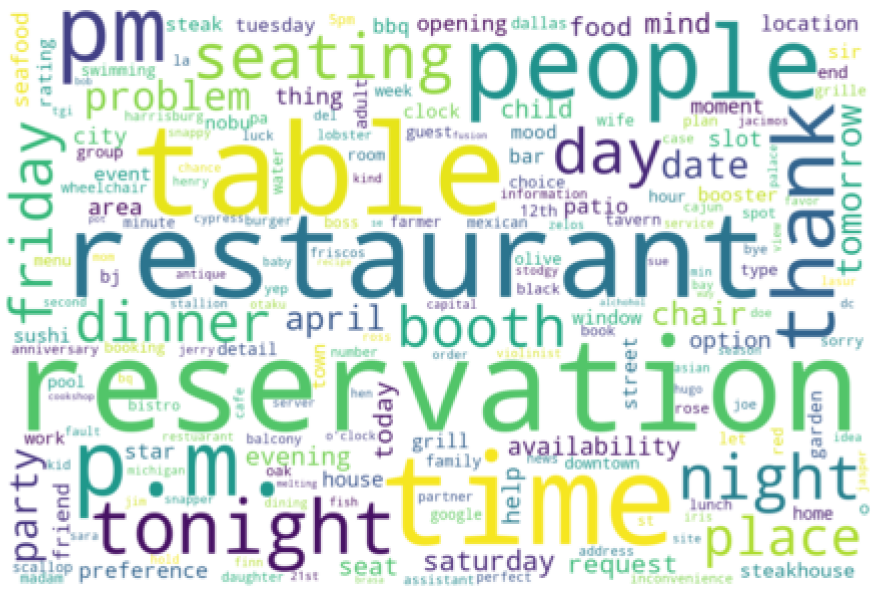
In [28]:
```python
%matplotlib inline
import matplotlib.pyplot as plt
```

```
In [29]: from wordcloud import WordCloud

         def wordcloud(counter):
             """A small wordloud wrapper"""
             wc = WordCloud(width=1200, height=800,
                            background_color="white",
                            max_words=200)
             wc.generate_from_frequencies(counter)

             # Plot
             fig=plt.figure(figsize=(6, 4))
             plt.imshow(wc, interpolation='bilinear')
             plt.axis("off")
             plt.tight_layout(pad=0)
             plt.show()
```

```
In [30]: # create wordcloud
         wordcloud(counter)
```



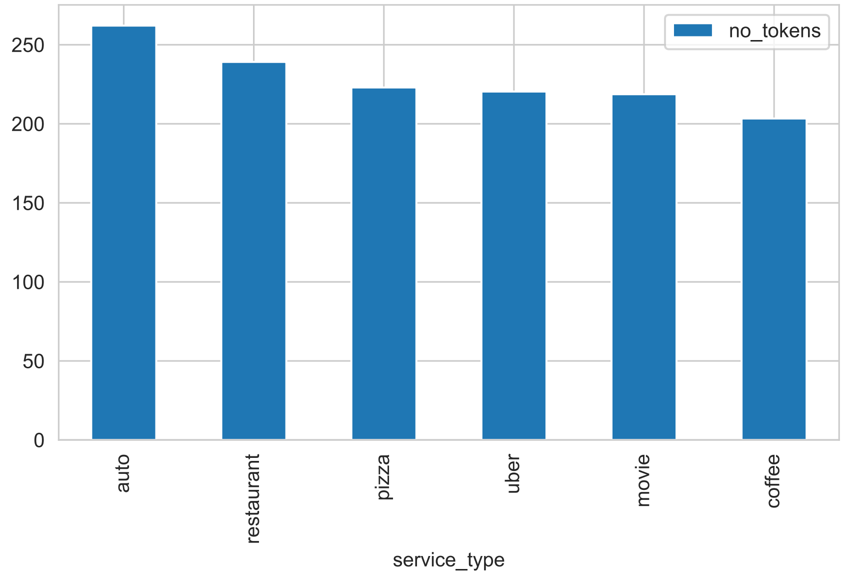
```
In [31]: tokens2 = df[df['Instruction_id']=='restaurant-table-3'].selfdialog_nouns \
                 .map(my_tokenizer).sum()

         counter2 = Counter(remove_stopwords(tokens2))
         wordcloud(counter2)
```



Exploring Complexity

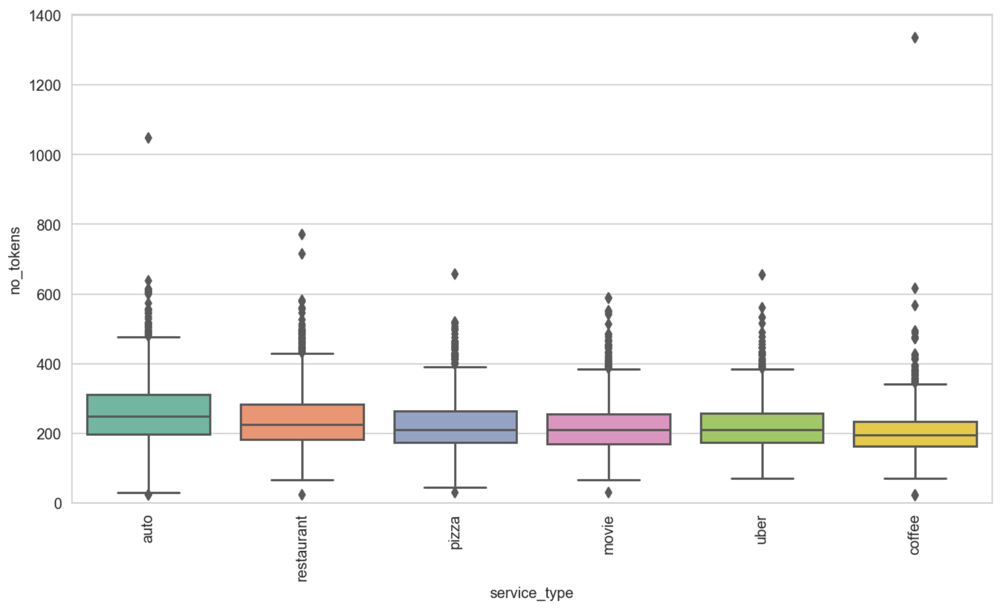
In [32]:
```
df['no_tokens'] = df.selfdialog_lemma\
  .map(lambda l: 0 if l==None else len(l.split()))
```

In [33]:
```
# mean number of tokens by category
df.groupby(['service_type']) \
  .agg({'no_tokens':'mean'}) \
  .sort_values(by='no_tokens', ascending=False) \
  .plot(kind='bar', figsize=(7,4));
```



In [34]:
```
# render plots as retina or png, because svg is very slow
%config InlineBackend.figure_format = 'retina'

import seaborn as sns

def multi_boxplot(data, x, y, ylim = None):
    '''Wrapper for sns boxplot with cut-off functionality'''
    # plt.figure(figsize=(30, 5))
    fig, ax = plt.subplots()
    plt.xticks(rotation=90)

    # order boxplots by median
    ordered_values = data.groupby(x)[[y]] \
                         .median() \
                         .sort_values(y, ascending=False) \
                         .index

    sns.boxplot(x=x, y=y, data=data, palette='Set2',
                order=ordered_values)

    fig.set_size_inches(11, 6)

    # cut-off y-axis at value ylim
    ax.set_ylim(0, ylim)
```

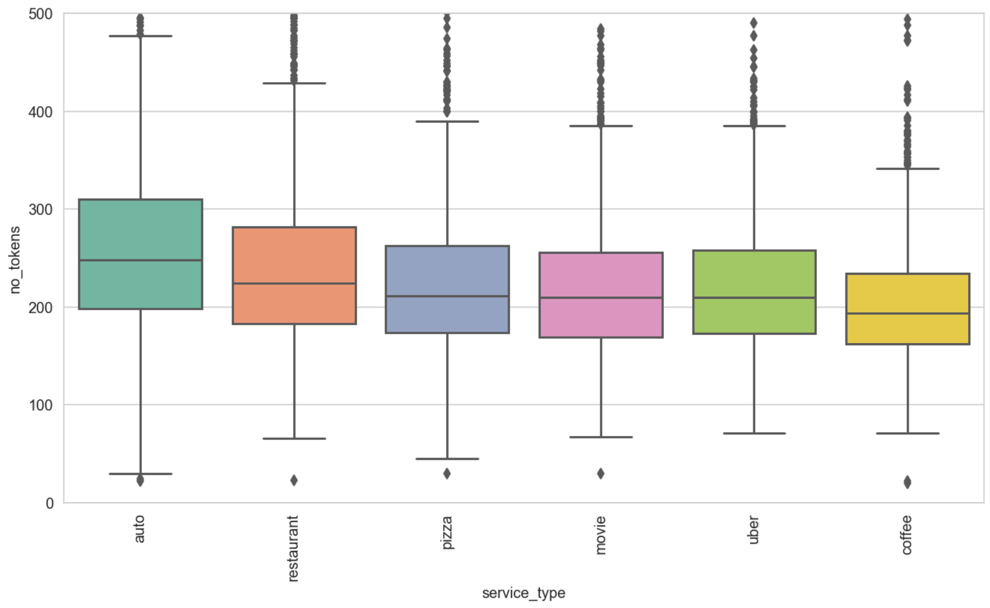
In [35]:
```
multi_boxplot(df, 'service_type', 'no_tokens');
```



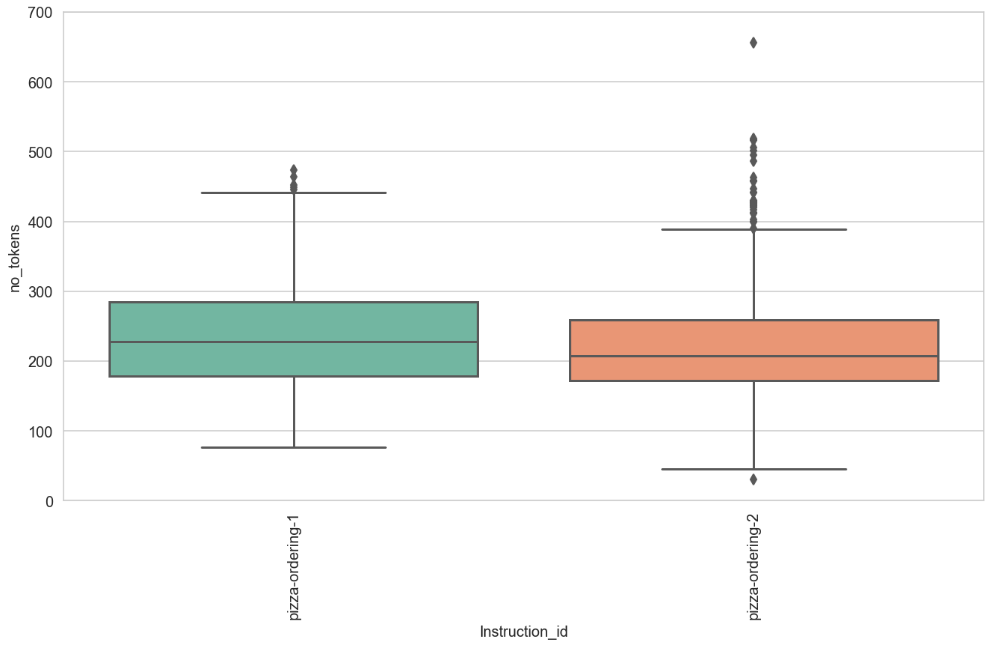
In [36]:
```
# print text of outliers
df['selfdialog_lemma'][df.no_tokens > 1500]
```

Out[36]:
```
Series([], Name: selfdialog_lemma, dtype: object)
```

```
In [37]: # cut-off diagram at y=500
         multi_boxplot(df, 'service_type', 'no_tokens', ylim=500)
```



```
In [38]: # comparing Instruction_id within a single service_type
         multi_boxplot(df[df.service_type=='pizza'],
                       'Instruction_id', 'no_tokens', ylim=700)
```



```
In [ ]:
```

CSML1010 Group3 Course_Project - Milestone 1 - Feature Engineering and Selection

Authors (Group3): Paul Doucet, Jerry Khidaroo

Project Repository: 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/)

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

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")
```

1. Data Preparation

Load Data

```
In [3]: import sqlite3

        sql = """
        SELECT p.selfdialog_clean, p.instruction_id
        FROM posts_nlp p
        """

        with sqlite3.connect('selfdialogs.db') as con:
            df_all = pd.read_sql_query(sql, con)
```

Merge some Classes that are very similiar to each other

In [4]:
```
# df_all[df_all["Instruction_id"] == 'coffee-ordering-1'] = 'coffee-ordering'
# df_all[df_all["Instruction_id"] == 'coffee-ordering-2'] = 'coffee-ordering'
# df_all[df_all["Instruction_id"] == 'pizza-ordering-1'] = 'pizza-ordering'
# df_all[df_all["Instruction_id"] == 'pizza-ordering-2'] = 'pizza-ordering'
# df_all[df_all["Instruction_id"] == 'restaurant-table-1'] = 'restaurant-table'
# df_all[df_all["Instruction_id"] == 'restaurant-table-2'] = 'restaurant-table'
# df_all[df_all["Instruction_id"] == 'uber-lyft-1'] = 'uber-lyft'
# df_all[df_all["Instruction_id"] == 'uber-lyft-2'] = 'uber-lyft'

df_all = df_all.replace(['coffee-ordering-1'], 'coffee-ordering')
df_all = df_all.replace(['coffee-ordering-2'], 'coffee-ordering')
df_all = df_all.replace(['pizza-ordering-1'], 'pizza-ordering')
df_all = df_all.replace(['pizza-ordering-2'], 'pizza-ordering')
df_all = df_all.replace(['restaurant-table-1'], 'restaurant-table')
df_all = df_all.replace(['restaurant-table-2'], 'restaurant-table')
df_all = df_all.replace(['uber-lyft-1'], 'uber-lyft')
df_all = df_all.replace(['uber-lyft-2'], 'uber-lyft')

print (df_all.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    1161
coffee-ordering       1376
movie-finder            54
movie-tickets-1        679
movie-tickets-2        377
movie-tickets-3        195
pizza-ordering        1468
restaurant-table      1198
restaurant-table-3     102
uber-lyft             1098
dtype: int64
```

Create Factorized 'category' column from 'Instruction_id' label column.

In [5]:
```
df_all['category'] = df_all['Instruction_id'].factorize()[0]
```

In [6]:
```
df_all.columns
```

Out[6]:
```
Index(['selfdialog_clean', 'Instruction_id', 'category'], dtype='object')
```

Do Some Additional CLeaning

In [7]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    #doc = "'" + doc + "'"
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    #doc = [[word.lower() for word in sent if word not in remove_terms] for sent in doc]
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)

    return doc

normalize_corpus = np.vectorize(normalize_document)
```

```
In [8]: for i, row in df_all.iterrows():
            df_all.at[i, "selfdialog_norm"] = normalize_corpus(row.selfdialog_clean)

        df_all = df_all.filter(['Instruction_id', 'category', 'selfdialog_norm'], axis=1)

        df_all.head(3)
```

Out[8]:

	Instruction_id	category	selfdialog_norm
0	restaurant-table	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

```
In [9]: df_all.columns

Out[9]: Index(['Instruction_id', 'category', 'selfdialog_norm'], dtype='object')
```

Remove NaN rows

```
In [10]: print(df_all.shape)
         df_all = df_all.dropna()
         df_all = df_all.reset_index(drop=True)
         df_all = df_all[df_all.selfdialog_norm != '']
         print(df_all.shape)

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

Save New Cleaned File

```
In [11]: df_all.to_csv('./data/dialog_norm.csv', index=False)
         #df_all.to_sql('dialog_norm', con, if_exists='replace')
```

Get a Sample of records.

In [12]:
```
# class_sample_size_dict = {
#     "auto-repair-appt-1": 191,
#     "coffee-ordering-1": 96,
#     "coffee-ordering-2": 97,
#     "movie-finder": 54,
#     "movie-tickets-1": 193,
#     "movie-tickets-2": 193,
#     "movie-tickets-3": 195,
#     "pizza-ordering-1": 96,
#     "pizza-ordering-2": 97,
#     "restaurant-table-1": 96,
#     "restaurant-table-2": 97,
#     "restaurant-table-3": 102,
#     "uber-lyft-1": 96,
#     "uber-lyft-2": 97
# }
# sum(class_sample_size_dict.values())
class_sample_size_dict = {
    "auto-repair-appt-1": 230,
    "coffee-ordering": 230,
    "movie-finder": 54,
    "movie-tickets-1": 250,
    "movie-tickets-2": 250,
    "movie-tickets-3": 195,
    "pizza-ordering": 230,
    "restaurant-table": 230,
    "restaurant-table-3": 101,
    "uber-lyft": 230
}
sum(class_sample_size_dict.values())
```

Out[12]:
```
2000
```

In [13]:
```
# cat_id_df = df_all[['Instruction_id', 'category']].drop_duplicates().sort_values('category')
# cat_count = len(cat_id_df)
# sample_size = 1000
# sample_per_cat = sample_size//cat_count
# print('sample_size: ', sample_size, 'sample_per_cat: ', sample_per_cat)
```

In [14]:
```
# Function to Get balanced Sample - Get a bit more than needed then down sample
def sampling_k_elements(group):
    name = group['Instruction_id'].iloc[0]
    k = class_sample_size_dict[name]
    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)
print (corpus_df.groupby('Instruction_id').size(), corpus_df.shape)
```

```
Instruction_id
auto-repair-appt-1    230
coffee-ordering       230
movie-finder           54
movie-tickets-1       250
movie-tickets-2       250
movie-tickets-3       195
pizza-ordering        230
restaurant-table      230
restaurant-table-3    101
uber-lyft             230
dtype: int64 (2000, 3)
```

In [15]:
```
# group by service_type, count distinct Instruction_id
cat_id_df = corpus_df.groupby('Instruction_id') \
           .agg({'category': pd.Series.count}) \
           .rename(columns={'category': 'num_posts'}) \
           .sort_values('num_posts', ascending=False)

# show top 5 records
cat_id_df.head(5)
```

Out[15]:

Instruction_id	num_posts
movie-tickets-1	250
movie-tickets-2	250
auto-repair-appt-1	230
coffee-ordering	230
pizza-ordering	230

In [16]:
```
# # Function to Get balanced Sample - Get a bit more than needed then down sample
# def sampling_k_elements(group, k=sample_per_cat + 20):
#     if len(group) < k:
#         return group
#     return group.sample(k, random_state=10)

# #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())
```

Generate Corpus List

In [17]:
```
doc_lst = []
for i, row in corpus_df.iterrows():
    doc_lst.append(row.selfdialog_norm.tolist())

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

```
2000
```

Out[17]:
```
['hi im issue car help sure whats problem light came saying headlight ok want get fixed right away today would ideal already
know want take yes intelligent auto solutions ok let pull website online scheduler see today ok im looks like two appointmen
ts open today could minutes im least minutes away ok time would pm tonight tell able fix spot call confirm makemodel car kia
soul ok said parts done appointment thats great news please book yes booked online thanks give info yes text youll phone tha
nk big help',
 'hi schedule appointment car okay auto repair shop would like check check intelligent auto solutions car bringing lexus im
driving put name cell phone number yes put jeff green cell phone number seems problem car makes sound step brakes anything e
lse would like check like oil change maintenance yes think im due oil change well got let check online see available check b
ring mins able make appointment bring car time pm great thanks initial cost brake checkup oil change okay accept credit card
yes great thanks bye youre welcome bye',
 'assistant favor yes course whats going car making weird rattly noises think checked find good mechanic certainly im checki
ng google right moment ok appears auto shop near work star rating want give call yes please ok ill put hold moment see say g
reat thanks ok im back said bring tomorrow ok long going keep depends whats going said could problem muffler wont know look
gave number theyll give call alright make sure get uber tomorrow morning yes time well probably need leave house ok ill hous
e get car ill make sure uber arrives well thank much youre welcome need anything else ok see tomorrow',
 'gail need help schedule appointment intelligent auto solutions car whats wrong car need schedule appointment look radiator
see drops fluid time park ground ok year model car bmw series sure name use use name scolar timer address miklan road forest
hills new mexico bring car tomorrow see get earlier situation annoying time bring work pm take abut minutes ok let check wou
ld prefer bring tomorrow morning let check time slots way please reserve car use mean time case car kept overnight well chec
ked time bring pm today ok let confirm everything bring car today pm check leaking radiator get car ise case car stays overn
ight thats correct repair shop need initial inspection thats ok go right ahead book appointment sure everything booked reque
sted thanks help talk later']
```

Build Vocabulary

In [18]:
```
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(doc_lst)
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 doc_lst]

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.
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:528: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:529: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:530: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\tensorflow\python\framework\dtypes.py:535: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])


Vocabulary Size: 8408
Vocabulary Sample: [('like', 1), ('would', 2), ('ok', 3), ('okay', 4), ('pm', 5), ('yes', 6), ('want', 7), ('tickets', 8),
('time', 9), ('see', 10)]
```

Build (context_words, target_word) pair generator

In [19]:
```
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            start = index - window_size
            end = index + window_size + 1

            context_words.append([words[i]
                                 for i in range(start, end)
                                 if 0 <= i < sentence_length
                                 and i != index])
            label_word.append(word)

            x = sequence.pad_sequences(context_words, maxlen=context_length)
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)
```

In [20]:
```
i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argwhere(y[0])[0][0]])

        if i == 10:
            break
        i += 1
```

```
Context (X): ['want', 'make', 'auto', 'repair'] -> Target (Y): appointment
Context (X): ['make', 'appointment', 'repair', 'shop'] -> Target (Y): auto
Context (X): ['appointment', 'auto', 'shop', 'called'] -> Target (Y): repair
Context (X): ['auto', 'repair', 'called', 'intelligent'] -> Target (Y): shop
Context (X): ['repair', 'shop', 'intelligent', 'auto'] -> Target (Y): called
Context (X): ['shop', 'called', 'auto', 'solutions'] -> Target (Y): intelligent
Context (X): ['called', 'intelligent', 'solutions', 'okay'] -> Target (Y): auto
Context (X): ['intelligent', 'auto', 'okay', 'search'] -> Target (Y): solutions
Context (X): ['auto', 'solutions', 'search', 'information'] -> Target (Y): okay
Context (X): ['solutions', 'okay', 'information', 'car'] -> Target (Y): search
Context (X): ['okay', 'search', 'car', 'audi'] -> Target (Y): information
```

Set up Dictionaries to Cross-Refrence 'Instruction_id' and its Factorized value 'category'

In [21]:
```
category_id_df = corpus_df[['Instruction_id', 'category']].drop_duplicates().sort_values('category')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category', 'Instruction_id']].values)
```

Split Data into Train and Test Sets

In [22]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(doc_lst, corpus_df['Instruction_id'], test_size=0.25, random_state = 0)
```

Bag of Words Feature Extraction

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer

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

Out[23]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 1, 2, ..., 0, 0, 0],
       [0, 3, 2, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 1, 1, 0],
       [0, 0, 0, ..., 0, 0, 1]], dtype=int64)
```

In [24]:
```python
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

Out[24]:

	PAD	like	would	ok	okay	pm	yes	want	tickets	time	see	thank	order	movie	please	...	bored	olivia	westfield	plazas	coworkers	impre
0	0	0	0	1	3	0	0	1	0	1	0	1	0	0	0	...	0	0	0	0	0	
1	0	1	2	5	0	1	3	2	0	1	1	1	0	0	1	...	0	0	0	0	0	
2	0	3	2	0	2	1	3	0	0	1	1	0	0	0	0	...	0	0	0	0	0	
3	0	0	0	6	0	0	3	1	0	1	2	1	0	0	1	...	0	0	0	0	0	
4	0	0	1	4	0	3	0	0	0	5	2	0	0	0	1	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1995	0	1	0	0	1	1	1	2	0	0	2	0	1	1	0	...	0	0	1	1	1	
1996	0	3	2	0	11	3	1	1	0	2	0	1	0	0	0	...	0	0	0	0	0	
1997	0	0	0	4	0	0	2	0	0	0	0	1	1	0	5	...	0	0	0	0	0	
1998	0	0	1	7	0	0	2	3	0	0	0	1	2	0	1	...	0	0	0	0	0	
1999	0	0	0	6	0	0	3	0	0	1	0	2	1	0	5	...	0	0	0	0	0	

2000 rows × 8408 columns

In [25]:
```python
# Get BOW features
X_train_bow = cv.fit_transform(X_train).toarray()
X_test_bow = cv.transform(X_test).toarray()
print (X_train_bow.shape)
print (X_test_bow.shape)
print (y_test.shape)
```

```
(1500, 8408)
(500, 8408)
(500,)
```

Define Model Builder Function

In [26]:
```python
#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 [27]:
```
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)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [28]:
```

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_metrics.accuracy])
    rows_benchmarks_all.append([descr, feat_type, b_metrics.precision, b_metrics.recall, b_metrics.f1_score, b_metrics.accurac
    df_benchmarks = pd.DataFrame(rows_benchmarks, columns=["Features_Benchedmarked", "Feat_Type", "Precision", "Recall", "f1_s
    df_benchmarks_all = pd.DataFrame(rows_benchmarks_all, columns=["Features_Benchedmarked", "Feat_Type", "Precision", "Recall
```

In [29]:
```
# Save benchmark output
Save_Benchmark("BOW Naive Bayes Baseline", "BOW", rm_nb_bow, True, True)
df_benchmarks
```

Out[29]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW Naive Bayes Baseline	BOW	0.8394773	0.8600000	0.8438828	0.8600000

In [30]:
```
from sklearn.metrics import confusion_matrix

rm_nb_bow.cm
```

Out[30]:
```
array([[64,  1,  0,  0,  0,  0,  0,  0,  0,  1],
       [ 0, 57,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0, 11,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  1, 52, 15,  1,  0,  0,  0,  0],
       [ 0,  0,  1,  9, 43, 16,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  0, 32,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 55,  1,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 61, 20,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  1,  0, 55]], dtype=int64)
```

In [31]:
```
from sklearn import metrics

print("Label" + rm_nb_bow.report)
```

```
Label                    precision    recall  f1-score   support

auto-repair-appt-1       0.97      1.00      0.98        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       1.00      0.73      0.85        15
   movie-tickets-1       0.75      0.85      0.80        61
   movie-tickets-2       0.62      0.74      0.68        58
   movie-tickets-3       0.94      0.65      0.77        49
    pizza-ordering       0.98      0.98      0.98        56
  restaurant-table       0.75      0.97      0.85        63
restaurant-table-3       0.00      0.00      0.00        20
         uber-lyft       0.98      0.98      0.98        56

          accuracy                           0.86       500
         macro avg       0.80      0.79      0.79       500
      weighted avg       0.84      0.86      0.84       500
```

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

Define Feature Selection Functions

In [32]:
```
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_scaled, x_test_scaled = Get_Scaled_Features(features_train, labels_train, features_test, labels_test, scaler)
#     X_train_chi, X_test_chi = Select_Best_Features_Chi(num_feats, x_train_scaled, labels_train, x_test_scaled, labels_test)
#     rm_chi = Build_Model(model, X_train_chi, labels_train, X_test_chi, labels_test)
#     return rm_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, labels_test)
    x_train_scaled, x_test_scaled = Get_Scaled_Features(X_train_chi, labels_train, X_test_chi, labels_test, scaler)
    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, scaler):
    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_chi, precisior

    return rm_chi
```

Iterate through number of features and get benchmark results

```
In [33]: def Get_ABC_Range(x, a, c):
             a = a
             tot = x.shape[1]
             b = 100 * (tot//100)
             c = c
             return a, b, c
```

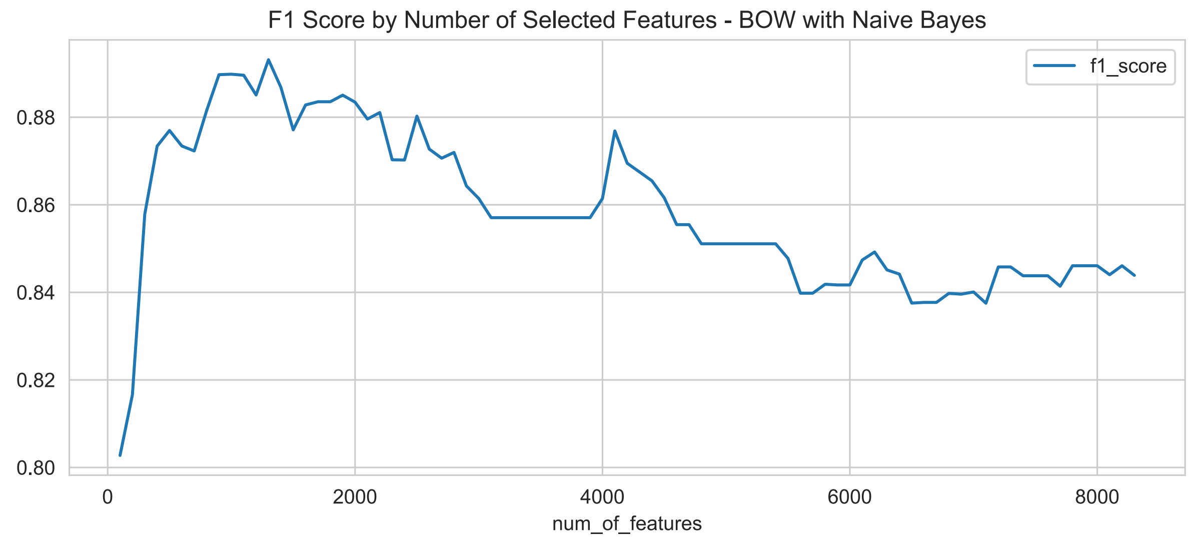
```
In [34]: import sys
```

```
In [35]: rows = []
         a, b, c = Get_ABC_Range(X_train_bow, 100, 100)
         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_max)
             rm_chi_i = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_max)
             rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])
             sys.stdout.write('\r'+str(i) + "/" + str(b))
             sys.stdout.flush()

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

```
6400/8400

C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))

6500/8400

C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
```

Plot f1-score by number of selected features

```
In [36]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - BOW with Naive Bayes", figsize

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x2664ea43208>
```



In [37]:
```
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)
```

```
1250 1350 1
```

Out[37]:

	num_of_features	f1_score	accuracy
12	1300	0.8931327	0.8940000
9	1000	0.8898212	0.8900000
8	900	0.8897150	0.8900000
10	1100	0.8895844	0.8900000
13	1400	0.8868523	0.8900000

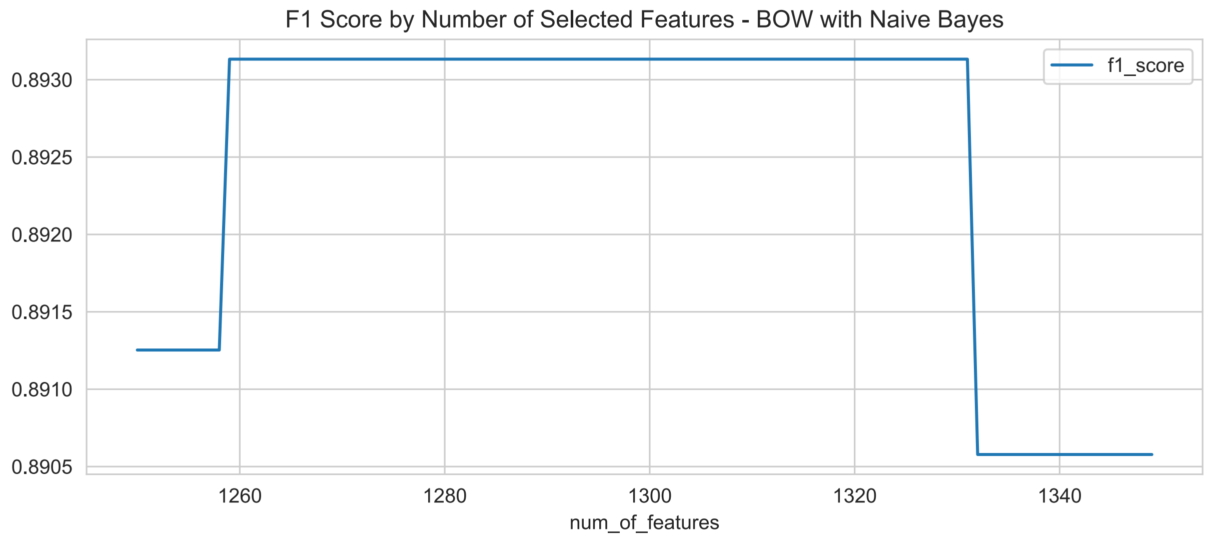
Get a more fine-grained look at the optimal number of features region

In [38]:
```
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_max)
    rm_chi_i = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_max)
    rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])
    sys.stdout.write('\r'+str(i) + "/" + str(b))
    sys.stdout.flush()

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

```
1349/1350
```

In [39]:
```
acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - BOW with Naive Bayes", figsize
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2664eb4e988>
```



In [40]:
```
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)
```

```
1300
```

Out[40]:

	num_of_features	f1_score	accuracy
50	1300	0.8931327	0.8940000
37	1287	0.8931327	0.8940000
53	1303	0.8931327	0.8940000
52	1302	0.8931327	0.8940000
51	1301	0.8931327	0.8940000

Benchmark BOW With Optimal Features Selected using Naive Bayes Model

In [41]:
```
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_test, scaler_mi
```

In [42]:
```
print(rm_chi_opt_bow.cm)
```

```
[[64  1  0  0  0  0  0  0  0  1]
 [ 0 57  0  0  0  0  1  0  0  0]
 [ 0  0 14  0  0  0  0  0  0  0]
 [ 0  0  1 50 11  1  0  0  0  0]
 [ 0  0  0 11 45  8  0  0  0  0]
 [ 0  0  0  0  2 40  0  0  0  0]
 [ 0  0  0  0  0  0 55  1  0  0]
 [ 0  0  0  0  0  0  0 57 10  0]
 [ 0  0  0  0  0  0  0  4 10  0]
 [ 0  0  0  0  0  0  0  1  0 55]]
```

In [43]:
```
print("Label" + rm_chi_opt_bow.report)
```

```
Label                    precision    recall  f1-score   support

auto-repair-appt-1       0.97      1.00      0.98        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       1.00      0.93      0.97        15
   movie-tickets-1       0.79      0.82      0.81        61
   movie-tickets-2       0.70      0.78      0.74        58
   movie-tickets-3       0.95      0.82      0.88        49
    pizza-ordering       0.98      0.98      0.98        56
  restaurant-table       0.85      0.90      0.88        63
restaurant-table-3       0.71      0.50      0.59        20
         uber-lyft       0.98      0.98      0.98        56

          accuracy                           0.89       500
         macro avg       0.89      0.87      0.88       500
      weighted avg       0.90      0.89      0.89       500
```

In [44]:
```
# Save benchmark output
Save_Benchmark("BOW Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "BOW", rm_chi_opt_bow, False, False)
df_benchmarks
```

Out[44]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW Naive Bayes Baseline	BOW	0.8394773	0.8600000	0.8438828	0.8600000
1	BOW Naive Bayes Optimal Features Selected: 1300	BOW	0.8956079	0.8940000	0.8931327	0.8940000

In [45]:
```
df_bow_train = pd.DataFrame(rm_chi_opt_bow.x_train_sel)
df_bow_train.to_csv('./data/bow_selected_train.csv', index=False)

df_bow_test = pd.DataFrame(rm_chi_opt_bow.x_test_sel)
df_bow_test.to_csv('./data/bow_selected_train.csv', index=False)
```

Bag of N-Grams Feature Extraction

```
In [46]: from sklearn.feature_extraction.text import CountVectorizer

         bv = CountVectorizer(ngram_range=(2,2))
         bv_matrix = bv.fit_transform(X_train)
         bv_matrix = bv_matrix.toarray()
         bv_matrix
```

```
Out[46]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
In [47]: # get all unique words in the corpus
         vocab = bv.get_feature_names()
         # show document feature vectors
         pd.DataFrame(bv_matrix, columns=vocab)
```

Out[47]:

	aamir khan	aaron says	abby family	abc thanks	abcgmailcom ok	abigail lives	abigails whoops	abigails yes	ability scan	able access	able accomodate	able attend	able book	able bring	able complete	...	zero televisions
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1496	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1497	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1498	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1499	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

1500 rows × 63889 columns

```
In [48]: # Get Bag of N-Gram features
         X_train_bong = bv.fit_transform(X_train).toarray()
         X_test_bong = bv.transform(X_test).toarray()
         print (X_train_bong.shape)
         print (X_test_bong.shape)
         print (y_test.shape)
```

```
(1500, 63889)
(500, 63889)
(500,)
```

Bag of N-Grams Feature Benchmarking with Naive Bayes Classifier

```
In [49]: from sklearn.naive_bayes import MultinomialNB

         model_nb_bong = MultinomialNB()
         results_nb_bong = Build_Model(model_nb_bong, X_train_bong, y_train, X_test_bong, y_test)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [50]:
```
# Save benchmark output
Save_Benchmark("Bag of N-Gram Naive Bayes baseline", "BONG", results_nb_bong, True, False)
df_benchmarks
```

Out[50]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Bag of N-Gram Naive Bayes baseline	BONG	0.8253737	0.8340000	0.8185937	0.8340000

In [51]:
```
from sklearn.metrics import confusion_matrix

results_nb_bong.cm
```

Out[51]:
```
array([[64,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 57,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  8,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  1, 46, 10,  1,  0,  0,  0,  1],
       [ 0,  0,  2, 15, 47, 22,  0,  1,  1,  0],
       [ 0,  0,  2,  0,  1, 26,  0,  0,  1,  0],
       [ 0,  0,  1,  0,  0,  0, 55,  1,  0,  0],
       [ 0,  1,  1,  0,  0,  0,  0, 60, 18,  1],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  1,  0, 54]], dtype=int64)
```

In [52]:
```
from sklearn import metrics

print(results_nb_bong.report)
```

```
                     precision    recall  f1-score   support

auto-repair-appt-1       1.00      1.00      1.00        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       1.00      0.53      0.70        15
   movie-tickets-1       0.78      0.75      0.77        61
   movie-tickets-2       0.53      0.81      0.64        58
   movie-tickets-3       0.87      0.53      0.66        49
    pizza-ordering       0.96      0.98      0.97        56
  restaurant-table       0.74      0.95      0.83        63
restaurant-table-3       0.00      0.00      0.00        20
         uber-lyft       0.98      0.96      0.97        56

          accuracy                           0.83       500
         macro avg       0.79      0.75      0.75       500
      weighted avg       0.83      0.83      0.82       500
```

Feature Selection: Bag of N-Gram Features with Naive Bayes Model Using Chi-Squared Selector

Iterate through number of features and get benchmark results

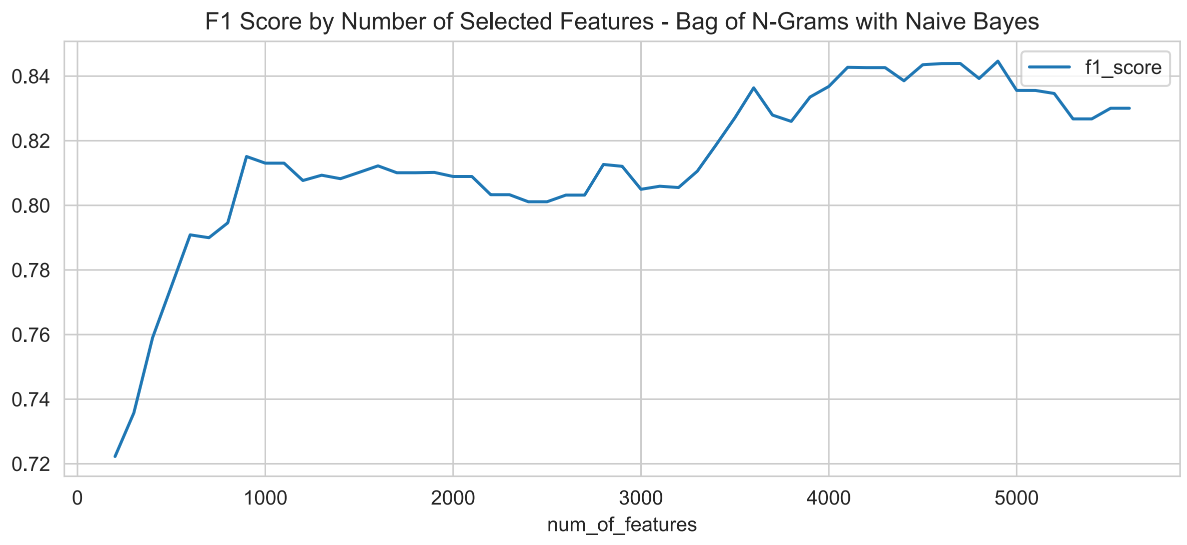
In [53]:
```
rows = []
a = 200
b = 5700
c = 100
for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(model_nb_bong, i, X_train_bong, y_train, X_test_bong, y_test, scaler_min_max)
    rows.append([i, results_i.f1_score, results_i.accuracy])
    sys.stdout.write('\r'+str(i) + "/" + str(b))
    sys.stdout.flush()

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

```
5600/5700
```

Plot f1-score by number of selected features

```
In [54]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - Bag of N-Grams with Naive Baye
```

```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x2665393b848>
```



```
In [55]: 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)
```

```
4850 4950 1
```

Out[55]:

	num_of_features	f1_score	accuracy
47	4900	0.8446399	0.8520000
45	4700	0.8439093	0.8500000
44	4600	0.8438834	0.8500000
43	4500	0.8435312	0.8500000
39	4100	0.8427264	0.8500000

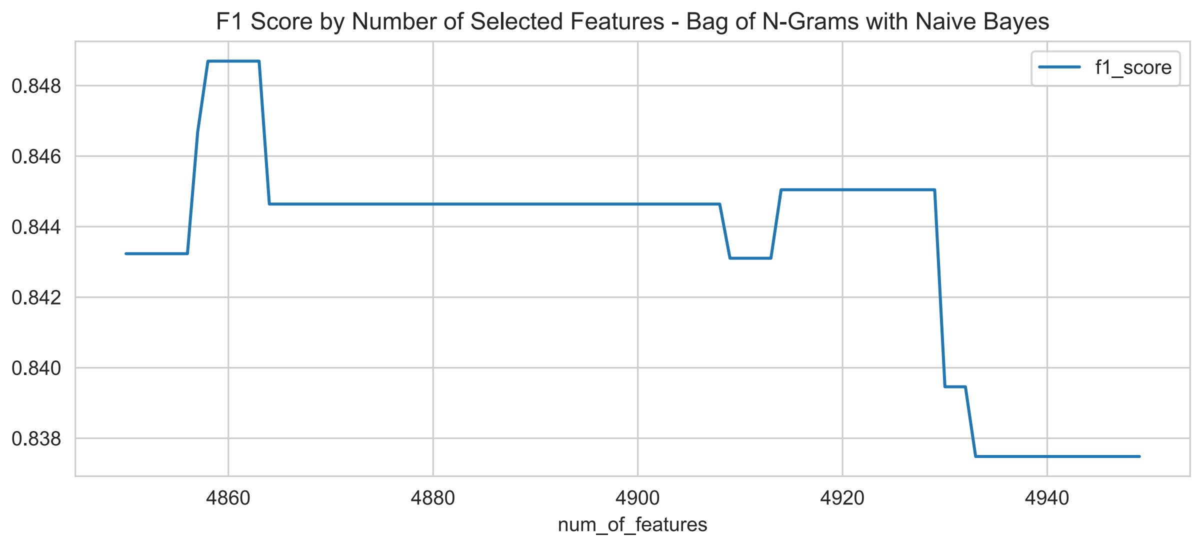
Get a more fine-grained look at the optimal number of features region

```
In [56]: rows = []
         for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
             results_i = SelectBestModelFeatures_Chi(model_nb_bong, i, X_train_bong, y_train, X_test_bong, y_test, scaler_min_max)
             rows.append([i, results_i.f1_score, results_i.accuracy])
             sys.stdout.write('\r'+str(i) + "/" + str(b))
             sys.stdout.flush()

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

```
4949/4950
```

```
In [57]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - Bag of N-Grams with Naive Baye
```

```
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x2662eb93d48>
```



```
In [58]: Opt_no_of_feat = int(acc_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
         Opt_no_of_feat
         acc_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[58]:

	num_of_features	f1_score	accuracy
8	4858	0.8486926	0.8560000
9	4859	0.8486926	0.8560000
10	4860	0.8486926	0.8560000
11	4861	0.8486926	0.8560000
12	4862	0.8486926	0.8560000

Benchmark Bag of N-Grams With Optimal Features Selected using Naive Bayes Model

```
In [59]: model_nb_bong_opt = MultinomialNB()
         results_bong_opt = SelectBestModelFeatures_Chi(model_nb_bong_opt, Opt_no_of_feat, X_train_bong, y_train, X_test_bong, y_test,
```

```
In [60]: print(results_bong_opt.report)
```

```
                     precision    recall  f1-score   support

auto-repair-appt-1       0.97      0.98      0.98        64
   coffee-ordering       0.98      0.97      0.97        58
      movie-finder       1.00      0.80      0.89        15
   movie-tickets-1       0.68      0.82      0.74        61
   movie-tickets-2       0.66      0.71      0.68        58
   movie-tickets-3       0.89      0.65      0.75        49
    pizza-ordering       0.96      0.98      0.97        56
  restaurant-table       0.79      0.95      0.86        63
restaurant-table-3       0.80      0.20      0.32        20
         uber-lyft       0.98      0.98      0.98        56

          accuracy                           0.86       500
         macro avg       0.87      0.80      0.82       500
      weighted avg       0.86      0.86      0.85       500
```

```
In [61]: # Save benchmark output
         Save_Benchmark("Bag of N-Gram Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "BONG", results_bong_opt, False,
         df_benchmarks
```

Out[61]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Bag of N-Gram Naive Bayes baseline	BONG	0.8253737	0.8340000	0.8185937	0.8340000
1	Bag of N-Gram Naive Bayes Optimal Features Selected: 4858	BONG	0.8638235	0.8560000	0.8486926	0.8560000

TF-IDF Feature Extraction

In [62]:
```python
#from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
X_test_counts = count_vect.transform(X_test)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(1500, 7190)
(500, 7190)
```

In [63]:
```python
vocab_tfidf = count_vect.get_feature_names()
pd.DataFrame(X_train_tfidf.toarray(), columns=vocab_tfidf)
```

Out[63]:

	aamir	aaron	abby	abc	abcgmailcom	abigail	abigails	ability	able	abnormal	aboutpm	aboutthe	abraham
0	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
2	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
3	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
4	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1496	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1497	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1498	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
1499	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

1500 rows × 7190 columns

TF-IDF Baseline Benchmarking with Naive Bayes Classifier: Multinomial variant

In [64]:
```python
clf = MultinomialNB()
results_nb_tfidf = Build_Model(clf, X_train_tfidf, y_train, X_test_tfidf, y_test)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Make Some Predictions

In [65]:
```python
X_new_data_counts = count_vect.transform(["appointment online car checking bmw okay hold minute problem okay entered thank nee
X_new_data_tfidf = tfidf_transformer.fit_transform(X_new_data_counts)
print(X_new_data_tfidf.shape)
y_pred_new = clf.predict(X_new_data_tfidf)
y_pred_new
```

```
(1, 7190)
```

Out[65]:
```
array(['auto-repair-appt-1'], dtype='<U18')
```

Metrics for TF-IDF with Naive Bayes Classifier: Multinomial variant

```
In [66]: from sklearn.metrics import confusion_matrix

         results_nb_tfidf.cm
```

```
Out[66]: array([[64,  1,  0,  0,  0,  0,  0,  0,  0,  1],
                [ 0, 57,  0,  0,  0,  0,  1,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  4, 48, 13,  1,  0,  0,  0,  0],
                [ 0,  0, 11, 13, 45, 38,  0,  0,  1,  0],
                [ 0,  0,  0,  0,  0, 10,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0, 55,  1,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 61, 19,  0],
                [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0,  1,  0, 55]], dtype=int64)
```

```
In [67]: from sklearn import metrics

         print("Label" + results_nb_tfidf.report)
```

```
Label                        precision    recall  f1-score   support

auto-repair-appt-1       0.97      1.00      0.98        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       0.00      0.00      0.00        15
   movie-tickets-1       0.73      0.79      0.76        61
   movie-tickets-2       0.42      0.78      0.54        58
   movie-tickets-3       1.00      0.20      0.34        49
    pizza-ordering       0.98      0.98      0.98        56
  restaurant-table       0.76      0.97      0.85        63
restaurant-table-3       0.00      0.00      0.00        20
         uber-lyft       0.98      0.98      0.98        56

          accuracy                           0.79       500
         macro avg       0.68      0.67      0.64       500
      weighted avg       0.79      0.79      0.76       500
```

```
In [68]: # Save benchmark output
         Save_Benchmark("TF-IDF Naive Bayes Baseline", "TF-IDF", results_nb_tfidf, True, False)
         df_benchmarks
```

Out[68]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	TF-IDF Naive Bayes Baseline	TF-IDF	0.7892568	0.7900000	0.7558597	0.7900000

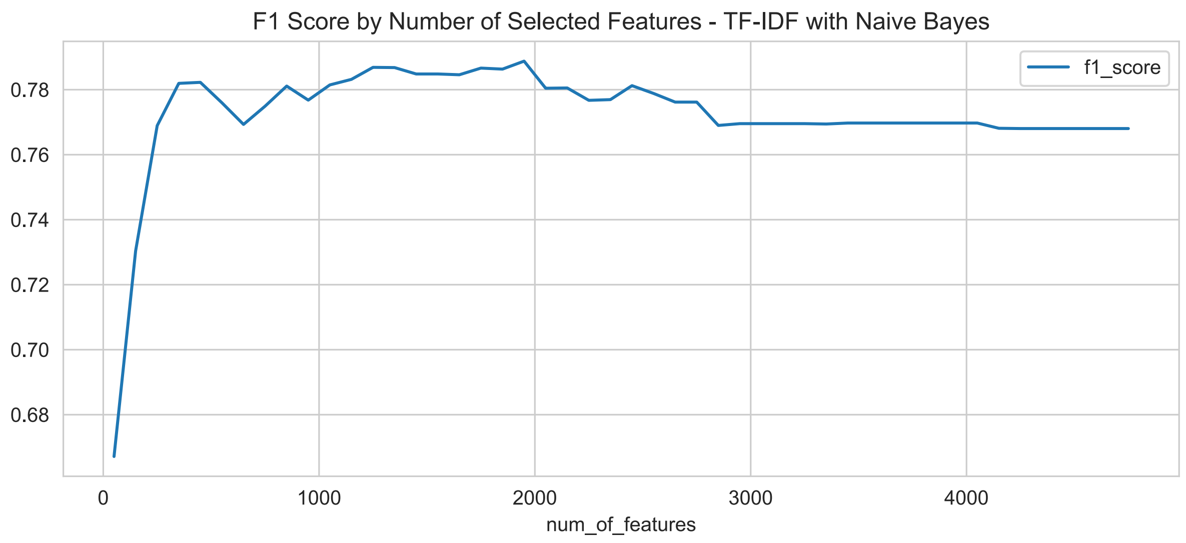
Feature Selection - TF-IDF with Naive Bayes

```
In [69]: rows = []
         scaler_max_abs = MaxAbsScaler()
         for i in range(50, 4850, 100): # range(a, b, c) will count from a to b by intervals of c.
             results_i = SelectBestModelFeatures_Chi(clf, i, X_train_tfidf, y_train, X_test_tfidf, y_test, scaler_max_abs)
             rows.append([i, results_i.f1_score, results_i.accuracy])

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

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
```

In [70]:
```python
sel_nb_tfidf_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - TF-IDF with Naive Bay
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x266533d6588>
```



In [71]:
```python
Opt_no_of_feat = int(sel_nb_tfidf_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)
sel_nb_tfidf_df.sort_values(by='f1_score', ascending=False).head(5)
```

```
1900 2000 1
```

Out[71]:

	num_of_features	f1_score	accuracy
19	1950	0.7887809	0.8180000
12	1250	0.7868693	0.8140000
13	1350	0.7867987	0.8140000
17	1750	0.7866386	0.8160000
18	1850	0.7863282	0.8160000

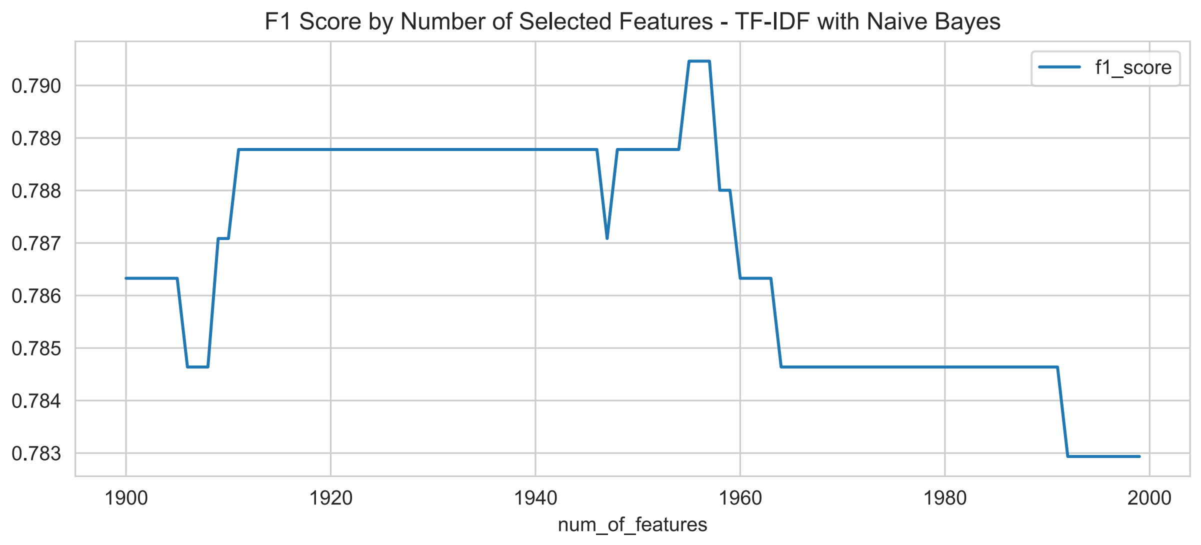
Take closer look at region around optimal features

In [72]:
```python
rows = []
for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(clf, i, X_train_tfidf, y_train, X_test_tfidf, y_test, scaler_max_abs)
    rows.append([i, results_i.f1_score, results_i.accuracy])

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

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precis
ion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
```

In [73]:
```
sel_nb_tfidf_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - TF-IDF with Naive Bay
```

Out[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x266534022c8>
```



In [74]:
```
Opt_no_of_feat = int(sel_nb_tfidf_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
Opt_no_of_feat
sel_nb_tfidf_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[74]:

	num_of_features	f1_score	accuracy
55	1955	0.7904632	0.8200000
56	1956	0.7904632	0.8200000
57	1957	0.7904632	0.8200000
40	1940	0.7887809	0.8180000
29	1929	0.7887809	0.8180000

Benchmark TF-IDF Features with Naive Bayes on Optimal Features

In [75]:
```
results_tf_nb_opt = SelectBestModelFeatures_Chi(clf, Opt_no_of_feat, X_train_tfidf, y_train, X_test_tfidf, y_test, scaler_max_
# Save benchmark output
Save_Benchmark("TF-IDF Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "TF-IDF", results_tf_nb_opt, False, Fal
df_benchmarks
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[75]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	TF-IDF Naive Bayes Baseline	TF-IDF	0.7892568	0.7900000	0.7558597	0.7900000
1	TF-IDF Naive Bayes Optimal Features Selected: 1955	TF-IDF	0.7882401	0.8200000	0.7904632	0.8200000

Metrics For Each Class

In [76]:
```
from sklearn import metrics
print("Label" + results_tf_nb_opt.report)
```

```
Label                        precision    recall  f1-score   support

auto-repair-appt-1       0.97      1.00      0.98        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       0.00      0.00      0.00        15
   movie-tickets-1       0.78      0.80      0.79        61
   movie-tickets-2       0.52      0.84      0.64        58
   movie-tickets-3       0.83      0.41      0.55        49
    pizza-ordering       0.98      0.98      0.98        56
  restaurant-table       0.74      0.97      0.84        63
restaurant-table-3       0.00      0.00      0.00        20
         uber-lyft       0.98      0.98      0.98        56

          accuracy                           0.82       500
         macro avg       0.68      0.70      0.68       500
      weighted avg       0.79      0.82      0.79       500
```

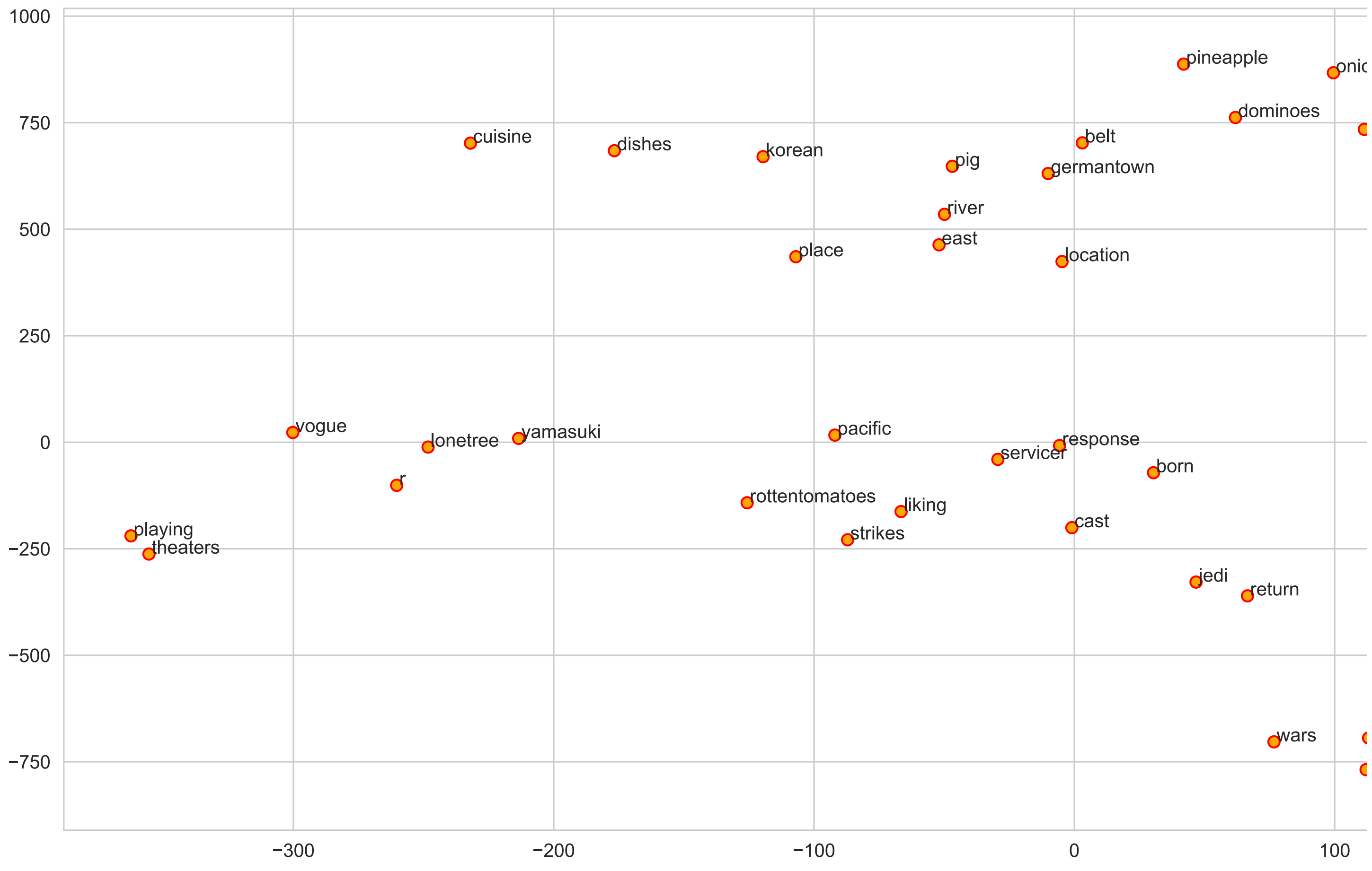
Word2Vec Feature Extraction

In [77]:
```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in X_train]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                  for search_term in ['pizza', 'jedi', 'star', 'east', 'korean','playing']}
similar_words
```

Out[77]:
```
{'pizza': ['onions', 'dominoes', 'supreme', 'mushroom', 'pineapple'],
 'jedi': ['return', 'wars', 'strikes', 'liking', 'cast'],
 'star': ['rating', 'wars', 'born', 'response', 'servicer'],
 'east': ['river', 'yamasuki', 'location', 'pacific', 'place'],
 'korean': ['pig', 'germantown', 'dishes', 'belt', 'cuisine'],
 'playing': ['theaters', 'r', 'lonetree', 'rottentomatoes', 'vogue']}
```

Visualizing word embeddings

```
In [78]: from sklearn.manifold import TSNE

         words = sum([[k] + v for k, v in similar_words.items()], [])
         wvs = w2v_model.wv[words]

         tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
         np.set_printoptions(suppress=True)
         T = tsne.fit_transform(wvs)
         labels = words

         plt.figure(figsize=(14, 8))
         plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
         for label, x, y in zip(labels, T[:, 0], T[:, 1]):
             plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```



Applying the word2vec model on our Train dataset

```
In [79]: wpt = nltk.WordPunctTokenizer()
         tokenized_corpus = [wpt.tokenize(document) for document in X_train]
         tokenized_corpus_test = [wpt.tokenize(document) for document in X_test]

         # Set values for various parameters
         feature_size = 100    # Word vector dimensionality
         window_context = 10          # Context window size
         min_word_count = 1   # Minimum word count
         sample = 1e-3   # Downsample setting for frequent words

         w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                                   window=window_context, min_count = min_word_count,
                                   sample=sample, iter=100)
```

```
In [80]: def Get_W2V_Model(feat_size):
             w2v_mod = word2vec.Word2Vec(tokenized_corpus, size=feat_size,
                                   window=window_context, min_count = min_word_count,
                                   sample=sample, iter=100)
             return w2v_mod
```

Do a Word Test

In [81]:
```python
w2v_model.wv['jedi']
```

Out[81]:
```
array([ 0.00444527, -0.20022497, -0.22189076,  0.33230653,  1.679024  ,
        1.1267267 , -0.01243084,  0.7186285 ,  0.7505811 , -1.4433627 ,
       -0.26907203, -0.3831739 , -0.97433585,  0.43306458,  1.3241014 ,
       -0.20783381,  1.658895  ,  0.06650375,  0.10575018, -0.21216157,
        0.9611219 ,  0.5581417 ,  1.3518509 ,  1.306351  ,  0.7245609 ,
        0.6889246 ,  0.01336428,  0.8197848 , -0.19549444, -0.30738696,
        0.6719403 ,  0.05018056, -0.7909318 , -0.06128511, -0.9679422 ,
        1.323732  , -0.20721155,  0.9038662 ,  0.06726235, -0.5826981 ,
       -0.82464373,  1.0225344 , -0.37858832,  0.3782428 ,  0.37311265,
        0.65518045, -1.5200177 ,  0.12924269,  0.568354  ,  0.64329857,
        0.7383073 ,  0.07866799, -0.7840065 , -0.21280016,  0.84652305,
       -0.29744875, -0.9724518 , -1.1632009 ,  1.2374166 , -1.007973  ,
       -0.14649172, -0.42728198, -0.6107526 , -0.4640104 , -1.6708323 ,
        0.5532841 , -0.30765277, -0.5363567 , -0.0059393 , -0.7886058 ,
       -1.0054519 ,  0.7428801 ,  1.1863735 ,  0.20501624, -0.9600224 ,
       -0.2863513 ,  1.6481388 , -0.6852151 , -0.2964974 , -0.00041215,
       -0.598797  , -0.2397256 , -1.1330425 ,  0.1618307 , -0.00649552,
       -0.05574537, -0.11044064,  0.9264163 ,  0.47784716, -0.4149952 ,
        0.13324085, -0.66842777, -0.11129779,  0.26184374, -0.28365067,
        1.0348022 ,  0.18655612, -0.68037283,  0.02332349, -0.01477248],
      dtype=float32)
```

Build framework for getting document level embeddings

In [82]:
```python
def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector


def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
                    for tokenized_sentence in corpus]
    return np.array(features)
```

In [83]:
```python
w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w2v_model,
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated `__getite
m__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

Out[83]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	0.1873218	-0.4735788	0.9725689	-1.5548968	-0.0483634	0.5444516	-0.2362861	-0.3395859	0.9621134	-0.7932116	-0.2347472	-0.9821420	1.56239
1	0.0178411	-0.0560393	0.9839213	-0.4040909	0.0176693	0.3752609	-0.5976276	0.0030219	0.0433316	-0.4721514	-0.4829037	-0.3136707	1.12700
2	-0.1484596	-0.1243862	-0.6997141	0.6127047	-0.7421537	0.5409077	0.1489333	0.8100881	1.0104762	-0.6006313	-0.2722385	-0.4159432	1.08594
3	0.1373973	0.0020785	0.7520878	-0.3897839	-0.1119516	0.1614587	0.2419387	-0.2300063	0.4225348	1.2530223	-0.3051739	0.1348216	0.82955
4	0.8935827	-0.6581176	1.0229267	0.1918840	0.0763810	0.0853404	0.8857397	-0.4736567	0.1782164	0.6857576	-0.0704821	0.4307056	1.45702
...	...	...	...	...	...	...	...	...	...	...	...	...	
1495	0.6359248	-0.8073446	1.2037342	-0.1941031	0.1144299	0.5235185	0.3673428	-0.1585717	0.4354501	0.3227923	0.2609709	-0.3090171	0.05004
1496	0.9471351	-0.4739289	1.4136777	0.3378935	0.2156104	-0.4278203	0.4220656	0.3073745	-1.0997889	-0.7592334	0.2078579	-0.3439014	-1.35721
1497	0.9265414	-0.4210868	-0.4615308	0.6569034	-0.2054187	0.8343573	0.6735410	-0.0878270	1.2543531	-0.3141014	-0.1724857	0.1617612	2.09583
1498	0.0983251	-0.6904276	1.0041442	-0.5233796	0.3611044	0.3317488	-0.1283708	0.0498702	0.2342417	0.2593339	0.1916878	0.3933110	0.99180
1499	0.2193377	-0.1996251	0.8627414	-0.4336795	-0.0377298	0.0567343	0.3240563	0.0392423	0.5753537	0.1041630	0.2219171	0.2447023	0.76004

1500 rows × 100 columns

In [84]:
```
w2v_test_array = averaged_word_vectorizer(corpus=tokenized_corpus_test, model=w2v_model,
                                              num_features=feature_size)

print(w2v_test_array.shape)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated `__getite
m__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':

(500, 100)
```

Word2vec Feature Benchmarking with Naive Bayes Classifier

In [85]:
```
from sklearn.naive_bayes import GaussianNB

scaler_min_max = MinMaxScaler()
#model_w2v_nb = MultinomialNB()
model_w2v_nb = GaussianNB()
results_nb_w2v = SelectBestModelFeatures_Chi(model_w2v_nb, 100, w2v_feature_array, y_train, w2v_test_array, y_test, scaler_mir
# Save benchmark output
Save_Benchmark("Word2Vec Naive Bayes Baseline", "Word2Vec", results_nb_w2v, True, False)
df_benchmarks
```

Out[85]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Word2Vec Naive Bayes Baseline	Word2Vec	0.8557780	0.8380000	0.8419654	0.8380000

In [86]:
```
results_nb_w2v.cm
```

Out[86]:
```
array([[63,  1,  0,  0,  0,  0,  0,  0,  0,  1],
       [ 0, 56,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0, 14,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 44, 16,  3,  0,  0,  0,  0],
       [ 0,  0,  1, 17, 37,  6,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  5, 40,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 55,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 40,  5,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 21, 15,  0],
       [ 1,  1,  0,  0,  0,  0,  0,  2,  0, 55]], dtype=int64)
```

In [87]:
```
print("Label" + results_nb_w2v.report)
```

```
Label                    precision    recall  f1-score   support

auto-repair-appt-1       0.97      0.98      0.98        64
   coffee-ordering       0.98      0.97      0.97        58
      movie-finder       1.00      0.93      0.97        15
   movie-tickets-1       0.70      0.72      0.71        61
   movie-tickets-2       0.61      0.64      0.62        58
   movie-tickets-3       0.89      0.82      0.85        49
    pizza-ordering       1.00      0.98      0.99        56
  restaurant-table       0.89      0.63      0.74        63
restaurant-table-3       0.42      0.75      0.54        20
         uber-lyft       0.93      0.98      0.96        56

          accuracy                           0.84       500
         macro avg       0.84      0.84      0.83       500
      weighted avg       0.86      0.84      0.84       500
```

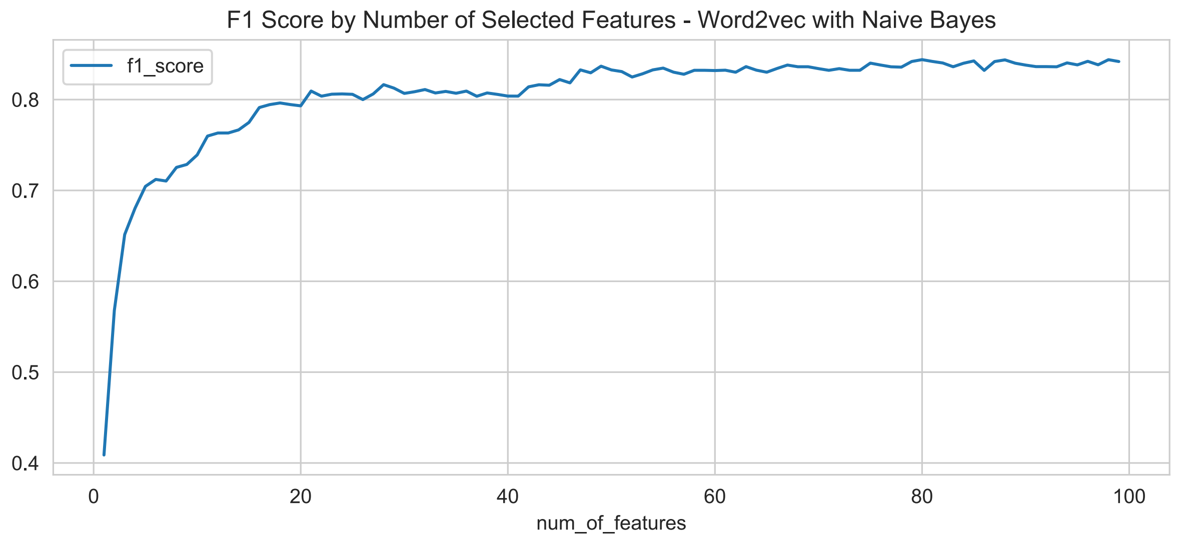
Feature Selection - Word2Vec Features with Naive Bayes Model

In [88]:
```
rows = []
for i in range(1, 100, 1): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(model_w2v_nb, i, w2v_feature_array, y_train, w2v_test_array, y_test, scaler_min_ma
    rows.append([i, results_i.f1_score, results_i.accuracy])

sel_nb_w2v_df = pd.DataFrame(rows, columns=["num_of_features", "f1_score", "accuracy"])
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [89]:
```
sel_nb_w2v_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - Word2vec with Naive Bay
```

Out[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26655495988>
```



In [90]:
```
Opt_no_of_feat = int(sel_nb_w2v_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
Opt_no_of_feat
sel_nb_w2v_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[90]:

	num_of_features	f1_score	accuracy
79	80	0.8439533	0.8400000
97	98	0.8439154	0.8400000
87	88	0.8437020	0.8400000
84	85	0.8425461	0.8380000
95	96	0.8421248	0.8380000

In [91]:
```
results_nb_w2v = SelectBestModelFeatures_Chi(model_w2v_nb, Opt_no_of_feat, w2v_feature_array, y_train, w2v_test_array, y_test,
# Save benchmark output
Save_Benchmark("Word2Vec Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "Word2Vec", results_nb_w2v, False, Fa
df_benchmarks
```

Out[91]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Word2Vec Naive Bayes Baseline	Word2Vec	0.8557780	0.8380000	0.8419654	0.8380000
1	Word2Vec Naive Bayes Optimal Features Selected: 80	Word2Vec	0.8576544	0.8400000	0.8439533	0.8400000

Word2vec features Extraction with Fastext Model

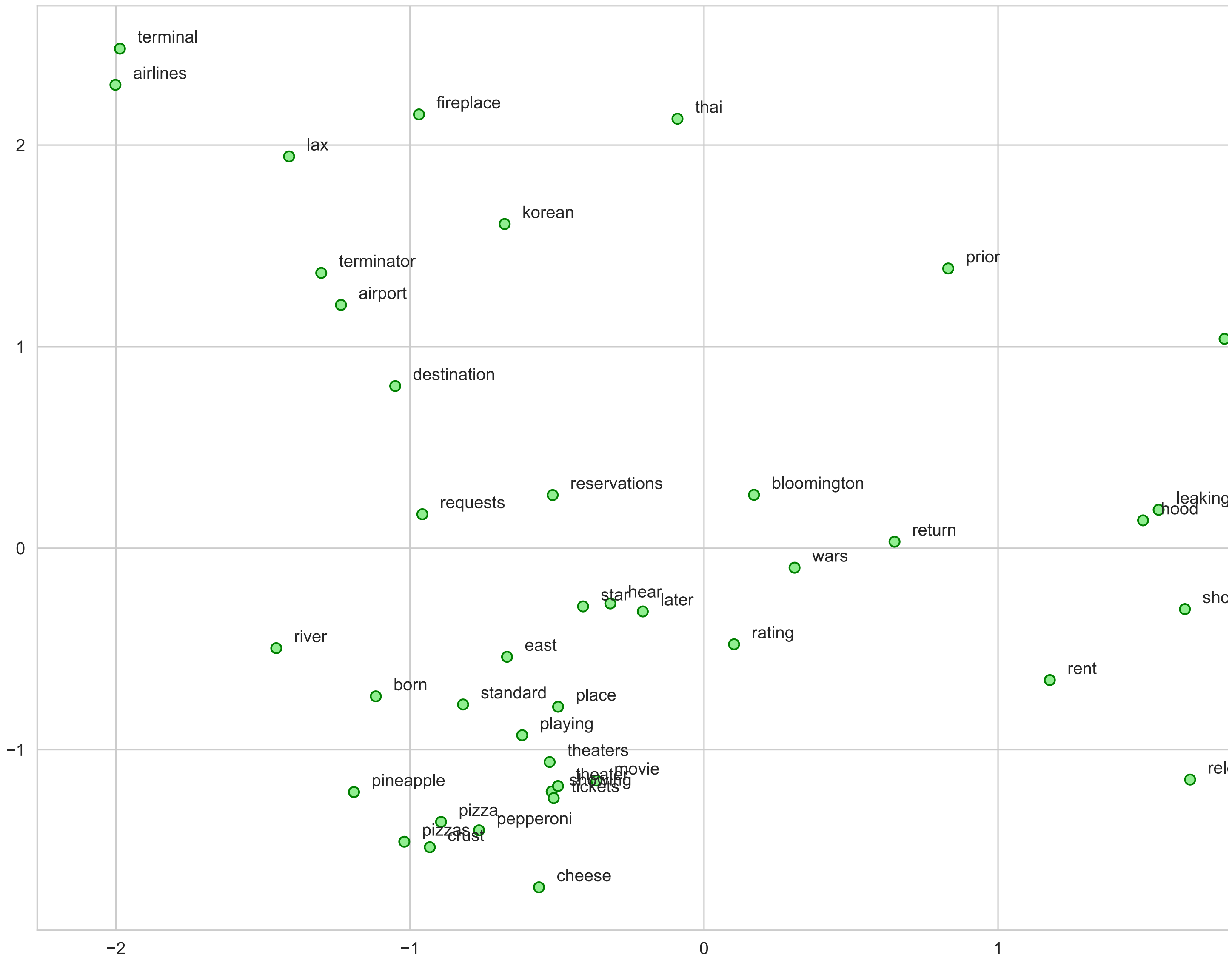
In [92]:
```
from gensim.models.fasttext import FastText

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in X_train]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 50          # Context window size
min_word_count = 5   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words


ft_model = FastText(tokenized_corpus, size=feature_size, window=window_context,
                    min_count=min_word_count,sample=sample, sg=1, iter=50)
```

In [93]:
```
# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in ft_model.wv.most_similar([search_term], topn=5)]
                  for search_term in ['rental', 'pizza', 'terminator', 'star', 'audi', 'east', 'korean','playing']}
similar_words
```

Out[93]:
```
{'rental': ['rent', 'warranty', 'release', 'diagnose', 'shouldnt'],
 'pizza': ['pepperoni', 'crust', 'cheese', 'pineapple', 'pizzas'],
 'terminator': ['terminal', 'airport', 'airlines', 'destination', 'lax'],
 'star': ['born', 'wars', 'rating', 'hear', 'return'],
 'audi': ['motor', 'leaking', 'manual', 'squealing', 'hood'],
 'east': ['river', 'bloomington', 'place', 'later', 'standard'],
 'korean': ['thai', 'reservations', 'prior', 'requests', 'fireplace'],
 'playing': ['showing', 'tickets', 'movie', 'theater', 'theaters']}
```

PCA on Fasttext Model

```
In [94]: from sklearn.decomposition import PCA

         words = sum([[k] + v for k, v in similar_words.items()], [])
         wvs = ft_model.wv[words]

         pca = PCA(n_components=2)
         np.set_printoptions(suppress=True)
         P = pca.fit_transform(wvs)
         labels = words

         plt.figure(figsize=(18, 10))
         plt.scatter(P[:, 0], P[:, 1], c='lightgreen', edgecolors='g')
         for label, x, y in zip(labels, P[:, 0], P[:, 1]):
             plt.annotate(label, xy=(x+0.06, y+0.03), xytext=(0, 0), textcoords='offset points')
```



```
In [95]: print(P.shape)

         (48, 2)
```

In [96]:
```
ft_model.wv['rental']
```

Out[96]:
```
array([-0.1823672 ,  0.38009638, -1.0052938 ,  0.7145282 , -0.11232223,
       -0.21826684,  0.407818  , -0.12781262,  0.34740442,  0.4387549 ,
        0.83798283,  0.40470704,  0.5549515 ,  0.27541617,  0.34209627,
        0.5072392 , -0.6549379 ,  0.09226233, -0.08641256, -0.48588362,
       -0.10057411, -1.0569569 ,  0.8498051 ,  0.669488  , -0.6443245 ,
       -0.43327567, -0.6818331 , -0.04491991, -0.17801598, -0.51823354,
        0.01526353,  0.0839986 ,  0.3688926 ,  1.4134214 , -0.48686066,
       -0.3249323 ,  1.3029019 ,  0.8207036 ,  1.6860458 ,  0.14209466,
        0.7031216 , -0.06915611, -0.8978621 ,  0.20565915,  0.61560935,
        0.14275207, -0.25982755,  0.97517586, -0.01436888, -0.18807109,
        0.15000644, -0.7596845 , -0.08836053,  0.5748159 ,  0.28015858,
        0.3685647 , -0.2454766 ,  0.10389301, -0.00854702, -0.5446641 ,
        0.6692645 , -0.77528125,  0.41549638, -0.26726422, -0.8171256 ,
       -0.4398857 , -0.46669653, -0.13181633, -0.11859373, -0.1515544 ,
        0.9451388 ,  0.63723797, -0.63304305,  0.1696579 ,  0.01498261,
       -0.02495871,  0.91755605, -0.04646447,  0.12400665,  0.9874966 ,
        0.12032788, -0.9476534 , -0.21852538,  0.33163622, -1.2796878 ,
        0.58851314, -0.20776466, -0.6023003 ,  0.8587102 ,  0.07792503,
        1.0467081 ,  0.5977417 , -0.16623087, -0.32082596,  0.22874902,
        0.47381738, -0.43015108, -0.70266104,  0.1747856 , -0.33276647],
      dtype=float32)
```

In [97]:
```
print(ft_model.wv.similarity(w1='pizza', w2='born'))
print(ft_model.wv.similarity(w1='playing', w2='movie'))
```

```
0.21294737
0.8077414
```

In [98]:
```
st1 = "'tickets movie showing john"
print('Odd one out for [',st1, ']:', ft_model.wv.doesnt_match(st1.split()))

st2 = "pepperoni pizzas cheese pies"
print('Odd one out for [',st2, ']:', ft_model.wv.doesnt_match(st2.split()))
```

```
Odd one out for [ 'tickets movie showing john ]: 'tickets
Odd one out for [ pepperoni pizzas cheese pies ]: pies

C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\gensim\models\keyedvectors.py:877: FutureWarning: arrays to stack must be
passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is deprecated as of
NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

Word2Vec Features from Fastext Benchmarking with Naive Bayes Model

In [99]:
```
w2v_ft_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=ft_model,
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated `__getite
m__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

Out[99]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	0.1873218	-0.4735788	0.9725689	-1.5548968	-0.0483634	0.5444516	-0.2362861	-0.3395859	0.9621134	-0.7932116	-0.2347472	-0.9821420	1.56239
1	0.0178411	-0.0560393	0.9839213	-0.4040909	0.0176693	0.3752609	-0.5976276	0.0030219	0.0433316	-0.4721514	-0.4829037	-0.3136707	1.12700
2	-0.1484596	-0.1243862	-0.6997141	0.6127047	-0.7421537	0.5409077	0.1489333	0.8100881	1.0104762	-0.6006313	-0.2722385	-0.4159432	1.08594
3	0.1373973	0.0020785	0.7520878	-0.3897839	-0.1119516	0.1614587	0.2419387	-0.2300063	0.4225348	1.2530223	-0.3051739	0.1348216	0.82955
4	0.8935827	-0.6581176	1.0229267	0.1918840	0.0763810	0.0853404	0.8857397	-0.4736567	0.1782164	0.6857576	-0.0704821	0.4307056	1.45702
...	...	...	...	...	...	...	...	...	...	...	...	...	
1495	0.6359248	-0.8073446	1.2037342	-0.1941031	0.1144299	0.5235185	0.3673428	-0.1585717	0.4354501	0.3227923	0.2609709	-0.3090171	0.05004
1496	0.9471351	-0.4739289	1.4136777	0.3378935	0.2156104	-0.4278203	0.4220656	0.3073745	-1.0997889	-0.7592334	0.2078579	-0.3439014	-1.35721
1497	0.9265414	-0.4210868	-0.4615308	0.6569034	-0.2054187	0.8343573	0.6735410	-0.0878270	1.2543531	-0.3141014	-0.1724857	0.1617612	2.09583
1498	0.0983251	-0.6904276	1.0041442	-0.5233796	0.3611044	0.3317488	-0.1283708	0.0498702	0.2342417	0.2593339	0.1916878	0.3933110	0.99180
1499	0.2193377	-0.1996251	0.8627414	-0.4336795	-0.0377298	0.0567343	0.3240563	0.0392423	0.5753537	0.1041630	0.2219171	0.2447023	0.76004

1500 rows × 100 columns

In [100]:
```
w2v_ft_test_array = averaged_word_vectorizer(corpus=tokenized_corpus_test, model=ft_model,
                                              num_features=feature_size)
```

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated `__getite
m__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

In [101]:
```
model_ft_nb = GaussianNB()
results_nb_ft = SelectBestModelFeatures_Chi(model_ft_nb, 100, w2v_ft_feature_array, y_train, w2v_ft_test_array, y_test, scaler
# Save benchmark output
Save_Benchmark("Word2Vec Fastext Naive Bayes Baseline", "Word2Vec_FT", results_nb_ft, True, False)
df_benchmarks
```

Out[101]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Word2Vec Fastext Naive Bayes Baseline	Word2Vec_FT	0.8346695	0.7760000	0.7700025	0.7760000

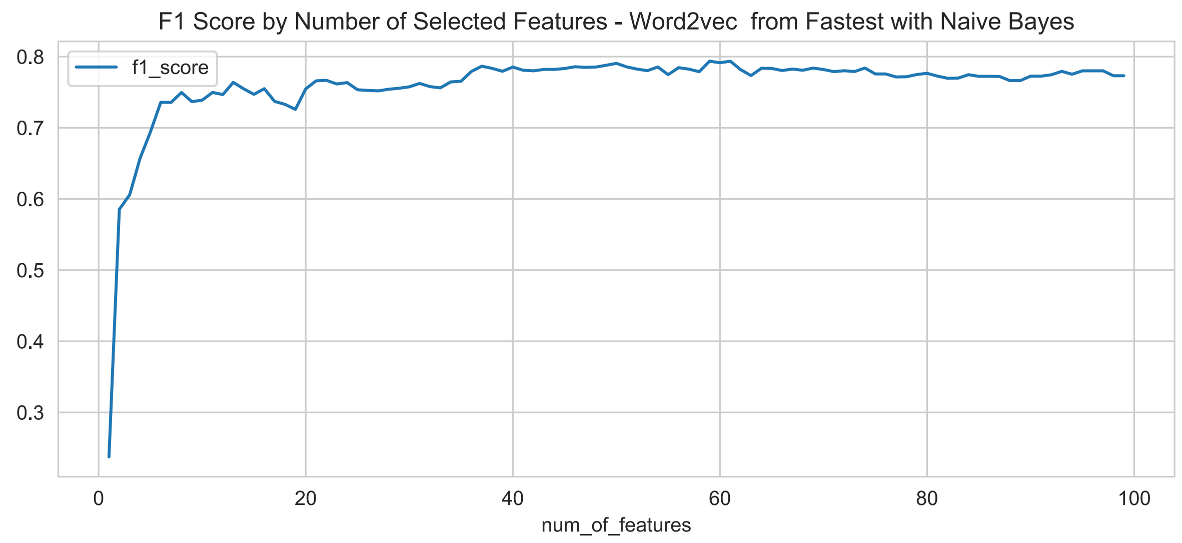
Word2Vec from Fastext Model Feature Selction with Naive Bayes Model

In [102]:
```
rows = []
for i in range(1, 100, 1): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(model_ft_nb, i, w2v_ft_feature_array, y_train, w2v_ft_test_array, y_test, scaler_m
    rows.append([i, results_i.f1_score, results_i.accuracy])

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

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [103]:
```
sel_nb_ft_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - Word2vec  from Fastest w
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x266565b7188>
```



In [104]:
```
Opt_no_of_feat = int(sel_nb_ft_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
Opt_no_of_feat
sel_nb_ft_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[104]:

	num_of_features	f1_score	accuracy
58	59	0.7935966	0.7940000
60	61	0.7935701	0.7940000
59	60	0.7914108	0.7920000
49	50	0.7905948	0.7920000
48	49	0.7878157	0.7900000

Benchmarking Word2Vec Fastext with Naive Bayes on Optimal number of Features

In [105]:
```
results_nb_ft = SelectBestModelFeatures_Chi(model_ft_nb, Opt_no_of_feat, w2v_ft_feature_array, y_train, w2v_ft_test_array, y_t
# Save benchmark output
Save_Benchmark("Word2Vec from Fastest Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "Word2Vec_FT", results_r
df_benchmarks
```

Out[105]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	Word2Vec Fastext Naive Bayes Baseline	Word2Vec_FT	0.8346695	0.7760000	0.7700025	0.7760000
1	Word2Vec from Fastest Naive Bayes Optimal Features Selected: 59	Word2Vec_FT	0.8598375	0.7940000	0.7935966	0.7940000

Feature Extraction: Glove Word Embeddings

GloVe Embeddings with spaCy

In [106]:
```
import spacy

nlp = spacy.load('en_vectors_web_lg')

total_vectors = len(nlp.vocab.vectors)
print('Total word vectors:', total_vectors)
```

```
Total word vectors: 1070971
```

Visualize GloVe word embeddings

In [107]:
```
unique_words = list(set([word for sublist in [doc.split() for doc in X_train] for word in sublist]))
word_glove_vectors = np.array([nlp(word).vector for word in unique_words])

pd.DataFrame(word_glove_vectors, index=unique_words)
```

Out[107]:

	0	1	2	3	4	5	6	7	8	9	10	11
understandi	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
indicated	0.1367500	0.3006500	-0.1054900	0.1601100	-0.1565900	0.0792680	-0.1848600	0.1041000	-0.1670500	2.3329999	0.3698800	0.2742200
texas	-0.5828600	0.5061000	0.1451900	-0.4449000	0.9426200	0.1022800	0.0637430	0.7265400	0.4655400	1.3874000	-0.9629200	0.1337300
ideal	0.3278400	0.5020700	-0.1448700	-0.1316000	0.5724900	-0.1279800	0.1261600	0.0742940	0.1124000	1.5570000	-0.0253190	-0.0422290
straving	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
...	...	...	...	...	...	...	...	...	...	...	...	...
barboursville	-0.3303600	-0.4823700	-0.0547950	0.6250300	0.2557200	0.0987390	0.3732800	0.9357300	-0.2250900	-1.1707000	0.1685300	0.2894500
track	0.5800400	0.4986700	-0.3614300	0.0628030	0.2676300	0.3472500	-0.5100900	-0.5865200	-0.3844300	1.8924000	-0.1764900	0.5007900
breads	-0.5264700	-0.0452860	0.3229000	-0.5016800	-0.1912900	0.5395500	0.2391000	0.8764800	-0.3858500	0.5463300	-0.1653100	0.6609800
replace	0.4306200	0.0214340	-0.1469600	0.2958300	-0.1506200	-0.3076700	-0.3531000	-0.1096300	-0.2808100	1.6600000	0.3781200	-0.0606600
pistol	-0.0125060	-0.1598700	-0.1212900	-0.2003600	-0.0347120	-0.9868500	0.1741200	-0.0303020	-0.4535200	0.9137700	-0.2863800	-0.2115600

7208 rows × 300 columns

```
In [108]: unique_words_test = list(set([word for sublist in [doc.split() for doc in X_test] for word in sublist]))
          word_glove_vectors_test = np.array([nlp(word).vector for word in unique_words_test])
          print(word_glove_vectors_test.shape)
```

```
(4083, 300)
```

GloVe Embeddings with Flair

```
In [109]: from flair.embeddings import WordEmbeddings, DocumentRNNEmbeddings

          glove_embedding = WordEmbeddings('glove')
          document_embeddings = DocumentRNNEmbeddings([glove_embedding])
```

```
In [110]: from flair.embeddings import Sentence

          # create an example sentence
          sentence = Sentence('The grass is green . And the sky is blue .')
          # embed the sentence with our document embedding
          document_embeddings.embed(sentence)
          # now check out the embedded sentence.
          print(sentence.get_embedding())
```

```
tensor([-0.2709,  0.3972,  0.1081,  0.0119,  0.0108, -0.0378,  0.3193,  0.0064,
        -0.0581,  0.3215,  0.1315,  0.2482, -0.1375,  0.0628,  0.1889, -0.4354,
         0.3966, -0.0217, -0.1223,  0.4234,  0.1926,  0.1787,  0.1651, -0.4583,
        -0.2747,  0.2594,  0.1016,  0.1648,  0.1303,  0.1427,  0.1964,  0.1226,
        -0.0198, -0.0187,  0.2192, -0.1632,  0.0610,  0.0512, -0.0577,  0.1941,
         0.5048,  0.1164,  0.4875,  0.3204, -0.2792, -0.0133,  0.0200,  0.3036,
        -0.3189, -0.0376,  0.0185, -0.2735,  0.2740, -0.2395, -0.0462,  0.3991,
         0.1392, -0.3327, -0.2154,  0.0279, -0.0468,  0.1200,  0.0236,  0.1842,
        -0.2395,  0.0079,  0.1869, -0.2945, -0.1159, -0.1012,  0.4943,  0.1650,
        -0.1132, -0.1934,  0.3647, -0.2932,  0.1885, -0.1305, -0.1697, -0.1395,
         0.0550,  0.2225,  0.1731,  0.0885, -0.2817,  0.0232, -0.2688, -0.4629,
         0.0754, -0.1194,  0.0802,  0.0167,  0.0675, -0.0945, -0.0038,  0.1645,
        -0.0534,  0.0594,  0.4695,  0.1307,  0.1198,  0.2824,  0.3201, -0.1236,
         0.1547,  0.2117, -0.1998, -0.2111, -0.0796,  0.0941,  0.1859, -0.0120,
        -0.0862,  0.0269,  0.2724, -0.3300,  0.0130, -0.0777,  0.3352,  0.1060,
        -0.0261, -0.0567,  0.0457,  0.1420, -0.1067,  0.1094,  0.2033,  0.1986],
       grad_fn=<CatBackward>)
```

```
In [111]: from nltk.tokenize import word_tokenize

          def Get_Glove_Features(corpus):
              dataset_size = len(corpus)
              X = np.zeros((dataset_size, 128))
              for iter in range(0, dataset_size):
                  text = corpus[iter]
                  if (text == ""):
                      text = "blank"
                  sentence = Sentence(text)
                  document_embeddings.embed(sentence)
                  X[iter] = sentence.get_embedding().detach().numpy()
              return X
```

```
In [112]: x_train_glove = Get_Glove_Features(X_train)
          x_test_glove = Get_Glove_Features(X_test)
          print(x_train_glove.shape, x_test_glove.shape)
```

```
(1500, 128) (500, 128)
```

In [113]:
```python
pd.DataFrame(x_test_glove)
```

Out[113]:

	0	1	2	3	4	5	6	7	8	9	10	11	1
0	-0.0196590	0.1428135	0.1311479	0.4675573	-0.1533206	-0.1925505	0.1357332	0.1593048	-0.0841656	0.2091627	0.1411715	0.2832936	-0.25233
1	-0.1760286	0.2079453	-0.0541483	0.3548227	-0.1485749	-0.2884350	0.3155001	-0.1812025	-0.2101861	0.1337148	0.1455723	0.1289710	-0.10025
2	-0.2140987	0.3028506	0.0540561	-0.0383727	0.1399113	-0.1364345	0.0314286	-0.0056308	-0.1821644	0.1494152	-0.0262595	0.0205132	-0.28149
3	-0.1200199	0.1093690	0.0670990	0.3877644	-0.1117848	-0.0157119	0.2777641	-0.0160455	-0.0011404	0.1085507	0.1085959	-0.1268610	-0.14650
4	-0.1058632	-0.0334023	-0.0221114	0.0517342	-0.0246841	-0.1919016	-0.0212520	-0.2525868	0.0356943	-0.1352948	-0.0528922	-0.0745253	-0.19014
...	...	...	...	...	...	...	...	...	...	...	...	...	.
495	-0.2426529	0.0870831	0.0204297	0.0948525	-0.0434478	-0.2026001	-0.1709892	0.1175784	-0.2879112	0.0860657	-0.0919038	0.1535550	-0.27727
496	-0.2423518	0.1615450	-0.0376042	0.2693284	-0.0692964	-0.2564172	0.0988994	0.0357920	-0.1245061	0.0818953	0.0447408	-0.0821921	-0.15871
497	-0.2549154	0.2903412	0.0189290	-0.1009964	0.1244586	-0.1737880	0.0795393	-0.0667115	-0.1096362	0.1469434	0.1344832	0.0203100	-0.22576
498	-0.3782700	0.1514379	0.2693895	0.1813663	-0.0644215	-0.0622330	0.1572477	-0.0625545	-0.1281991	-0.0345937	0.1299711	0.1168005	-0.25350
499	-0.2233154	0.2917125	0.0330171	-0.0468356	0.1666162	-0.1151662	-0.0106977	-0.0162728	-0.1762970	0.1475689	-0.0101359	0.0002981	-0.24346

500 rows × 128 columns

In [114]:
```python
from sklearn.naive_bayes import GaussianNB

model_glove_nb = GaussianNB()
results_nb_glove = Build_Model(model_glove_nb, x_train_glove, y_train, x_test_glove, y_test)
# Save benchmark output
# rows_benchmarks.append(["Glove with Naive Bayes All Features", f1_nb_glove, accuracy_nb_glove])
# df_benchmarks = pd.DataFrame(rows_benchmarks, columns=["Features_Benchedmarked", "f1_score", "accuracy"])
# df_benchmarks
```

In [115]:
```python
print(results_nb_glove.report)
```

```
                    precision    recall  f1-score   support

auto-repair-appt-1       0.45      0.23      0.31        64
   coffee-ordering       0.21      0.16      0.18        58
      movie-finder       0.08      0.20      0.11        15
   movie-tickets-1       0.33      0.13      0.19        61
   movie-tickets-2       0.35      0.22      0.27        58
   movie-tickets-3       0.16      0.29      0.21        49
    pizza-ordering       0.24      0.20      0.22        56
  restaurant-table       0.34      0.38      0.36        63
restaurant-table-3       0.07      0.20      0.11        20
         uber-lyft       0.22      0.25      0.23        56

          accuracy                           0.23       500
         macro avg       0.25      0.23      0.22       500
      weighted avg       0.28      0.23      0.24       500
```

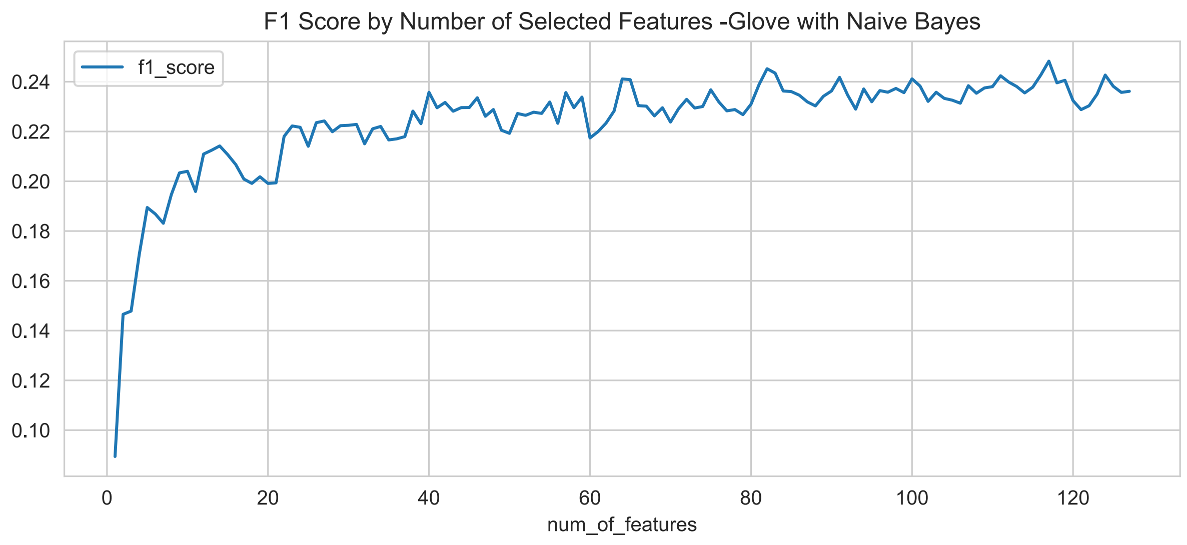
Feature Selection on Glove Features with Naive Bayes Model

```
In [116]: rows = []
          for i in range(1, 128, 1): # range(a, b, c) will count from a to b by intervals of c.
              results_i = SelectBestModelFeatures_Chi(model_glove_nb, i, x_train_glove, y_train, x_test_glove, y_test, scaler_min_max)
              rows.append([i, results_i.f1_score, results_i.accuracy])

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

```
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\pauld\Anaconda3\envs\CP\lib\site-packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Precisio
n is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
In [117]: sel_nb_glove_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features -Glove with Naive Bayes
```

```
Out[117]: <matplotlib.axes._subplots.AxesSubplot at 0x266e196b848>
```



```
In [118]: Opt_no_of_feat = int(sel_nb_glove_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
          Opt_no_of_feat
          sel_nb_glove_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[118]:

	num_of_features	f1_score	accuracy
116	117	0.2482685	0.2380000
81	82	0.2452180	0.2420000
82	83	0.2434520	0.2380000
115	116	0.2427089	0.2320000
123	124	0.2426739	0.2360000

```
In [119]: results_nb_glove = SelectBestModelFeatures_Chi(model_glove_nb, Opt_no_of_feat, x_train_glove, y_train, x_test_glove, y_test, s

          # Save benchmark output
          # Save_Benchmark("Glove Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "GLoVe", results_nb_glove, False, Fals
          # df_benchmarks
```

Leave the Glove Feature result out for now since it clearly is problematic

Combining Features

Combine BOW and BAG of nGrams

```
In [120]: def Get_Combined_Features(feat_1, feat_2):
              row_size = len(feat_1)
              col_size_1 = np.size(feat_1, axis=1)
              col_size_total = np.size(feat_1, axis=1) + np.size(feat_2, axis=1)
              X = np.zeros((row_size, col_size_total))

              for i in range(0, row_size):
                  for j in range(0, col_size_1):
                      X[i, j] = feat_1[i, j]

                  for k in range(col_size_1, col_size_total):
                      X[i, k] = feat_2[i, k - col_size_1]
              return X
```

Combine Features Arrays together

In [121]:
```
# Get Scaled BOW Features
x_bow_train_norm, x_bow_test_norn = Get_Scaled_Features(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test

# Add Bag of nGrams
x_bong_train_norm, x_bong_test_norn = Get_Scaled_Features(results_bong_opt.x_train_sel, y_train, results_bong_opt.x_test_sel,
x_train_bow_bong = Get_Combined_Features(x_bow_train_norm, x_bong_train_norm)
x_test_bow_bong = Get_Combined_Features(x_bow_test_norn, x_bong_test_norn)

# Add TF-IDF
# x_tfidf_train_norm, x_tfidf_test_norn = Get_Scaled_Features(results_tf_nb_opt.x_train_sel, y_train, results_tf_nb_opt.x_test
# x_train_bow_bong = Get_Combined_Features(x_train_bow_bong, x_tfidf_train_norm)
# x_test_bow_bong = Get_Combined_Features(x_test_bow_bong, x_tfidf_test_norn)

# Add Word2Vec
# x_w2v_train_norm, x_w2v_test_norn = Get_Scaled_Features(results_nb_w2v.x_train_sel, y_train, results_nb_w2v.x_test_sel, y_te
# x_train_bow_bong = Get_Combined_Features(x_train_bow_bong, x_w2v_train_norm)
# x_test_bow_bong = Get_Combined_Features(x_test_bow_bong, x_w2v_test_norn)
```

In [122]:
```
print(x_train_bow_bong.shape)
print(x_test_bow_bong.shape)
```
```
(1500, 6158)
(500, 6158)
```

In [123]:
```
pd.DataFrame(x_test_bow_bong)
```

Out[123]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
0	0.0000000	0.2222222	0.0000000	0.1111111	0.1250000	0.1333333	0.0000000	0.2222222	0.3750000	0.0000000	0.0000000	0.0000000	0.0000000	0.000
1	0.3076923	0.0000000	0.1818182	0.4444444	0.3750000	0.0000000	0.5454545	0.0000000	0.2500000	0.2857143	0.0000000	0.1111111	0.3333333	0.000
2	0.1538462	0.2222222	0.1818182	0.1111111	0.1250000	0.0000000	0.2727273	0.0000000	0.1250000	0.1428571	0.0000000	0.0000000	0.0000000	0.000
3	0.1538462	0.2222222	0.5454545	0.1111111	0.1250000	0.1333333	0.0909091	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.000
4	0.2307692	0.1111111	0.0000000	0.2222222	0.2500000	0.0000000	0.0000000	0.0000000	0.5000000	0.1428571	0.0000000	0.0000000	0.0000000	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
495	0.2307692	0.0000000	0.1818182	0.1111111	0.1250000	0.0000000	0.1818182	0.0000000	0.1250000	0.2857143	0.0000000	0.0000000	0.2222222	0.000
496	0.0000000	0.0000000	0.0000000	0.3333333	0.0000000	0.0000000	0.0000000	0.0000000	0.3750000	0.2857143	0.0000000	0.0000000	0.0000000	0.000
497	0.0000000	0.1111111	0.0000000	0.0000000	0.1250000	0.4666667	0.0000000	0.0000000	0.1250000	0.1428571	0.4500000	0.0000000	0.0000000	0.000
498	0.0000000	0.6666667	0.0000000	0.0000000	0.2500000	0.0666667	0.0000000	0.0000000	0.1250000	0.0000000	0.2000000	0.0000000	0.1111111	0.000
499	0.3846154	0.1111111	0.0000000	0.1111111	0.1250000	0.0000000	0.0000000	0.1111111	0.2500000	0.2857143	0.0000000	0.0000000	0.0000000	0.000

500 rows × 6158 columns

In [124]:
```
# model_bow_bong = GaussianNB()
model_bow_bong = MultinomialNB()
results_nb_bow_bong = Build_Model(model_bow_bong, x_train_bow_bong, y_train, x_test_bow_bong, y_test)
```

In [125]:
```
Save_Benchmark("BOW and Bag of N-Grams Combined Baseline", "BOW_BONG", results_nb_bow_bong, True, False)
df_benchmarks
```

Out[125]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW and Bag of N-Grams Combined Baseline	BOW_BONG	0.8721995	0.8580000	0.8480611	0.8580000

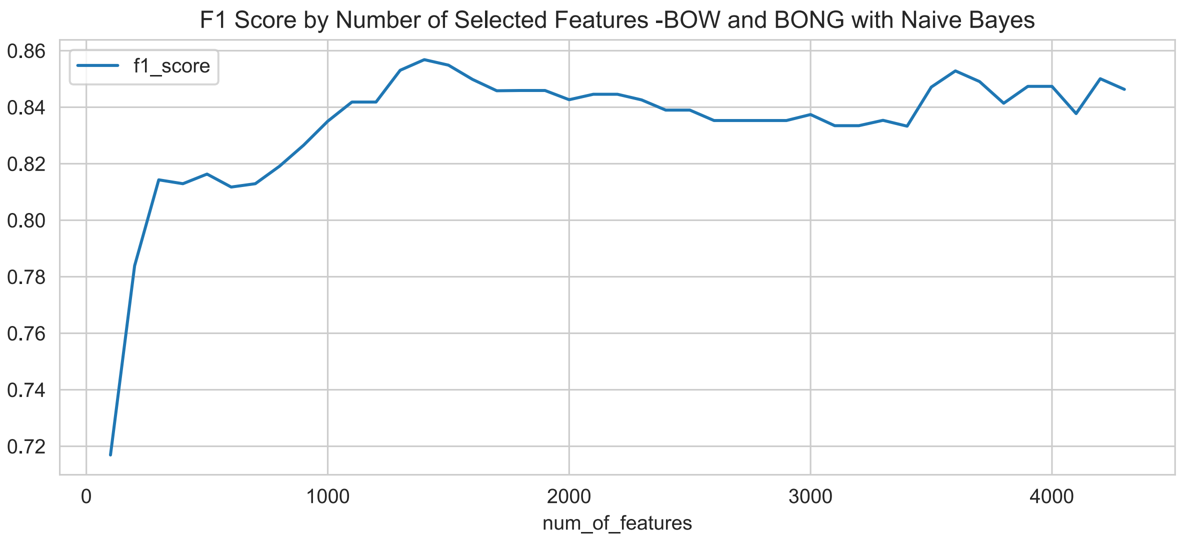
In [126]:
```
print("Label" + results_nb_bow_bong.report)
```

```
Label                    precision    recall  f1-score   support

auto-repair-appt-1       0.94      1.00      0.97        64
   coffee-ordering       0.98      0.98      0.98        58
      movie-finder       1.00      0.93      0.97        15
   movie-tickets-1       0.69      0.84      0.76        61
   movie-tickets-2       0.63      0.64      0.63        58
   movie-tickets-3       0.94      0.67      0.79        49
    pizza-ordering       0.98      0.98      0.98        56
  restaurant-table       0.78      0.97      0.87        63
restaurant-table-3       1.00      0.15      0.26        20
         uber-lyft       0.98      0.96      0.97        56

          accuracy                           0.86       500
         macro avg       0.89      0.81      0.82       500
      weighted avg       0.87      0.86      0.85       500
```

In [127]:
```
rows = []
for i in range(100, 4400, 100): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(model_bow_bong, i, x_train_bow_bong, y_train, x_test_bow_bong, y_test, scaler_min_
    rows.append([i, results_i.f1_score, results_i.accuracy])

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

In [128]:
```
sel_nb__bow_bong_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features -BOW and BONG with
```

Out[128]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26761bf0788>
```



In [129]:
```
Opt_no_of_feat = int(sel_nb__bow_bong_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)
sel_nb__bow_bong_df.sort_values(by='f1_score', ascending=False).head(5)
```

```
1350 1450 1
```

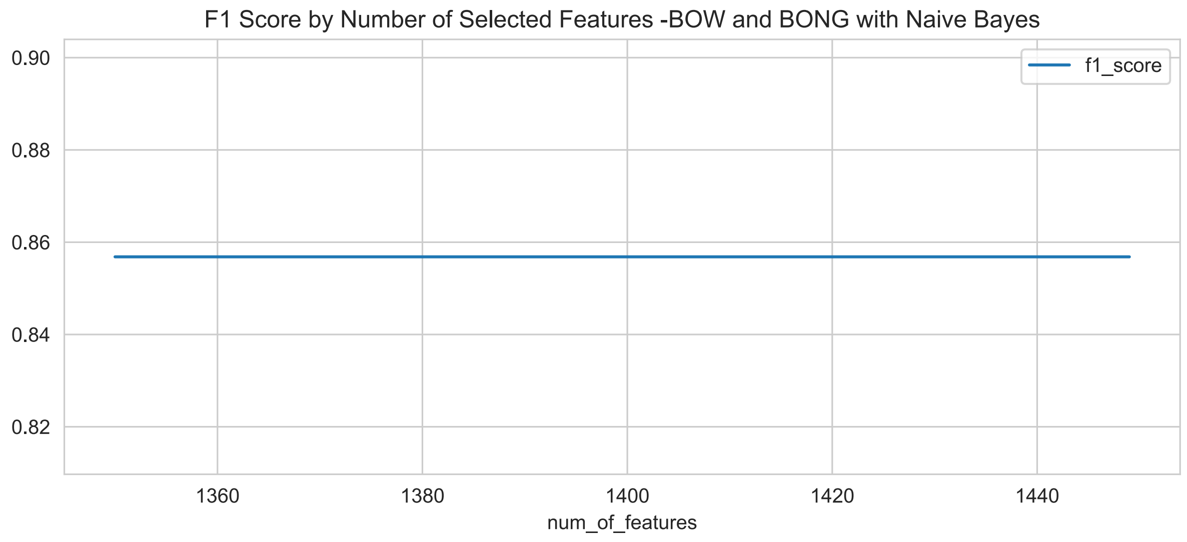
Out[129]:

	num_of_features	f1_score	accuracy
13	1400	0.8568311	0.8600000
14	1500	0.8548868	0.8580000
12	1300	0.8530774	0.8560000
35	3600	0.8528526	0.8620000
41	4200	0.8500844	0.8620000

In [130]:
```
rows = []
for i in range(a, b, c): # range(a, b, c) will count from a to b by intervals of c.
    results_i = SelectBestModelFeatures_Chi(model_bow_bong, Opt_no_of_feat, x_train_bow_bong, y_train, x_test_bow_bong, y_test
    rows.append([i, results_i.f1_score, results_i.accuracy])

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

In [131]:
```
sel_nb__bow_bong_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features -BOW and BONG with
```

Out[131]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26761906948>
```



In [132]:
```
Opt_no_of_feat = int(sel_nb__bow_bong_df.sort_values(by='f1_score', ascending=False).iloc[0]['num_of_features'])
Opt_no_of_feat
sel_nb__bow_bong_df.sort_values(by='f1_score', ascending=False).head(5)
```

Out[132]:

	num_of_features	f1_score	accuracy
0	1350	0.8568311	0.8600000
63	1413	0.8568311	0.8600000
73	1423	0.8568311	0.8600000
72	1422	0.8568311	0.8600000
71	1421	0.8568311	0.8600000

In [133]:
```
#model_bow_bong = GaussianNB() # = MultinomialNB()
results_nb_bow_bong = SelectBestModelFeatures_Chi(model_bow_bong, Opt_no_of_feat, x_train_bow_bong, y_train, x_test_bow_bong,
Save_Benchmark("BOW + Bag of NGrams Top: " + str(Opt_no_of_feat) + " Features with Naive Bayes", "BOW_BONG", results_nb_bow_bc
df_benchmarks
```

Out[133]:

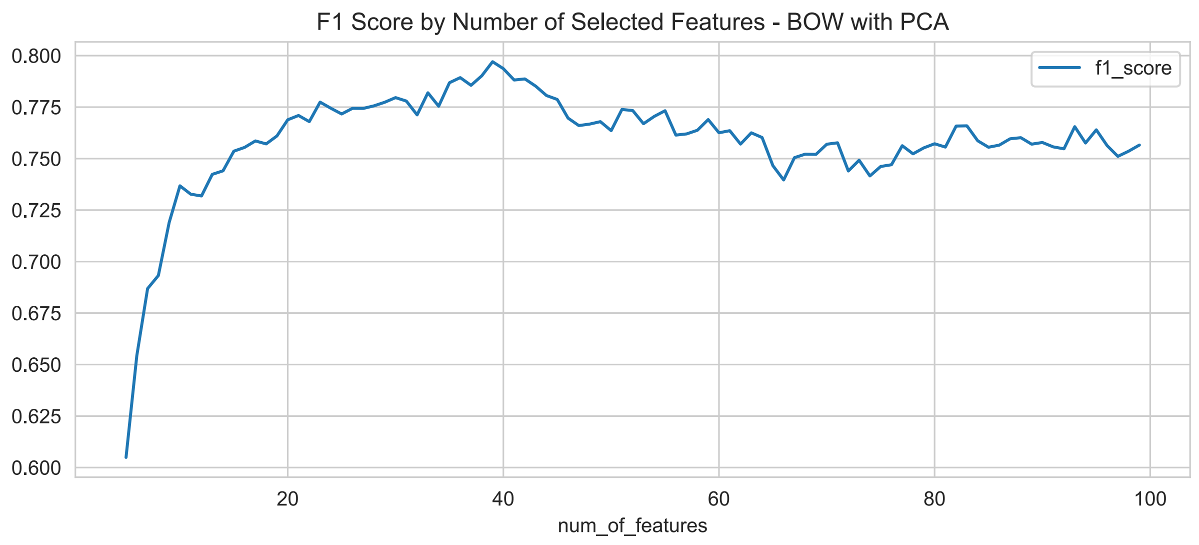
	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW and Bag of N-Grams Combined Baseline	BOW_BONG	0.8721995	0.8580000	0.8480611	0.8580000
1	BOW + Bag of NGrams Top: 1350 Features with Naive Bayes	BOW_BONG	0.8687971	0.8580000	0.8548460	0.8580000

Try PCA Feature Extraction on the BOW Model

In [134]:
```
from sklearn.decomposition import PCA

# Define PCA Selection Function
def Get_PCA_Features(i,  X_train_pca, y_train_pca, X_test_pca, y_test_pca):
    pca = PCA(n_components=i)
    fit = pca.fit(X_train_pca, y_train_pca)
    pca_train = fit.transform(X_train_pca)
    pca_test = fit.transform(X_test_pca)
    return pca_train, pca_test
```

```
In [135]: # Loop through different no. of component values
          model_nb_bow = GaussianNB()
          rows = []
          for i in range(5, 100, 1): # range(a, b, c) will count from a to b by intervals of c.
              x_train_pca_i, x_test_pca_i = Get_PCA_Features(i,  X_train_bow, y_train, X_test_bow, y_test)
              results_i = Build_Model(model_nb_bow, x_train_pca_i, y_train, x_test_pca_i, y_test)
              rows.append([i, results_i.f1_score, results_i.accuracy])
          acc_df = pd.DataFrame(rows, columns=["num_of_features", "f1_score", "accuracy"])
```

```
In [136]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - BOW with PCA", figsize=(10, 4)

Out[136]: <matplotlib.axes._subplots.AxesSubplot at 0x267627bfe08>
```



```
In [137]: 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)

          39

Out[137]:
```

	num_of_features	f1_score	accuracy
34	39	0.7970447	0.7940000
35	40	0.7936665	0.7920000
33	38	0.7902101	0.7860000
31	36	0.7893219	0.7860000
37	42	0.7886873	0.7860000

```
In [138]: x_train_pca, x_test_pca = Get_PCA_Features(Opt_no_of_feat,  X_train_bow, y_train, X_test_bow, y_test)
          results_bow_pca = Build_Model(model_nb_bow, x_train_pca, y_train, x_test_pca, y_test)
          Save_Benchmark("BOW With Top: " + str(Opt_no_of_feat) + "  PCA Components Seleted", "BOW_PCA", results_bow_pca, True, False)
          df_benchmarks

Out[138]:
```

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW With Top: 39 PCA Components Seleted	BOW_PCA	0.8093699	0.7920000	0.7945170	0.7920000

In [139]:
```
pd.DataFrame(x_train_pca)
```
Out[139]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	3.2508397	2.3423283	-3.3287123	1.2432846	-0.4818184	-0.5617054	0.9005430	-1.2353033	-2.1836135	0.5994084	-0.9415038	-2.6467423	-1.31183
1	2.1058651	2.2693454	-1.2190922	0.7453415	-0.5049125	-0.5814228	0.0428271	-2.4933518	-0.6798187	-1.0518233	-0.0645039	1.4886256	0.21667
2	-1.3993901	-1.2738850	-0.3952355	1.3882063	1.0359852	-2.5372349	0.3738042	-1.9966860	-1.5240268	-0.5956319	0.5132392	0.8549735	1.30789
3	3.1407774	-2.3881468	-4.2071201	-2.0824285	-0.6172805	2.1562496	0.9542362	2.3686900	0.9512897	-0.6298431	0.7119423	0.5467152	1.09164
4	5.1812297	0.2280930	5.2504999	-2.4289578	-1.0404616	0.9418258	-0.2385953	0.9077629	-0.4714933	-1.4779061	-0.1980890	-0.3005617	1.09966
...	...	...	...	...	...	...	...	...	...	...	...	...	
1495	0.1200764	2.0194262	-4.5981556	0.7207413	-1.3485184	2.0135445	1.7769123	3.0453646	-0.4183715	-1.2196198	1.1560994	-0.0841336	-1.23739
1496	-3.0241520	-0.4472913	2.0915811	-1.6104284	0.7989843	0.1361926	0.3751697	-1.3271387	-0.4214292	-2.4217892	0.2479778	-0.6376148	-0.28384
1497	-0.5727306	-0.3757216	-1.1531209	1.6954253	0.8097133	-1.9111208	0.2711184	-0.8001599	-2.1088299	1.4629085	0.4246893	-0.3118048	2.69419
1498	3.9770367	2.9613395	2.6870076	-0.7936948	-1.5575576	0.3599110	0.4172481	-0.1201595	-1.4343553	-0.2624879	0.5080779	-1.5388985	-2.07109
1499	2.5189639	-1.2341631	-3.8628389	-0.6691393	-0.2348378	1.0311390	1.2823604	2.2097953	-0.1945616	0.0793196	-0.2750217	-0.5505809	-1.09469

1500 rows × 39 columns

Feature Engineering, Extraction and Selection Final Results

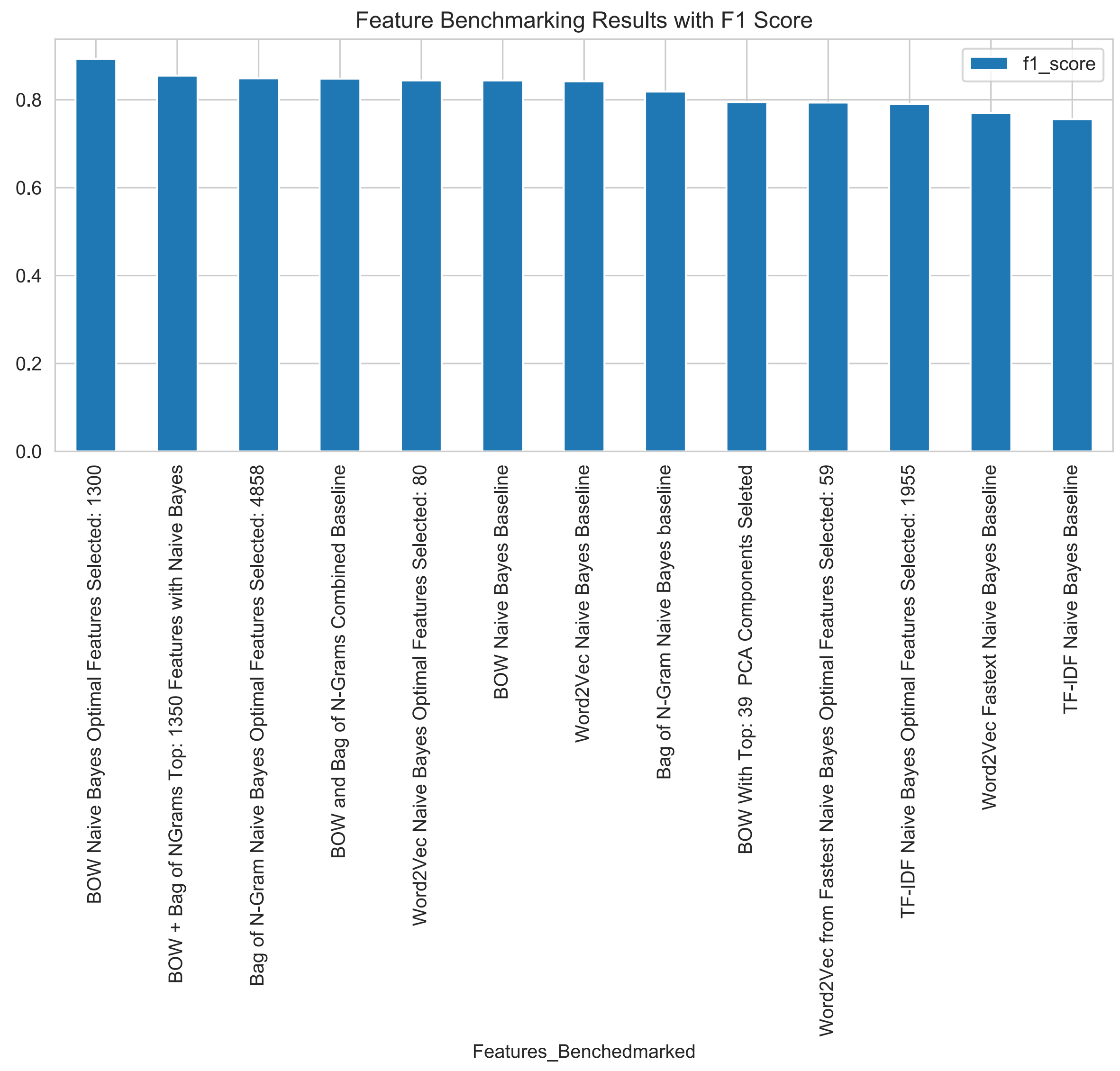
In [140]:
```
# Show All benchmarks
df_benchmarks_all
```
Out[140]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW Naive Bayes Baseline	BOW	0.8394773	0.8600000	0.8438828	0.8600000
1	BOW Naive Bayes Optimal Features Selected: 1300	BOW	0.8956079	0.8940000	0.8931327	0.8940000
2	Bag of N-Gram Naive Bayes baseline	BONG	0.8253737	0.8340000	0.8185937	0.8340000
3	Bag of N-Gram Naive Bayes Optimal Features Selected: 4858	BONG	0.8638235	0.8560000	0.8486926	0.8560000
4	TF-IDF Naive Bayes Baseline	TF-IDF	0.7892568	0.7900000	0.7558597	0.7900000
5	TF-IDF Naive Bayes Optimal Features Selected: 1955	TF-IDF	0.7882401	0.8200000	0.7904632	0.8200000
6	Word2Vec Naive Bayes Baseline	Word2Vec	0.8557780	0.8380000	0.8419654	0.8380000
7	Word2Vec Naive Bayes Optimal Features Selected: 80	Word2Vec	0.8576544	0.8400000	0.8439533	0.8400000
8	Word2Vec Fastext Naive Bayes Baseline	Word2Vec_FT	0.8346695	0.7760000	0.7700025	0.7760000
9	Word2Vec from Fastest Naive Bayes Optimal Features Selected: 59	Word2Vec_FT	0.8598375	0.7940000	0.7935966	0.7940000
10	BOW and Bag of N-Grams Combined Baseline	BOW_BONG	0.8721995	0.8580000	0.8480611	0.8580000
11	BOW + Bag of NGrams Top: 1350 Features with Naive Bayes	BOW_BONG	0.8687971	0.8580000	0.8548460	0.8580000
12	BOW With Top: 39 PCA Components Seleted	BOW_PCA	0.8093699	0.7920000	0.7945170	0.7920000

Best results were produced from the BOW Features with optimal Features selected using a Naive Bayes Multinomial Model

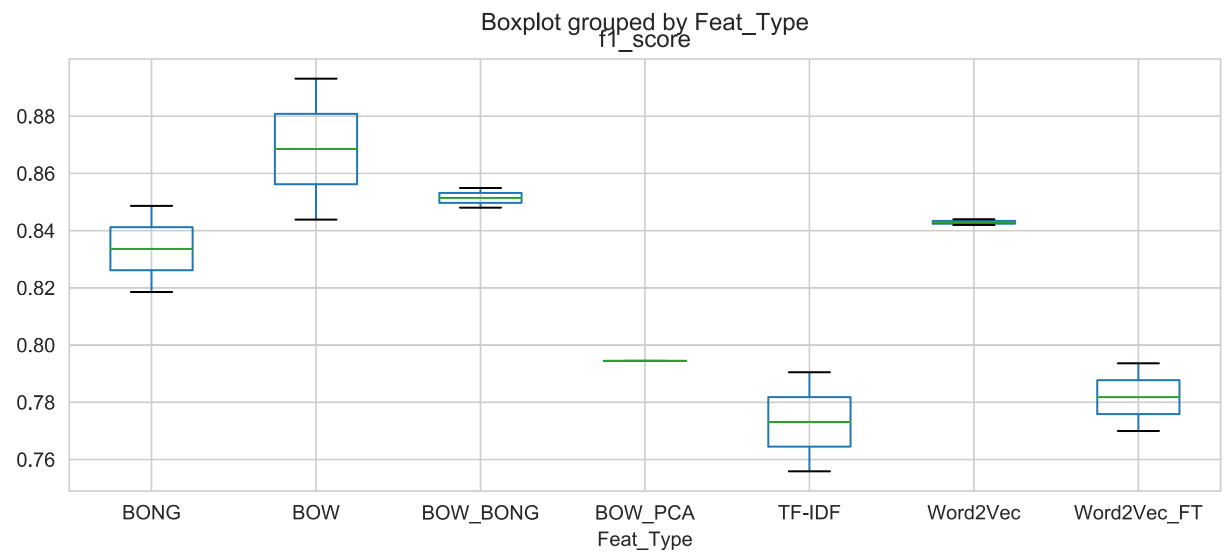
```
In [141]: df_benchmarks_all.sort_values(by='f1_score', ascending=False).plot(x="Features_Benchedmarked", y="f1_score", kind='bar', title
```

```
Out[141]: <matplotlib.axes._subplots.AxesSubplot at 0x26762954388>
```



```
In [142]: df_benchmarks_all.boxplot(column=['f1_score'], by='Feat_Type', figsize=(10, 4))
```

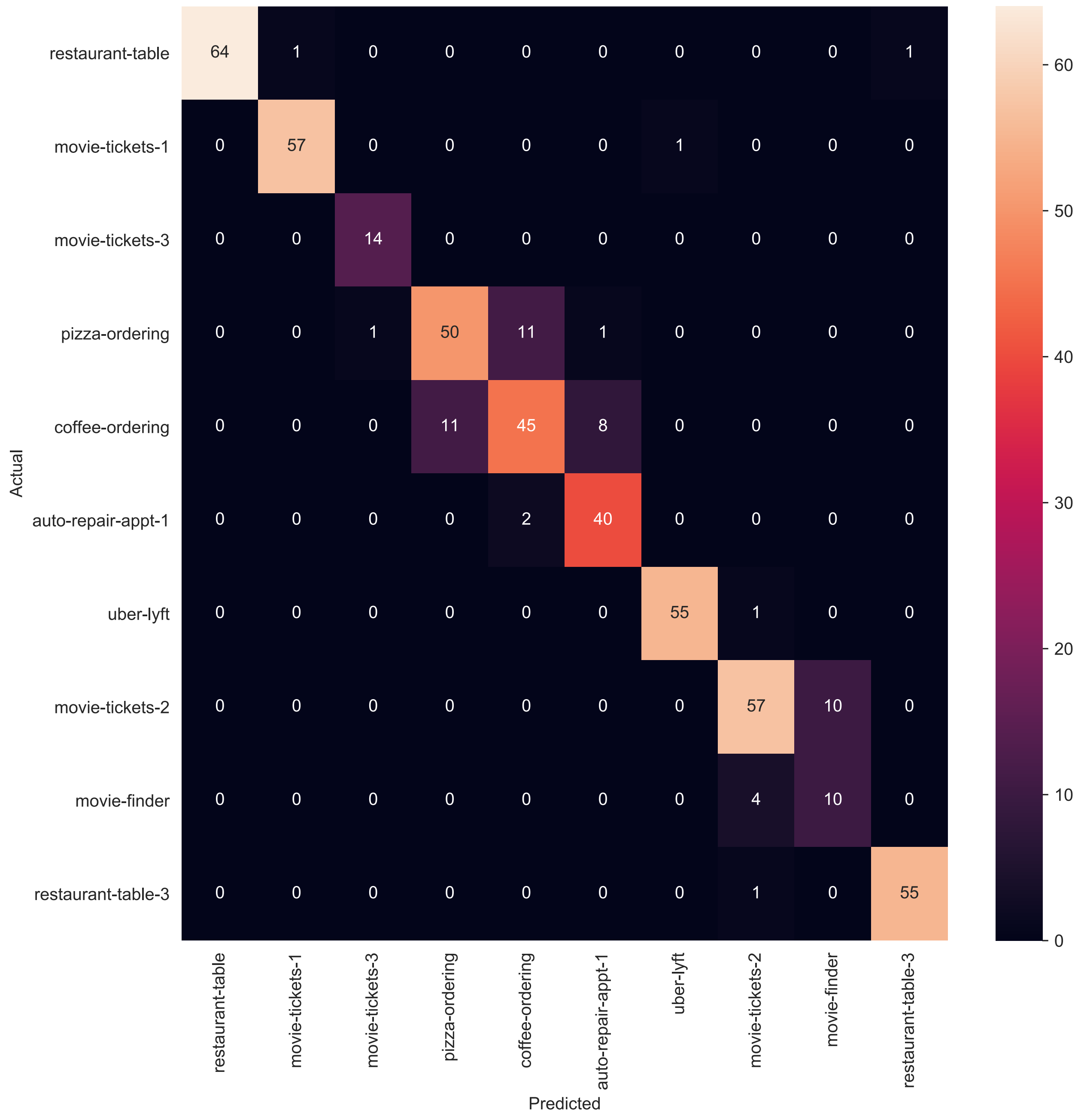
```
Out[142]: <matplotlib.axes._subplots.AxesSubplot at 0x2676297c8c8>
```



Confusion Matrix Heat Map of the Predictions fron the Best Resulting Features

This gives us a visual on where the model is failing

In [143]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(rm_chi_opt_bow.cm, annot=True, fmt='d',
            xticklabels=category_id_df.Instruction_id.values, yticklabels=category_id_df.Instruction_id.values)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [ ]:

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)

Dataset: Taskmaster-1 dataset from Google. Taskmaster-1 (https://research.google/tools/datasets/taskmaster-1/)

Dataset Source: https://github.com/google-research-datasets/Taskmaster (https://github.com/google-research-datasets/Taskmaster)

Notebook 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	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)
        df_all = df_all[df_all.selfdialog_norm != '']
        print(df_all.shape)
```

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

```
In [6]: print (df_all.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    1160
coffee-ordering       1376
movie-finder            54
movie-tickets-1        678
movie-tickets-2        377
movie-tickets-3        195
pizza-ordering        1467
restaurant-table      1198
restaurant-table-3     102
uber-lyft             1098
dtype: int64
```

```
In [7]: no_of_samples = 2000
        small_classes_count = 54 + 195 + 102 + 377
        smp_per_cls = (no_of_samples - small_classes_count)//6
        delta = no_of_samples - small_classes_count - (smp_per_cls * 6)
        print(smp_per_cls, delta)
```

```
212 0
```

In [8]:
```
#weight_higher = ['restaurant-table-2', 'movie-tickets-1', 'movie-tickets-3','uber-lift-2','coffee-ordering-1','coffee-orderin
# class_sample_size_dict = { #2000 Samples
#     "auto-repair-appt-1": 230,
#     "coffee-ordering": 230,
#     "movie-finder": 54,
#     "movie-tickets-1": 250,
#     "movie-tickets-2": 250,
#     "movie-tickets-3": 195,
#     "pizza-ordering": 230,
#     "restaurant-table": 230,
#     "restaurant-table-3": 101,
#     "uber-lyft": 230
# }
class_sample_size_dict = { # 3000 Samples
    "auto-repair-appt-1": smp_per_cls - 100,
    "coffee-ordering": smp_per_cls,
    "movie-finder": 54,
    "movie-tickets-1": smp_per_cls + delta + 100,
    "movie-tickets-2": 377,
    "movie-tickets-3": 195,
    "pizza-ordering": smp_per_cls,
    "restaurant-table": smp_per_cls,
    "restaurant-table-3": 102,
    "uber-lyft": smp_per_cls
}
sum(class_sample_size_dict.values())
```

Out[8]:
```
2000
```

Get a Sample of records.

In [9]:
```
# Function to Get balanced Sample - Get a bit more than needed then down sample
def sampling_k_elements(group):
    name = group['Instruction_id'].iloc[0]
    k = class_sample_size_dict[name]
    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)
print (corpus_df.groupby('Instruction_id').size(), corpus_df.shape)
```

```
Instruction_id
auto-repair-appt-1    112
coffee-ordering       212
movie-finder           54
movie-tickets-1       312
movie-tickets-2       377
movie-tickets-3       195
pizza-ordering        212
restaurant-table      212
restaurant-table-3    102
uber-lyft             212
dtype: int64 (2000, 3)
```

Generate Corpus List

```
In [10]: doc_lst = []
         for i, row in corpus_df.iterrows():
             doc_lst.append(row.selfdialog_norm)

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

```
2000
```

```
Out[10]: ['hi im issue car help sure whats problem light came saying headlight ok want get fixed right away today would ideal already
         know want take yes intelligent auto solutions ok let pull website online scheduler see today ok im looks like two appointmen
         ts open today could minutes im least minutes away ok time would pm tonight tell able fix spot call confirm makemodel car kia
         soul ok said parts done appointment thats great news please book yes booked online thanks give info yes text youll phone tha
         nk big help',
          'hi schedule appointment car okay auto repair shop would like check check intelligent auto solutions car bringing lexus im
         driving put name cell phone number yes put jeff green cell phone number seems problem car makes sound step brakes anything e
         lse would like check like oil change maintenance yes think im due oil change well got let check online see available check b
         ring mins able make appointment bring car time pm great thanks initial cost brake checkup oil change okay accept credit card
         yes great thanks bye youre welcome bye',
          'assistant favor yes course whats going car making weird rattly noises think checked find good mechanic certainly im checki
         ng google right moment ok appears auto shop near work star rating want give call yes please ok ill put hold moment see say g
         reat thanks ok im back said bring tomorrow ok long going keep depends whats going said could problem muffler wont know look
         gave number theyll give call alright make sure get uber tomorrow morning yes time well probably need leave house ok ill hous
         e get car ill make sure uber arrives well thank much youre welcome need anything else ok see tomorrow',
          'gail need help schedule appointment intelligent auto solutions car whats wrong car need schedule appointment look radiator
         see drops fluid time park ground ok year model car bmw series sure name use use name scolar timer address miklan road forest
         hills new mexico bring car tomorrow see get earlier situation annoying time bring work pm take abut minutes ok let check wou
         ld prefer bring tomorrow morning let check time slots way please reserve car use mean time case car kept overnight well chec
         ked time bring pm today ok let confirm everything bring car today pm check leaking radiator get car ise case car stays overn
         ight thats correct repair shop need initial inspection thats ok go right ahead book appointment sure everything booked reque
         sted thanks help talk later']
```

Split Data into Train and Test Sets

```
In [11]: 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 = 0)
```

Build Vocabulary

```
In [12]: 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: 7051
Vocabulary Sample: [('like', 1), ('would', 2), ('tickets', 3), ('pm', 4), ('ok', 5), ('okay', 6), ('yes', 7), ('want', 8),
('movie', 9), ('see', 10)]
```

Bag of Words Feature Extraction

```
In [13]: 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[13]: array([[0, 4, 4, ..., 0, 0, 0],
                [0, 5, 7, ..., 0, 0, 0],
                [0, 2, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 1, ..., 0, 1, 0],
                [0, 3, 3, ..., 0, 0, 0],
                [0, 7, 6, ..., 0, 0, 1]], dtype=int64)
```

```
In [14]: # get all unique words in the corpus
         vocab = cv.get_feature_names()
         # show document feature vectors
         pd.DataFrame(cv_matrix, columns=vocab)
```

Out[14]:

	PAD	like	would	tickets	pm	ok	okay	yes	want	movie	see	time	thank	order	please	...	bucca	peppo	invited	honored	savei	jambalaya
0	0	4	4	4	2	0	4	0	1	3	4	2	0	0	1	...	0	0	0	0	0	0
1	0	5	7	7	1	9	0	0	0	1	1	0	3	0	3	...	0	0	0	0	0	0
2	0	2	0	0	2	3	0	2	1	0	0	0	2	0	0	...	0	0	0	0	0	0
3	0	2	4	4	1	4	0	0	2	1	0	2	2	2	0	...	0	0	0	0	0	0
4	0	0	0	2	2	0	0	1	2	2	0	0	1	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	0	2	1	3	2	5	0	0	4	1	3	1	1	0	3	...	0	0	0	0	0	0
1496	0	9	9	2	9	1	1	1	1	4	3	0	0	0	4	...	0	0	0	0	0	0
1497	0	0	1	0	0	3	0	1	0	0	0	2	0	0	0	...	0	0	0	0	0	0
1498	0	3	3	3	1	2	3	0	0	0	2	0	0	0	0	...	0	0	0	0	0	0
1499	0	7	6	5	3	4	0	2	0	0	0	1	2	0	0	...	0	0	0	0	0	0

1500 rows × 7051 columns

```
In [15]: # 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)
```

```
(1500, 7051)
(500, 7051)
(500,)
```

Define Model Builder Function

In [16]:
```
#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 [17]:
```
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 [18]:
```

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_metrics.accuracy])
    rows_benchmarks_all.append([descr, feat_type, b_metrics.precision, b_metrics.recall, b_metrics.f1_score, b_metrics.accurac
    df_benchmarks = pd.DataFrame(rows_benchmarks, columns=["Features_Benchedmarked", "Feat_Type", "Precision", "Recall", "f1_s
    df_benchmarks_all = pd.DataFrame(rows_benchmarks_all, columns=["Features_Benchedmarked", "Feat_Type", "Precision", "Recall
```

In [19]:
```
# Save benchmark output
Save_Benchmark("BOW Naive Bayes Baseline", "BOW", rm_nb_bow, True, True)
#df_benchmarks
```

In [20]:
```
from sklearn.metrics import confusion_matrix
#rm_nb_bow.cm
```

In [21]:
```
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 [22]:
```
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, labels_test)
    x_train_scaled, x_test_scaled = Get_Scaled_Features(X_train_chi, labels_train, X_test_chi, labels_test, scaler)
    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, scaler):
    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_chi, precision

    return rm_chi
```

Iterate through number of features and get benchmark results

In [23]:
```
a = 100
tot = X_train_bow.shape[1]
b = 100 * (tot//100)
c = 100
print(a, b, c)
```
```
100 7000 100
```

In [24]:
```
import sys
```

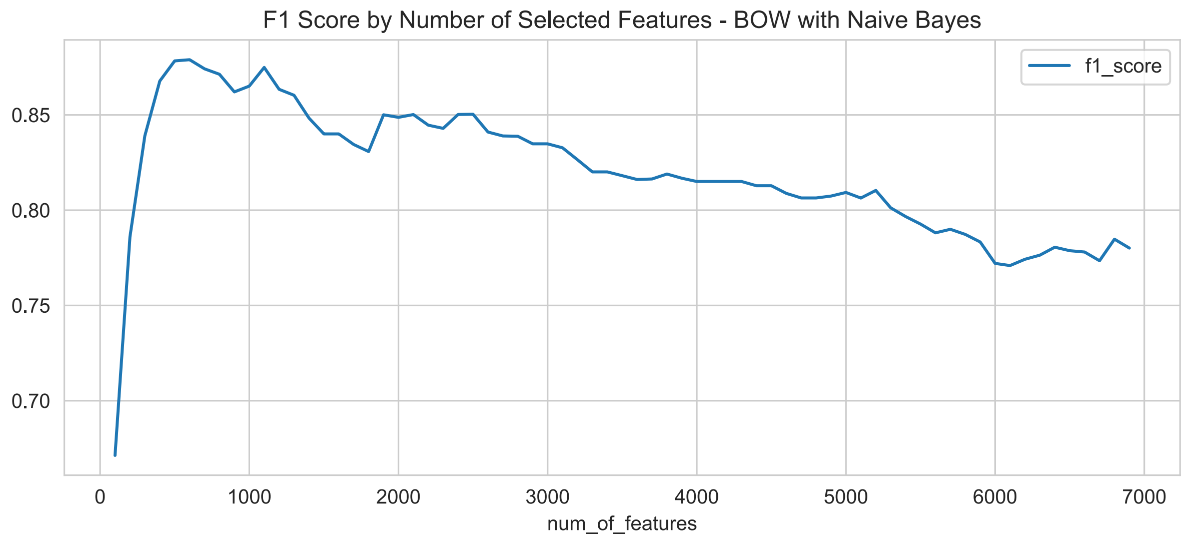
```
In [25]: rows = []

         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_max)
             rm_chi_i = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_max)
             rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])
             sys.stdout.write('\r'+str(i) + "/" + str(b))
             sys.stdout.flush()

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

```
6900/7000
```

Plot f1-score by number of selected features

```
In [26]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - BOW with Naive Bayes", figsize
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x20d56895888>
```



```
In [27]: 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)
```

```
550 650 1
```

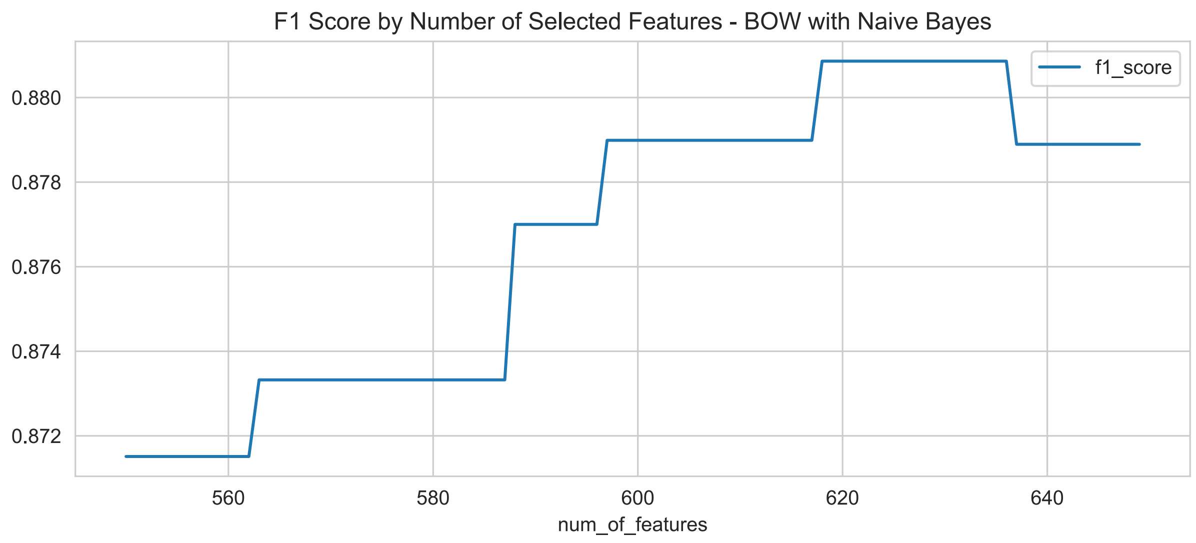
Get a more fine-grained look at the optimal number of features region

```
In [28]: 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 = SelectBestModelFeatures_Chi(model_nb_bow, i, X_train_bow, y_train, X_test_bow, y_test, scaler_min_max)
             rows.append([i, rm_chi_i.f1_score, rm_chi_i.accuracy])
             sys.stdout.write('\r'+str(i) + "/" + str(b))
             sys.stdout.flush()

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

```
649/650
```

```
In [29]: acc_df.plot(x="num_of_features", y="f1_score", title="F1 Score by Number of Selected Features - BOW with Naive Bayes", figsize
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x20d5695ce88>
```



```
In [30]: 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)
```

```
624
```

Benchmark BOW With Optimal Features Selected using Naive Bayes Model

```
In [31]: 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_test, scaler_mi
```

```
In [32]: #print(rm_chi_opt_bow.cm)
```

```
In [33]: #print("Label" + rm_chi_opt_bow.report)
```

```
In [34]: # Save benchmark output
         Save_Benchmark("BOW Naive Bayes Optimal Features Selected: " + str(Opt_no_of_feat), "BOW", rm_chi_opt_bow, False, False)
         df_benchmarks
```

Out[34]:

	Features_Benchedmarked	Feat_Type	Precision	Recall	f1_score	accuracy
0	BOW Naive Bayes Baseline	BOW	0.8302053	0.7980000	0.7771830	0.7980000
1	BOW Naive Bayes Optimal Features Selected: 624	BOW	0.8944239	0.8840000	0.8808603	0.8840000

1. Benchmark Comparison

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

```
In [35]: # Manage Results List
         def Result_Update_Or_Append(model_id, model_name, feat_status, hyper_param_status, best_params, f1_score, reset_entr):
             global entries
             if (reset_entr):
                 entries = {}
             entries[model_id+model_name+feat_status+hyper_param_status] = [model_id, model_name, feat_status, hyper_param_status, best

             result_list = list(entries.values())
             return result_list
```

Baseline Features

In [36]:
```
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_jobs=-1),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(n_jobs=-1),
]
CV = 10
cv_df = pd.DataFrame(index=range(CV * len(models)))
reset_entries = True
#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)

    results = Result_Update_Or_Append(model_id, model_name, 'baseline', 'default', '', f1_scores.mean(), reset_entries)
    print("Mean F1 Score: %.2f (+/- %.2f) [%s]" %(f1_scores.mean(), f1_scores.std(), model_name))

    # for i in range(0, 9, 1):
    #     Result_Update_Or_Append(model_id, model_name, 'baseline', 'default', '', f1_scores[i], reset_entries)
    reset_entries = False

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

```
Mean F1 Score: 0.81 (+/- 0.03) [RandomForestClassifier]
Mean F1 Score: 0.84 (+/- 0.02) [LinearSVC]
Mean F1 Score: 0.80 (+/- 0.02) [MultinomialNB]
Mean F1 Score: 0.86 (+/- 0.01) [LogisticRegression]
```

Optimised Features

In [37]:
```

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

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=CV)
    #recalls = cross_val_score(model, rm_chi_opt_bow.x_train_sel, y_train, scoring='recall_weighted', cv=CV)

    results = Result_Update_Or_Append(model_id, model_name, 'optimized', 'default', '', f1_scores.mean(), False)
    print("Mean F1 Score: %.2f (+/- %.2f) [%s]" %(f1_scores.mean(), f1_scores.std(), model_name))

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

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

```
Mean F1 Score: 0.83 (+/- 0.02) [RandomForestClassifier]
Mean F1 Score: 0.82 (+/- 0.02) [LinearSVC]
Mean F1 Score: 0.87 (+/- 0.02) [MultinomialNB]
Mean F1 Score: 0.85 (+/- 0.02) [LogisticRegression]
```

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.

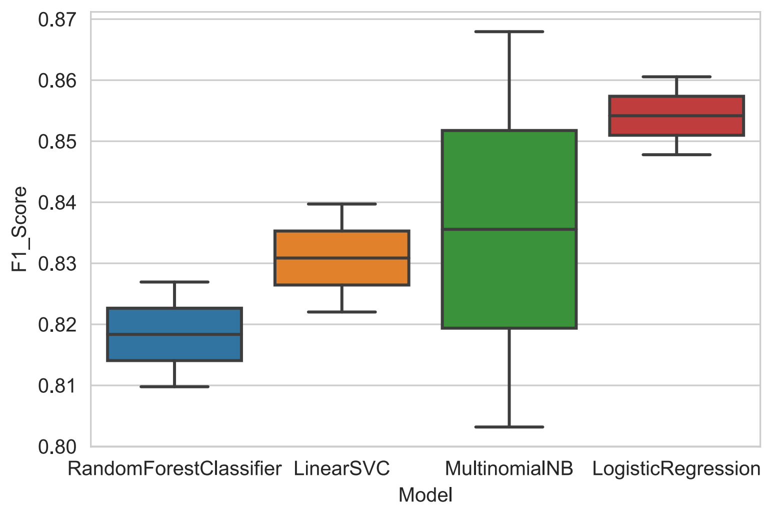
In [38]:
```
from IPython.display import display, HTML

# #models_df = cv_df.groupby(['Model_Id', 'Model','Features', 'Hyper_Param', 'Best_Params']).agg(['mean'])
# 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
#cv_df
#display(HTML(cv_df.to_html()))
display(cv_df)
```

	Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score
0	RF	RandomForestClassifier	baseline	default		0.8097789
1	SVC	LinearSVC	baseline	default		0.8397106
2	NB	MultinomialNB	baseline	default		0.8031807
3	LR	LogisticRegression	baseline	default		0.8605552
4	RF	RandomForestClassifier	optimized	default		0.8269332
5	SVC	LinearSVC	optimized	default		0.8220224
6	NB	MultinomialNB	optimized	default		0.8679468
7	LR	LogisticRegression	optimized	default		0.8477885

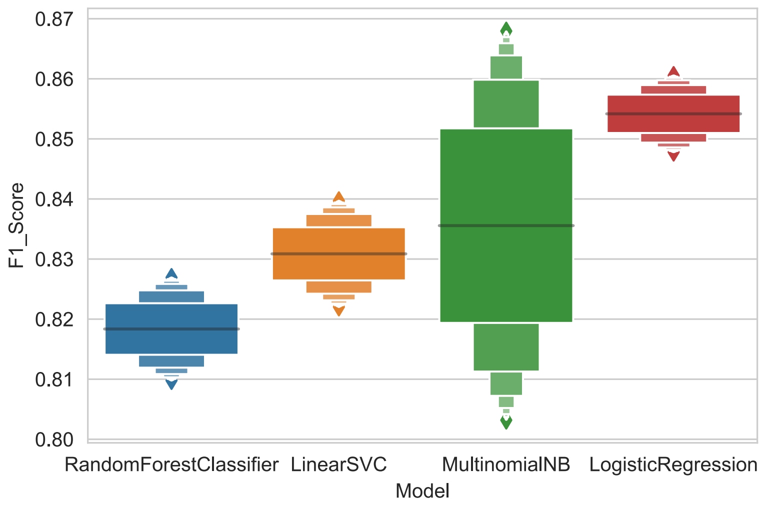
In [39]:
```
# import seaborn as sns

# fig, (ax1, ax2) = plt.subplots(figsize=(12, 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", linewidth=2, ax=a
# sns.barplot(y='F1_Score', x='Model', data=cv_df, palette="colorblind", hue='Features', ax=ax2);
```

In [40]:
```
import seaborn as sns

sns.boxplot(x='Model', y='F1_Score', data=cv_df)
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20d56bc6348>
```



```
In [41]: sns.lvplot(x='Model', y='F1_Score', data=cv_df)

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x20d58225308>
```



Optimize the Hyperparameters Using Grid Search

```
In [42]: 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 [43]: from sklearn.model_selection import GridSearchCV

est_param_arr = [
    Estimator_Parameters(RandomForestClassifier(), [{'n_estimators': [90,100,110],'max_depth': [4,5,6], 'random_state': [0,1,2
    Estimator_Parameters(LinearSVC(), [{'C': [1200, 1300, 1400, 1500],'loss': ['hinge', 'squared_hinge'], 'dual': [True, False
    Estimator_Parameters(MultinomialNB(), [{'alpha': [0.3,0.4,0.42,0.44,0.46],'fit_prior': [True, False]}], "optimized", rm_ch
    Estimator_Parameters(LogisticRegression(), [{'C': [0.1, 0.5,1,2,3], 'penalty': ['l1', 'l2', 'elasticnet', 'none'],'dual':
]

grid_dict = {}

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)
    results = Result_Update_Or_Append(model_id, estimator_name, est_param.feat_type, 'tuned', str(best_parameters), best_accur
    print(estimator_name, best_accuracy, best_parameters, est_param.feat_type)
    grid_dict[estimator_name] = best_parameters
cv_df = pd.DataFrame(results, columns=['Model_Id','Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Score'])
```

```
RandomForestClassifier 0.6289336608282621 {'max_depth': 6, 'n_estimators': 90, 'random_state': 0} optimized
LinearSVC 0.7807117463583535 {'C': 1500, 'dual': True, 'loss': 'hinge', 'max_iter': 900, 'penalty': 'l2'} optimized
MultinomialNB 0.8794765148993544 {'alpha': 0.44, 'fit_prior': False} optimized
LogisticRegression 0.8544466743737245 {'C': 0.1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'} optimized
```

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

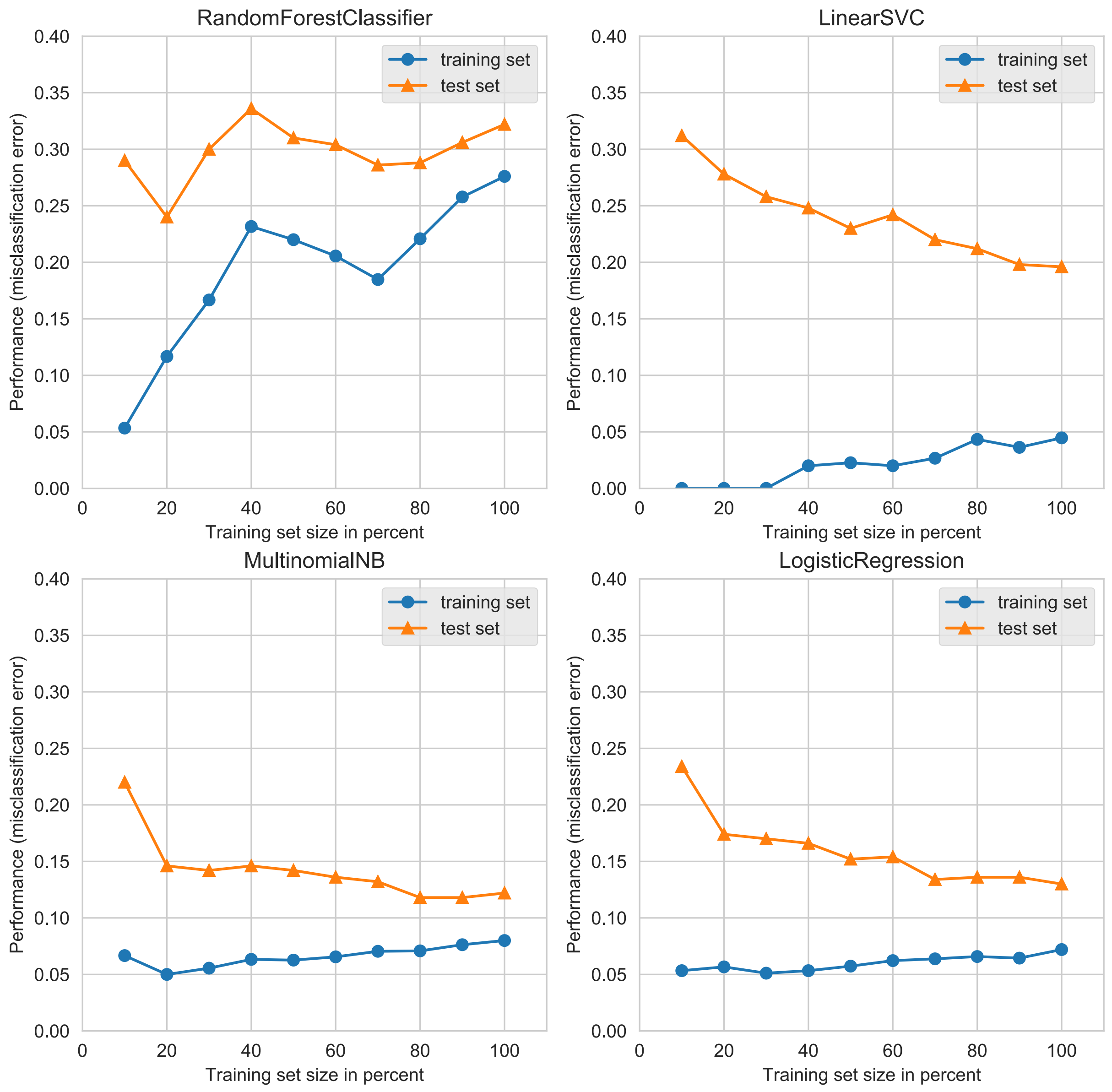
In [44]:
```
cv_df
```

Out[44]:

	Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score
0	RF	RandomForestClassifier	baseline	default		0.8097789
1	SVC	LinearSVC	baseline	default		0.8397106
2	NB	MultinomialNB	baseline	default		0.8031807
3	LR	LogisticRegression	baseline	default		0.8605552
4	RF	RandomForestClassifier	optimized	default		0.8269332
5	SVC	LinearSVC	optimized	default		0.8220224
6	NB	MultinomialNB	optimized	default		0.8679468
7	LR	LogisticRegression	optimized	default		0.8477885
8	RF	RandomForestClassifier	optimized	tuned	{'max_depth': 6, 'n_estimators': 90, 'random_state': 0}	0.6289337
9	SVC	LinearSVC	optimized	tuned	{'C': 1500, 'dual': True, 'loss': 'hinge', 'max_iter': 900, 'penalty': 'l2'}	0.7807117
10	NB	MultinomialNB	optimized	tuned	{'alpha': 0.44, 'fit_prior': False}	0.8794765
11	LR	LogisticRegression	optimized	tuned	{'C': 0.1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.8544467

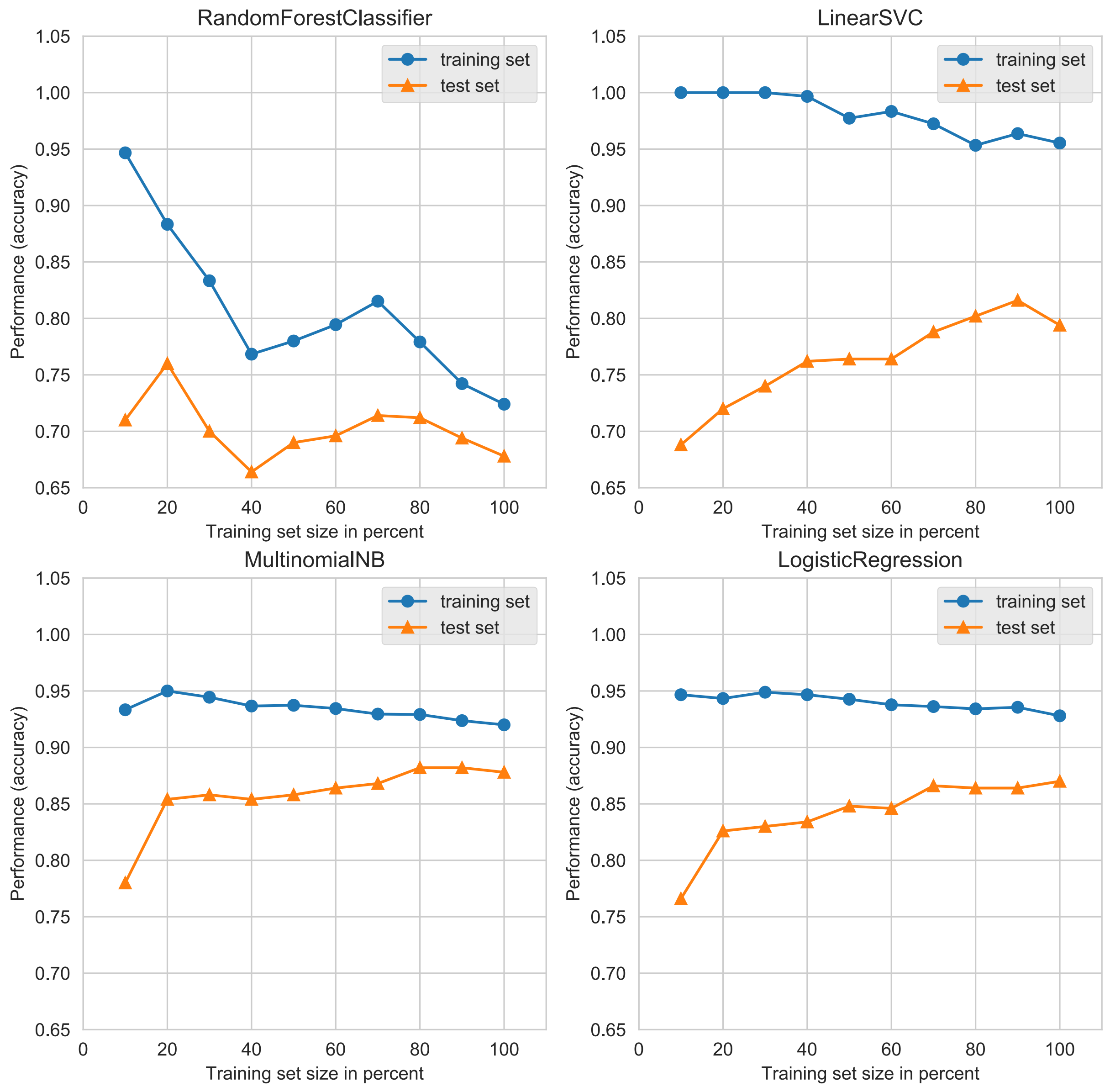
2. a. Learning Curves: Training/ Testing Errors - Optimized Hyperarameters

In [45]:
```python
from mlxtend.plotting import plot_learning_curves
import itertools
import matplotlib.gridspec as gridspec

models = [
    RandomForestClassifier(**grid_dict['RandomForestClassifier']),
    LinearSVC(**grid_dict['LinearSVC']),
    MultinomialNB(**grid_dict['MultinomialNB']),
    LogisticRegression(**grid_dict['LogisticRegression']),
]

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, print_model=Fal
    plt.ylim(0.00, 0.40)
    plt.title(model_name)

plt.show()
```



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

```
In [46]: from mlxtend.plotting import plot_learning_curves
         import matplotlib.gridspec as gridspec
         import itertools

         models = [
             RandomForestClassifier(**grid_dict['RandomForestClassifier']),
             LinearSVC(**grid_dict['LinearSVC']),
             MultinomialNB(**grid_dict['MultinomialNB']),
             LogisticRegression(**grid_dict['LogisticRegression']),
         ]

         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]])
             fig3 = plot_learning_curves(rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test, model, scoring='accura
             plt.ylim(0.65, 1.05)
             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.

Initialize Models with optimized hyperparameters

```
In [47]: clf1 = RandomForestClassifier(**grid_dict['RandomForestClassifier'])
         clf2 = LinearSVC(**grid_dict['LinearSVC'])
         clf3 = MultinomialNB(**grid_dict['MultinomialNB'])
         clf4 = LogisticRegression(**grid_dict['LogisticRegression'])
         clf_list = [clf1, clf2, clf3, clf4]
```

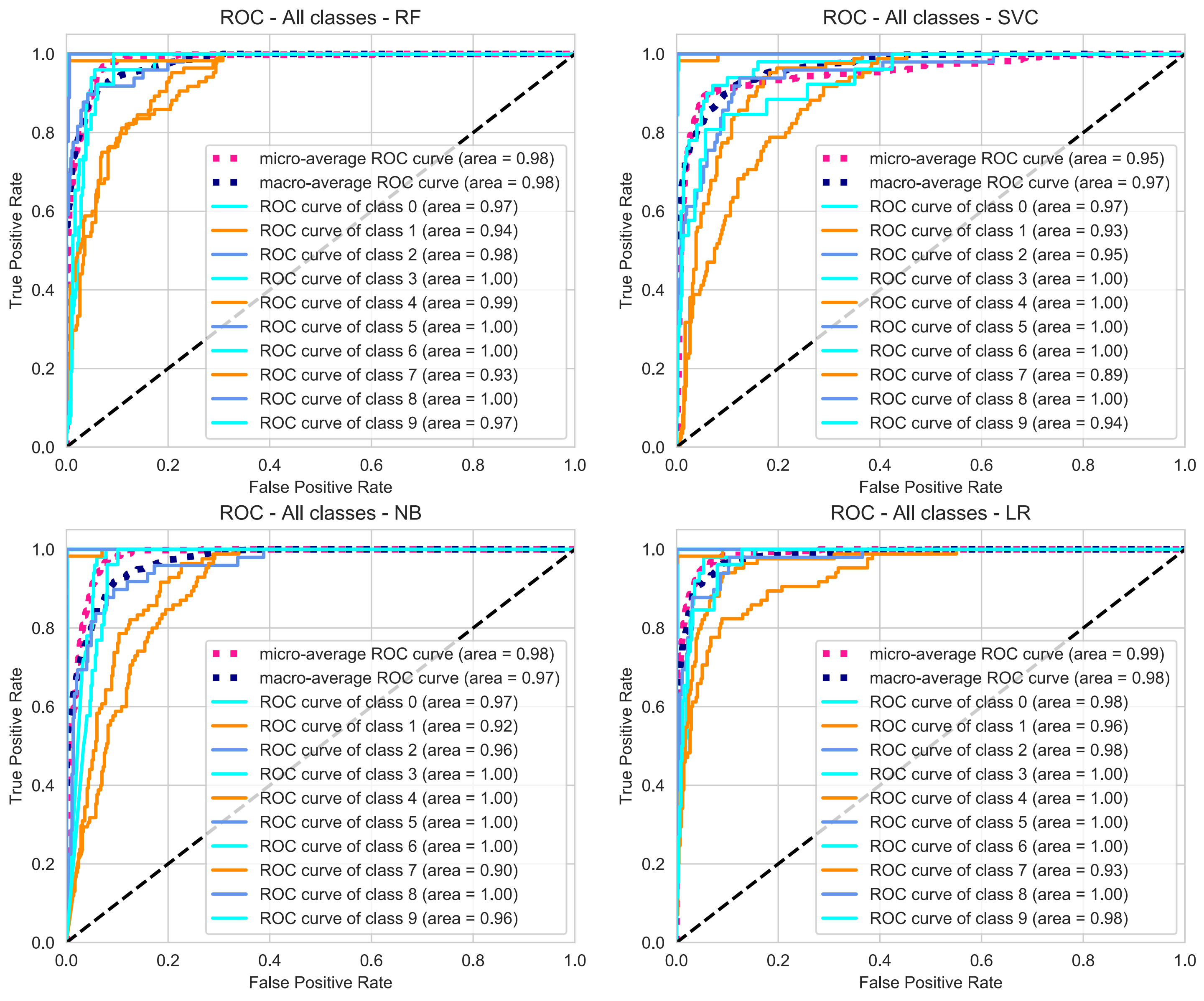
ROC/ AUC

```
In [48]: from sklearn import svm
         from sklearn.metrics import roc_curve, auc
         from sklearn.preprocessing import label_binarize
         from sklearn.multiclass import OneVsRestClassifier
         from scipy import interp
         from sklearn.metrics import roc_auc_score
         from itertools import cycle

         # # Binarize the output
         y_tr = label_binarize(y_train, classes=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
         n_classes = y_tr.shape[1]
         y_te = label_binarize(y_test, classes=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
         n_classes_te = y_te.shape[1]

         random_state = np.random.RandomState(0)

         y_score_dict = dict()
         labels = ['RF', 'SVC', 'NB','LR']
         roc_dict = dict()

         for clf, label in zip(clf_list, labels):
             classifier = OneVsRestClassifier(clf)
             if label == 'SVC':
                 y_score = classifier.fit(rm_chi_opt_bow.x_train_sel, y_tr).decision_function(rm_chi_opt_bow.x_test_sel)
                 y_score_dict[label] = y_score
             else:
                 y_score = classifier.fit(rm_chi_opt_bow.x_train_sel, y_tr).predict_proba(rm_chi_opt_bow.x_test_sel)
                 y_score_dict[label] = y_score
```

```
In [49]: for label in labels:
             # Compute ROC curve and ROC area for each class
             fpr = dict()
             tpr = dict()
             roc_auc = dict()
             for i in range(n_classes):
                 #print(i)
                 fpr[i], tpr[i], _ = roc_curve(y_te[:, i], y_score_dict[label][:, i])
                 roc_auc[i] = auc(fpr[i], tpr[i])

             # Compute micro-average ROC curve and ROC area
             fpr["micro"], tpr["micro"], _ = roc_curve(y_te.ravel(), y_score_dict[label].ravel())
             roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

             # First aggregate all false positive rates
             all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

             # Then interpolate all ROC curves at these points
             mean_tpr = np.zeros_like(all_fpr)
             for i in range(n_classes):
                 mean_tpr += interp(all_fpr, fpr[i], tpr[i])

             # Finally average it and compute AUC
             mean_tpr /= n_classes

             fpr["macro"] = all_fpr
             tpr["macro"] = mean_tpr
             roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

             roc_dict[label] = [fpr, tpr, roc_auc, all_fpr, mean_tpr]
```

Visuaize ROC Curves

In [50]:
```python
from itertools import cycle

lw = 2

fig2 = plt.figure(figsize=(12, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for label, grd in zip(labels, grid):
      ax = plt.subplot(gs[grd[0], grd[1]])

      # Plot ROC curves for all Classes
      #plt.figure()
      plt.plot(roc_dict[label][0]["micro"], roc_dict[label][1]["micro"],
            label='micro-average ROC curve (area = {0:0.2f})' ''.format(roc_dict[label][2]["micro"]), color='deeppink', linest
      plt.plot(roc_dict[label][0]["macro"], roc_dict[label][1]["macro"],
            label='macro-average ROC curve (area = {0:0.2f})' ''.format(roc_dict[label][2]["macro"]), color='navy', linestyle=

      colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
      for i, color in zip(range(n_classes), colors):
            plt.plot(roc_dict[label][0][i], roc_dict[label][1][i], color=color, lw=lw,
                  label='ROC curve of class {0} (area = {1:0.2f})' ''.format(i, roc_dict[label][2][i]))

      plt.plot([0, 1], [0, 1], 'k--', lw=lw)
      plt.xlim([0.0, 1.0])
      plt.ylim([0.0, 1.05])
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('ROC - All classes - ' + label)
      plt.legend(loc="lower right")
      #plt.legend(loc='lower left', bbox_to_anchor=(1, 0.5))
```



Compute Weighted AUC Scores

In [51]:
```
rows = []
labels = ['RF', 'SVC', 'NB','LR']
for label in  labels:
      macro_roc_auc_ovo = roc_auc_score(y_te, y_score_dict[label], multi_class="ovo", average="macro")
      weighted_roc_auc_ovo = roc_auc_score(y_te, y_score_dict[label], multi_class="ovo", average="weighted")
      macro_roc_auc_ovr = roc_auc_score(y_te, y_score_dict[label], multi_class="ovr", average="macro")
      weighted_roc_auc_ovr = roc_auc_score(y_te, y_score_dict[label], multi_class="ovr", average="weighted")

      rows.append([label, macro_roc_auc_ovo, weighted_roc_auc_ovo, macro_roc_auc_ovr, weighted_roc_auc_ovr])

print('Agregated ROC AUC scores:')
cv_df = pd.DataFrame(rows, columns=['Model_Id','One-vs-One Macro', 'One-vs-One Weighted', 'One-vs-Rest Macro', 'One-vs-Rest We
cv_df
```

```
Agregated ROC AUC scores:
```

Out[51]:

	Model_Id	One-vs-One Macro	One-vs-One Weighted	One-vs-Rest Macro	One-vs-Rest Weighted
0	RF	0.9786291	0.9707965	0.9786291	0.9707965
1	SVC	0.9671918	0.9579182	0.9671918	0.9579182
2	NB	0.9708260	0.9602845	0.9708260	0.9602845
3	LR	0.9829638	0.9770185	0.9829638	0.9770185

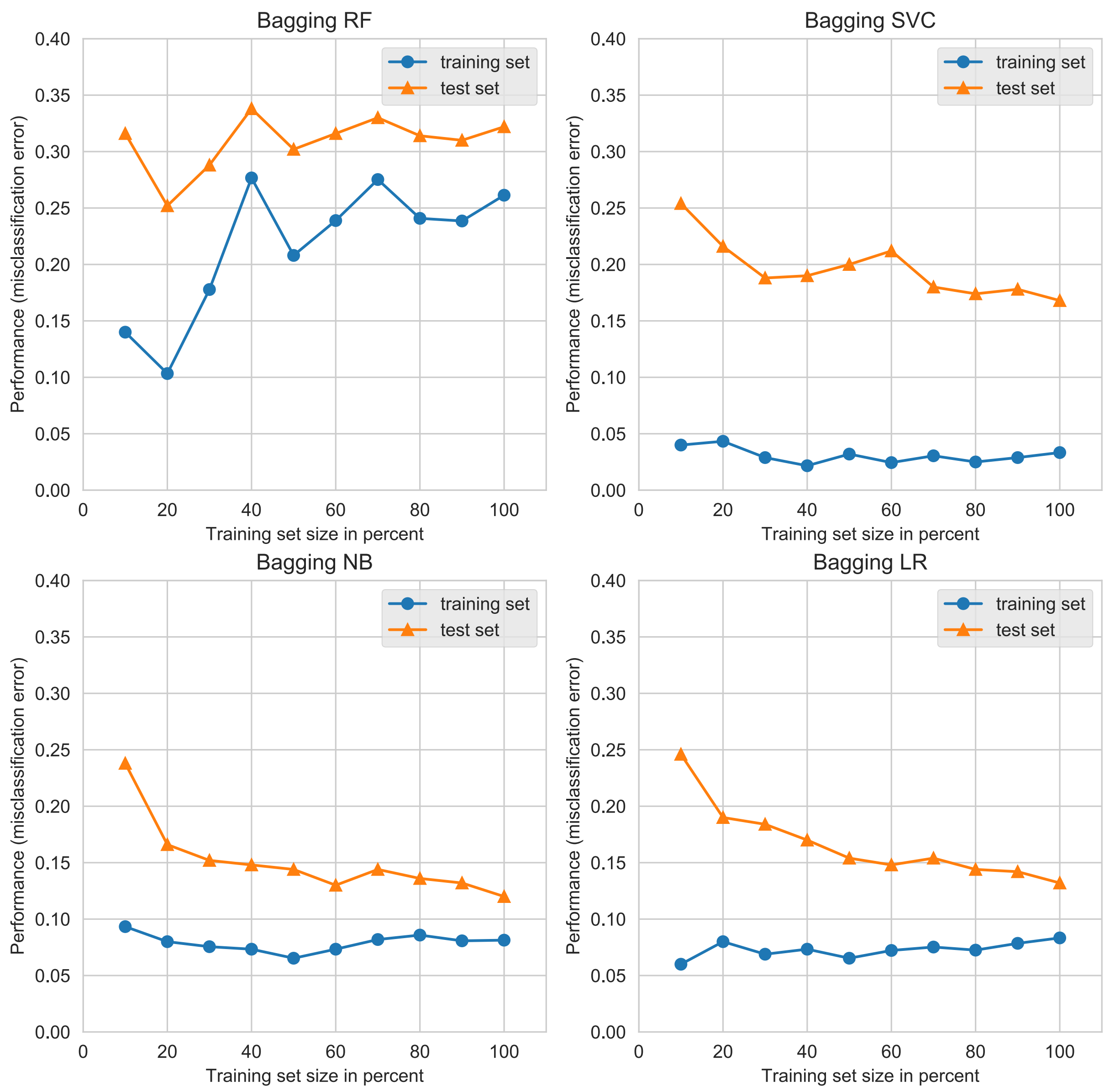
3. Ensemble Learning

Bagging

In [52]:
```
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 [53]: 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, print_model=Fal
    plt.ylim(0.00, 0.40)
    plt.title(label)

plt.show()
```



Bagging Scores Varied by Ensemble Size

In [54]:
```
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)

scores_mean_dict = {}
scores_std_dict = {}
scores_dict = {}

for clf, label, grd in zip(clf_list, labels, grid):
    num_est = map(int, np.linspace(5,50,6))
    bg_clf_cv_mean = []
    bg_clf_cv_std = []
    row_results = []
    for n_est in num_est:
        bg_clf = BaggingClassifier(base_estimator=clf, 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())
        row_results.append([label, scores.mean(), scores.std(), n_est])

    scores_mean_dict[label] = bg_clf_cv_mean
    scores_std_dict[label] = bg_clf_cv_std
    scores_dict[label] = row_results

    num_est = list(map(int, np.linspace(5,50,6)))
    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 Size');
plt.show()
```

Bagging RF Score by Ensemble Size

Bagging SVC Score by Ensemble Size

Get best Ensemble Size for each Model

In [55]:
```
best_no_of_est_dict = {}
labels = ['Bagging RF', 'Bagging SVC', 'Bagging NB','Bagging LR']
for label in labels:
    scores_df = pd.DataFrame(scores_dict[label], columns=["name", "f1_mean", "f1-std", "num_est"])
    Opt_no_of_est = int(scores_df.sort_values(by='f1_mean', ascending=False).iloc[0]['num_est'])
    best_no_of_est_dict[label] = Opt_no_of_est
best_no_of_est_dict
```

Out[55]:
```
{'Bagging RF': 5, 'Bagging SVC': 50, 'Bagging NB': 41, 'Bagging LR': 23}
```

In [56]:
```
bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=best_no_of_est_dict['Bagging RF'], max_samples=0.9)
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=best_no_of_est_dict['Bagging SVC'], max_samples=0.8)
bagging3 = BaggingClassifier(base_estimator=clf3, n_estimators=best_no_of_est_dict['Bagging NB'], max_samples=0.8)
bagging4 = BaggingClassifier(base_estimator=clf4, n_estimators=best_no_of_est_dict['Bagging LR'], max_samples=0.8)
```

In [57]:
```
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]

for clf, label, model_id in zip(clf_list, labels, model_ids):
    scores = cross_val_score(clf, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
    results = Result_Update_Or_Append(model_id, label, 'optimized', 'tuned', '', scores.mean(), False)
    print("F1-Score Weighted: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))

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

```
F1-Score Weighted: 0.61 (+/- 0.03) [Bagging RF]
F1-Score Weighted: 0.80 (+/- 0.01) [Bagging SVC]
F1-Score Weighted: 0.87 (+/- 0.01) [Bagging NB]
F1-Score Weighted: 0.84 (+/- 0.01) [Bagging LR]
```

In [58]:
```
cv_df
```

Out[58]:

	Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score
0	RF	RandomForestClassifier	baseline	default		0.8097789
1	SVC	LinearSVC	baseline	default		0.8397106
2	NB	MultinomialNB	baseline	default		0.8031807
3	LR	LogisticRegression	baseline	default		0.8605552
4	RF	RandomForestClassifier	optimized	default		0.8269332
5	SVC	LinearSVC	optimized	default		0.8220224
6	NB	MultinomialNB	optimized	default		0.8679468
7	LR	LogisticRegression	optimized	default		0.8477885
8	RF	RandomForestClassifier	optimized	tuned	{'max_depth': 6, 'n_estimators': 90, 'random_state': 0}	0.6289337
9	SVC	LinearSVC	optimized	tuned	{'C': 1500, 'dual': True, 'loss': 'hinge', 'max_iter': 900, 'penalty': 'l2'}	0.7807117
10	NB	MultinomialNB	optimized	tuned	{'alpha': 0.44, 'fit_prior': False}	0.8794765
11	LR	LogisticRegression	optimized	tuned	{'C': 0.1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.8544467
12	RF	Bagging RF	optimized	tuned		0.6077235
13	SVC	Bagging SVC	optimized	tuned		0.8038190
14	NB	Bagging NB	optimized	tuned		0.8670945
15	LR	Bagging LR	optimized	tuned		0.8357849

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

Boosting

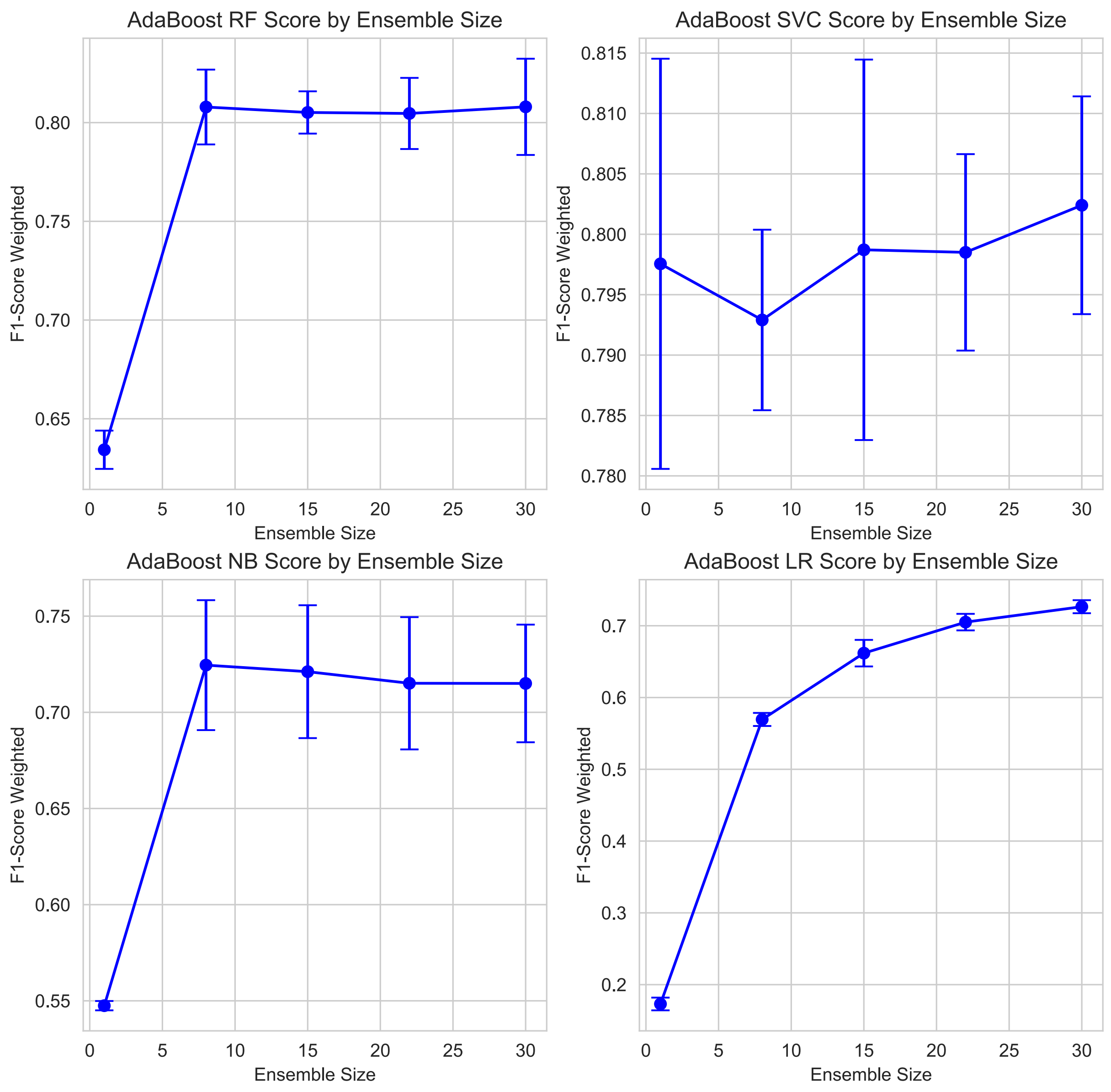
In [59]:
```
from sklearn.ensemble import AdaBoostClassifier

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 [60]:
```
from sklearn.ensemble import AdaBoostClassifier

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

scores_mean_dict = {}
scores_std_dict = {}
scores_dict = {}

num_est = map(int, np.linspace(5,30,6))

for boosting, label in zip(bst_list, labels):
    num_est = map(int, np.linspace(1,30,5))
    bg_clf_cv_mean = []
    bg_clf_cv_std = []
    row_results = []
    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())
        row_results.append([label, scores.mean(), scores.std(), n_est])
        print("F1-Score Weighted: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label + ', n_estimators: ' + str(n_est)
    scores_mean_dict[label] = bg_clf_cv_mean
    scores_std_dict[label] = bg_clf_cv_std
    scores_dict[label] = row_results
```

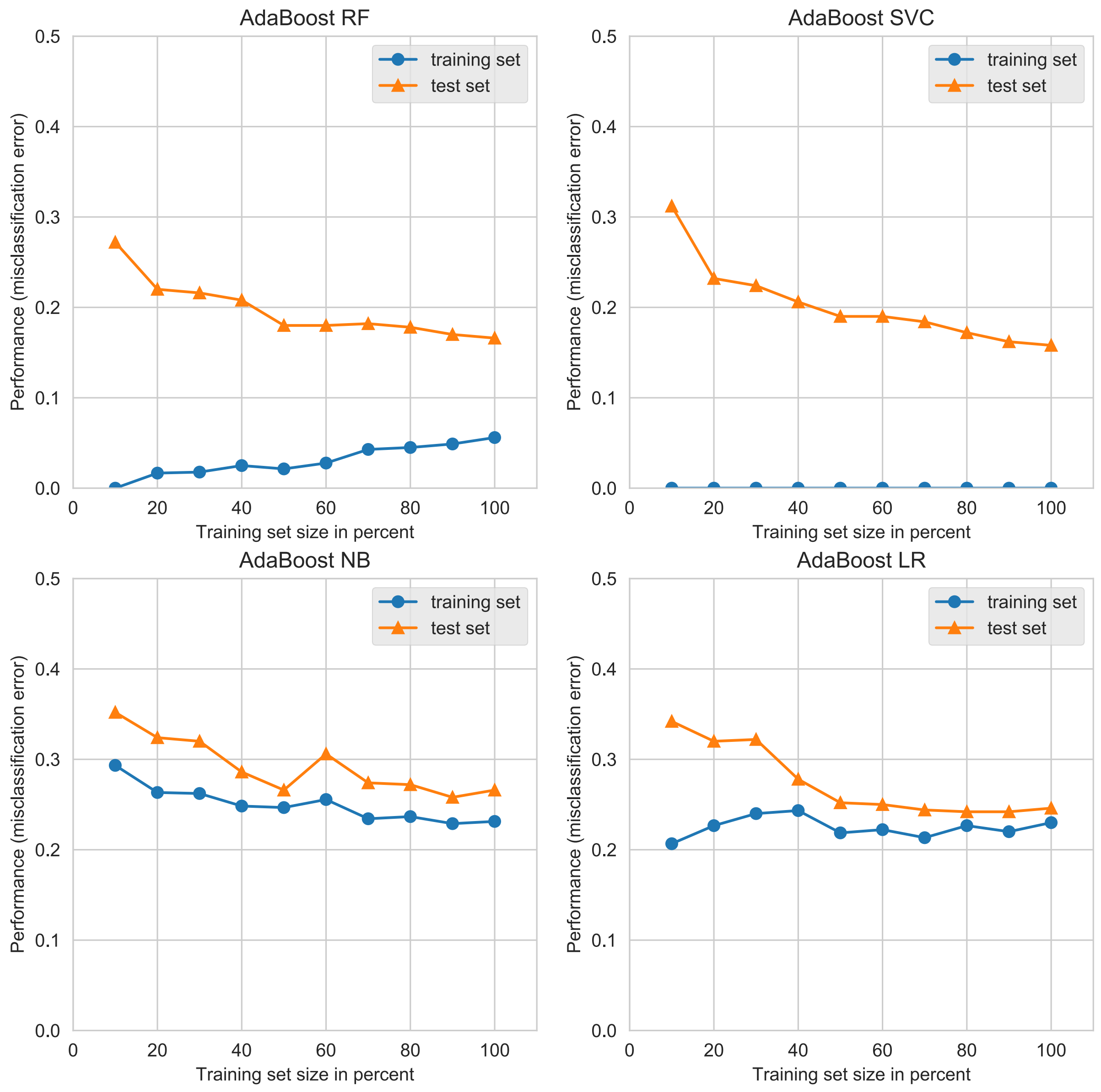
```
F1-Score Weighted: 0.63 (+/- 0.01) [AdaBoost RF, n_estimators: 1]
F1-Score Weighted: 0.81 (+/- 0.02) [AdaBoost RF, n_estimators: 8]
F1-Score Weighted: 0.81 (+/- 0.01) [AdaBoost RF, n_estimators: 15]
F1-Score Weighted: 0.80 (+/- 0.02) [AdaBoost RF, n_estimators: 22]
F1-Score Weighted: 0.81 (+/- 0.02) [AdaBoost RF, n_estimators: 30]
F1-Score Weighted: 0.80 (+/- 0.02) [AdaBoost SVC, n_estimators: 1]
F1-Score Weighted: 0.79 (+/- 0.01) [AdaBoost SVC, n_estimators: 8]
F1-Score Weighted: 0.80 (+/- 0.02) [AdaBoost SVC, n_estimators: 15]
F1-Score Weighted: 0.80 (+/- 0.01) [AdaBoost SVC, n_estimators: 22]
F1-Score Weighted: 0.80 (+/- 0.01) [AdaBoost SVC, n_estimators: 30]
F1-Score Weighted: 0.55 (+/- 0.00) [AdaBoost NB, n_estimators: 1]
F1-Score Weighted: 0.72 (+/- 0.03) [AdaBoost NB, n_estimators: 8]
F1-Score Weighted: 0.72 (+/- 0.03) [AdaBoost NB, n_estimators: 15]
F1-Score Weighted: 0.72 (+/- 0.03) [AdaBoost NB, n_estimators: 22]
F1-Score Weighted: 0.72 (+/- 0.03) [AdaBoost NB, n_estimators: 30]
F1-Score Weighted: 0.17 (+/- 0.01) [AdaBoost LR, n_estimators: 1]
F1-Score Weighted: 0.57 (+/- 0.01) [AdaBoost LR, n_estimators: 8]
F1-Score Weighted: 0.66 (+/- 0.02) [AdaBoost LR, n_estimators: 15]
F1-Score Weighted: 0.71 (+/- 0.01) [AdaBoost LR, n_estimators: 22]
F1-Score Weighted: 0.73 (+/- 0.01) [AdaBoost LR, n_estimators: 30]
```

In [61]:
```
fig2 = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for label, grd in zip(labels, grid):
    ax = plt.subplot(gs[grd[0], grd[1]])
    num_est = list(map(int, np.linspace(1,30,5)))
    (_, caps, _) = plt.errorbar(num_est, scores_mean_dict[label], yerr=scores_std_dict[label], 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 Size');
plt.show()
```



Learning Curves for Boosted Models

```
In [62]: #plot Boosting learning curve
         labels = ['AdaBoost RF', 'AdaBoost SVC', 'AdaBoost NB','AdaBoost LR']
         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_moc
             plt.ylim(0.00, 0.50)
             plt.title(label)
         plt.show()
```



Get best Ensemble Size for each Model

```
In [63]: best_no_of_est_dict = {}
         for label in labels:
             scores_df = pd.DataFrame(scores_dict[label], columns=["name", "f1_mean", "f1-std", "num_est"])
             Opt_no_of_est = int(scores_df.sort_values(by='f1_mean', ascending=False).iloc[0]['num_est'])
             best_no_of_est_dict[label] = Opt_no_of_est
         best_no_of_est_dict

Out[63]: {'AdaBoost RF': 30, 'AdaBoost SVC': 30, 'AdaBoost NB': 8, 'AdaBoost LR': 30}
```

```
In [64]: boosting1 = AdaBoostClassifier(base_estimator=clf1, n_estimators=best_no_of_est_dict['AdaBoost RF'])
         boosting2 = AdaBoostClassifier(base_estimator=clf2, n_estimators=best_no_of_est_dict['AdaBoost SVC'], algorithm='SAMME')
         boosting3 = AdaBoostClassifier(base_estimator=clf3, n_estimators=best_no_of_est_dict['AdaBoost NB'])
         boosting4 = AdaBoostClassifier(base_estimator=clf4, n_estimators=best_no_of_est_dict['AdaBoost LR'])
         boost_list = [boosting1, boosting2, boosting3, boosting4]
         labels_bst = ['AdaBoost RF', 'AdaBoost SVC', 'AdaBoost NB','AdaBoost LR']
```

```
In [65]: 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')
             results = Result_Update_Or_Append(model_id, label, 'optimized', 'tuned', '', scores.mean(), False)
             print("F1-Score Weighted: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))

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

```
F1-Score Weighted: 0.80 (+/- 0.01) [AdaBoost RF]
F1-Score Weighted: 0.81 (+/- 0.00) [AdaBoost SVC]
F1-Score Weighted: 0.72 (+/- 0.03) [AdaBoost NB]
F1-Score Weighted: 0.73 (+/- 0.01) [AdaBoost LR]
```

```
In [66]: cv_df
```

Out[66]:

	Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score
0	RF	RandomForestClassifier	baseline	default		0.8097789
1	SVC	LinearSVC	baseline	default		0.8397106
2	NB	MultinomialNB	baseline	default		0.8031807
3	LR	LogisticRegression	baseline	default		0.8605552
4	RF	RandomForestClassifier	optimized	default		0.8269332
5	SVC	LinearSVC	optimized	default		0.8220224
6	NB	MultinomialNB	optimized	default		0.8679468
7	LR	LogisticRegression	optimized	default		0.8477885
8	RF	RandomForestClassifier	optimized	tuned	{'max_depth': 6, 'n_estimators': 90, 'random_state': 0}	0.6289337
9	SVC	LinearSVC	optimized	tuned	{'C': 1500, 'dual': True, 'loss': 'hinge', 'max_iter': 900, 'penalty': 'l2'}	0.7807117
10	NB	MultinomialNB	optimized	tuned	{'alpha': 0.44, 'fit_prior': False}	0.8794765
11	LR	LogisticRegression	optimized	tuned	{'C': 0.1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.8544467
12	RF	Bagging RF	optimized	tuned		0.6077235
13	SVC	Bagging SVC	optimized	tuned		0.8038190
14	NB	Bagging NB	optimized	tuned		0.8670945
15	LR	Bagging LR	optimized	tuned		0.8357849
16	RF	AdaBoost RF	optimized	tuned		0.7987084
17	SVC	AdaBoost SVC	optimized	tuned		0.8070579
18	NB	AdaBoost NB	optimized	tuned		0.7245388
19	LR	AdaBoost LR	optimized	tuned		0.7265426

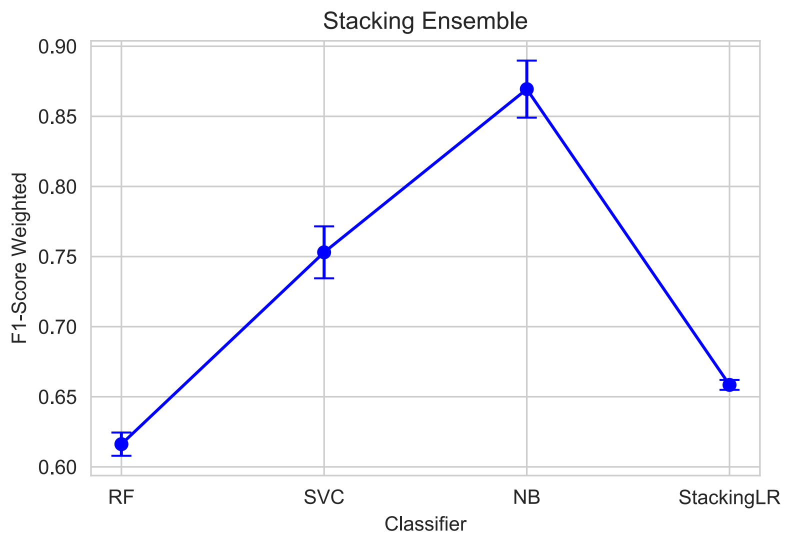
The Boosting ensemble did not provide any improvements on the baseline and optimized modeling.

Stacking

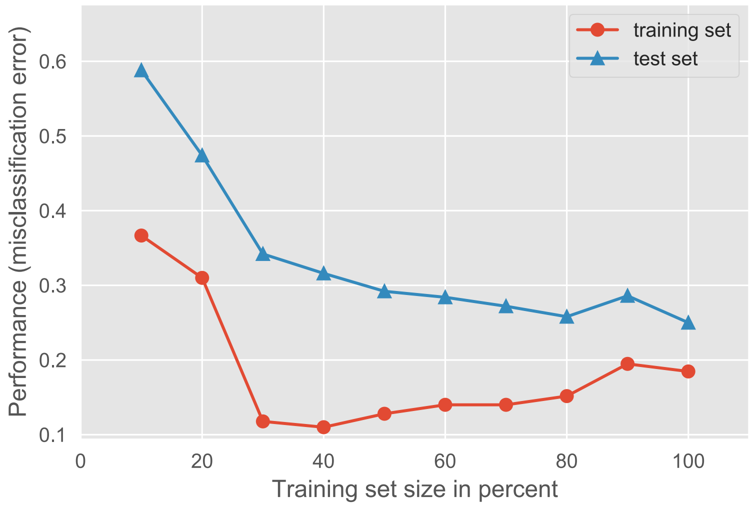
In [67]:
```python
from mlxtend.classifier import StackingClassifier

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='f1_weighted')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    if (label == 'Stacking LR'):
        results = Result_Update_Or_Append('Stack', label, 'optimized', 'tuned', '', scores.mean(), False)
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())
```

```
Accuracy: 0.62 (+/- 0.01) [Random Forest]
Accuracy: 0.75 (+/- 0.02) [LinearSVC]
Accuracy: 0.87 (+/- 0.02) [MultinomialNB]
Accuracy: 0.66 (+/- 0.00) [Stacking LR]

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

In [68]:
```python
#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('F1-Score Weighted'); plt.xlabel('Classifier'); plt.title('Stacking Ensemble');
plt.show()
```



```
In [69]: #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=False, style='g
         plt.show()
```



```
In [70]: 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']
         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='f1_weighted')
             print("F1-Score Weighted: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
             if (label == 'Stacking Boosted LR'):
                 results = Result_Update_Or_Append('Stack', label, 'optimized', 'tuned', '', scores.mean(), False)
             clf_cv_mean.append(scores.mean())
             clf_cv_std.append(scores.std())
```

```
F1-Score Weighted: 0.80 (+/- 0.01) [Boosted RF]
F1-Score Weighted: 0.80 (+/- 0.01) [Boosted SVC]
F1-Score Weighted: 0.72 (+/- 0.03) [Boosted NB]
F1-Score Weighted: 0.52 (+/- 0.04) [Stacking Boosted LR]

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

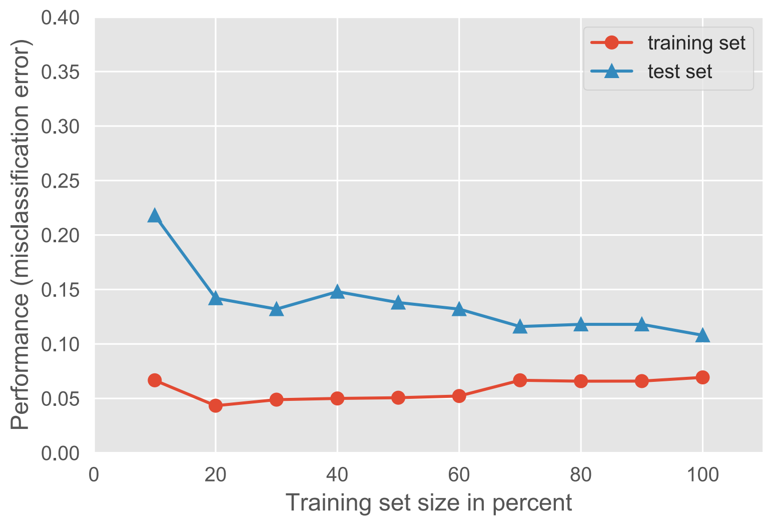
The Stacking performed poorly on our modeling.

Voting

In [71]:
```
from sklearn.ensemble import VotingClassifier

clf2 = svm.SVC(kernel='linear', probability=True)

labels = ['hard', 'soft']

for label in labels:
    eclf1 = VotingClassifier(estimators=[('Random Forest', clf1), ('LinearSVC', clf2), ('MultinomialNB', clf3), ('Logistic Reg
    scores = cross_val_score(eclf1, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
    print("F1-Score Weighted: %.7f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), "Voting: " + label))
    results = Result_Update_Or_Append('Voting', 'Voting ' + label, 'optimized', 'tuned', '', scores.mean(), False)

eclf2 = VotingClassifier(estimators=[('Random Forest', clf1), ('LinearSVC', clf2), ('MultinomialNB', clf3), ('Logistic Regress
scores = cross_val_score(eclf2, rm_chi_opt_bow.x_train_sel, y_train, cv=3, scoring='f1_weighted')
print("F1-Score Weighted: %.7f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), "Weighted Voting: " + label))
results = Result_Update_Or_Append('Voting', 'Weighted Voting soft', 'optimized', 'tuned', '', scores.mean(), False)

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

```
F1-Score Weighted: 0.8492384 (+/- 0.01) [Voting: hard]
F1-Score Weighted: 0.8674039 (+/- 0.02) [Voting: soft]
F1-Score Weighted: 0.8692584 (+/- 0.02) [Weighted Voting: soft]
```

In [72]:
```
#plot Voting 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, eclf2, print_model=False, style='
plt.ylim(0.00, 0.40)
plt.show()
```



New Method: - A Keras Based LSTM RNN Classifier on a Word2Vec Feature matrix

In [73]:
```
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

In [74]:
```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin

from keras.models import Model, Input
from keras.layers import Dense, LSTM, Dropout, Embedding, SpatialDropout1D, Bidirectional, concatenate
from keras.layers import GlobalAveragePooling1D, GlobalMaxPooling1D
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

class KerasTextClassifier:
    __author__ = "Edward Ma"
    __copyright__ = "Copyright 2018, Edward Ma"
    __credits__ = ["Edward Ma"]
    __license__ = "Apache"
    __version__ = "2.0"
    __maintainer__ = "Edward Ma"
    __email__ = "makcedward@gmail.com"

    OOV_TOKEN = "UnknownUnknown"

    def __init__(self,
                 max_word_input, word_cnt, word_embedding_dimension, labels,
                 batch_size, epoch, validation_split,
                 verbose=0):
        self.verbose = verbose
        self.max_word_input = max_word_input
        self.word_cnt = word_cnt
        self.word_embedding_dimension = word_embedding_dimension
        self.labels = labels
        self.batch_size = batch_size
        self.epoch = epoch
        self.validation_split = validation_split

        self.label_encoder = None
        self.classes_ = None
        self.tokenizer = None

        self.model = self._init_model()
        self._init_label_encoder(y=labels)
        self._init_tokenizer()

    def _init_model(self):
        input_layer = Input((self.max_word_input,))
        text_embedding = Embedding(
            input_dim=self.word_cnt+2, output_dim=self.word_embedding_dimension,
            input_length=self.max_word_input, mask_zero=False)(input_layer)

        text_embedding = SpatialDropout1D(0.5)(text_embedding)

        bilstm = Bidirectional(LSTM(units=256, return_sequences=True, recurrent_dropout=0.5))(text_embedding)
        x = concatenate([GlobalAveragePooling1D()(bilstm), GlobalMaxPooling1D()(bilstm)])
        x = Dropout(0.5)(x)
        x = Dense(128, activation="relu")(x)
        x = Dropout(0.5)(x)

        output_layer = Dense(units=len(self.labels), activation="softmax")(x)
        model = Model(input_layer, output_layer)
        model.compile(
            optimizer="adam",
            loss="sparse_categorical_crossentropy",
            metrics=["accuracy"])
        return model

    def _init_tokenizer(self):
        self.tokenizer = Tokenizer(
            num_words=self.word_cnt+1, split=' ', oov_token=self.OOV_TOKEN)

    def _init_label_encoder(self, y):
        self.label_encoder = LabelEncoder()
        self.label_encoder.fit(y)
        self.classes_ = self.label_encoder.classes_

    def _encode_label(self, y):
        return self.label_encoder.transform(y)

    def _decode_label(self, y):
        return self.label_encoder.inverse_transform(y)

    def _get_sequences(self, texts):
        seqs = self.tokenizer.texts_to_sequences(texts)
        return pad_sequences(seqs, maxlen=self.max_word_input, value=0)
    def _preprocess(self, texts):
        # Placeholder only.
        return [text for text in texts]

    def _encode_feature(self, x):
        #self.tokenizer.fit_on_texts(self._preprocess(x))
        self.tokenizer.fit_on_texts(x)
        self.tokenizer.word_index = {e: i for e,i in self.tokenizer.word_index.items() if i <= self.word_cnt}
        self.tokenizer.word_index[self.tokenizer.oov_token] = self.word_cnt + 1
        return self._get_sequences(self._preprocess(x))

    def fit(self, X, y):
        """
            Train the model by providing x as feature, y as label

            :params x: List of sentence
            :params y: List of label
        """

        encoded_x = X #self._encode_feature(X)
        encoded_y = self._encode_label(y)

        self.model.fit(encoded_x, encoded_y,
                       batch_size=self.batch_size, epochs=self.epoch,
                       validation_split=self.validation_split)

    def predict_proba(self, X, y=None):
        encoded_x = self._get_sequences(self._preprocess(X))
        return self.model.predict(encoded_x)

    def predict(self, X, y=None):
        y_pred = np.argmax(self.predict_proba(X), axis=1)
        return self._decode_label(y_pred)
```

In [75]:
```
names = ['Keras']
def build_model(names, x, y):
    pipelines = []

    for name in names:
        print('train %s' % name)

        pipeline = KerasTextClassifier(
            max_word_input=100, word_cnt=30000, word_embedding_dimension=100,
            labels=list(set(y_train.tolist())), batch_size=128, epoch=500, validation_split=0.1)

        pipeline.fit(x, y)
        pipelines.append({'name': name, 'pipeline': pipeline})

    return pipelines
```

In [76]:
```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in X_train]
tokenized_corpus_test = [wpt.tokenize(document) for document in X_test]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)
```

```
In [77]: def Get_W2V_Model(feat_size):
             w2v_mod = word2vec.Word2Vec(tokenized_corpus, size=feat_size,
                                   window=window_context, min_count = min_word_count,
                                   sample=sample, iter=100)
             return w2v_mod

         def average_word_vectors(words, model, vocabulary, num_features):

             feature_vector = np.zeros((num_features,),dtype="float64")
             nwords = 0.

             for word in words:
                 if word in vocabulary:
                     nwords = nwords + 1.
                     feature_vector = np.add(feature_vector, model[word])

             if nwords:
                 feature_vector = np.divide(feature_vector, nwords)

             return feature_vector


         def averaged_word_vectorizer(corpus, model, num_features):
             vocabulary = set(model.wv.index2word)
             features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
                             for tokenized_sentence in corpus]
             return np.array(features)
```

```
In [78]: w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w2v_model,
                                                      num_features=feature_size)
         pd.DataFrame(w2v_feature_array)
```

```
Out[78]:
               0          1          2          3          4          5          6          7          8          9         10         11
   0  -1.3312345  0.8442764 -0.4991527 -0.5021745 -0.8322402  0.0309065 -0.5367231 -0.1731812  1.0658388 -1.3871197 -0.1602523 -0.4871433 -0.28314
   1  -0.4821480  0.1799417 -0.1487643 -0.3790470 -0.8508312  0.1728734 -0.2748752 -0.5173210 -0.0052769 -0.1826608 -0.4974244 -0.1147543  0.16800
   2  -1.3723073  1.1678853  0.3676666  0.0207735 -0.8026465  0.2584133 -0.0145578 -0.7979925 -0.3477923 -0.0492855 -0.7262382  1.4620123 -0.59319
   3  -0.6603885 -0.3710861 -0.5297987 -0.0719619 -0.6716781  0.5123873 -0.4310504 -0.4722893  0.5978767 -1.0871523 -0.2788596 -0.6078884  0.10500
   4  -0.3734234  0.8881036  0.2888957 -0.1819863 -0.6273411  0.6878009 -0.2893474 -1.1814670  0.3573857 -1.4139143 -0.1626995 -0.6190537 -0.40584
 ...         ...        ...        ...        ...        ...        ...        ...        ...        ...        ...        ...        ...
1495   0.0516278  0.6725624 -0.0349346 -0.2282780 -0.1228931 -0.2207755 -0.3305663 -0.4531567  0.2010263 -1.0292585 -0.0923110  0.3014351 -0.86344
1496  -1.5174414  0.7087296 -0.1051768  0.3922001 -1.0883881  1.6235439 -0.8145349 -0.1249675  0.2172512 -0.6893953 -0.2822263 -1.1614270 -0.30085
1497  -0.8132665  1.2675030 -0.0648667 -0.2097883 -0.4225448  0.1922942 -0.0887497 -0.4862867  0.2793771  0.3243506 -1.0880130  0.8490449 -0.25567
1498  -1.0705797  0.6413399 -0.4644007 -0.1221672 -1.0240761  0.6370830 -0.4524890 -0.5656550  1.1340843 -1.4255551  0.1216372 -0.1450912 -0.37460
1499  -1.0669445  0.6884673  0.0554788 -0.2054361 -1.2291761  0.5970930 -0.1212397 -0.4167460  0.1504088  0.0707212 -0.4044700 -0.2612854 -1.06803

1500 rows × 100 columns
```

```
In [79]: w2v_test_array = averaged_word_vectorizer(corpus=tokenized_corpus_test, model=w2v_model,
                                                   num_features=feature_size)

         print(w2v_test_array.shape)

(500, 100)
```

```
In [93]: len(y_train)

Out[93]: 1500
```

```
In [94]: lstm_model = KerasTextClassifier(
                         max_word_input=100, word_cnt=30000, word_embedding_dimension=100,
                         labels=list(set(y_train.tolist())), batch_size=128, epoch=150, validation_split=0.2)
         #pipelines = build_model(names, w2v_feature_array, y_train)
         #lstm_model.fit(w2v_feature_array, y_train)
```

Summary of Findings

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

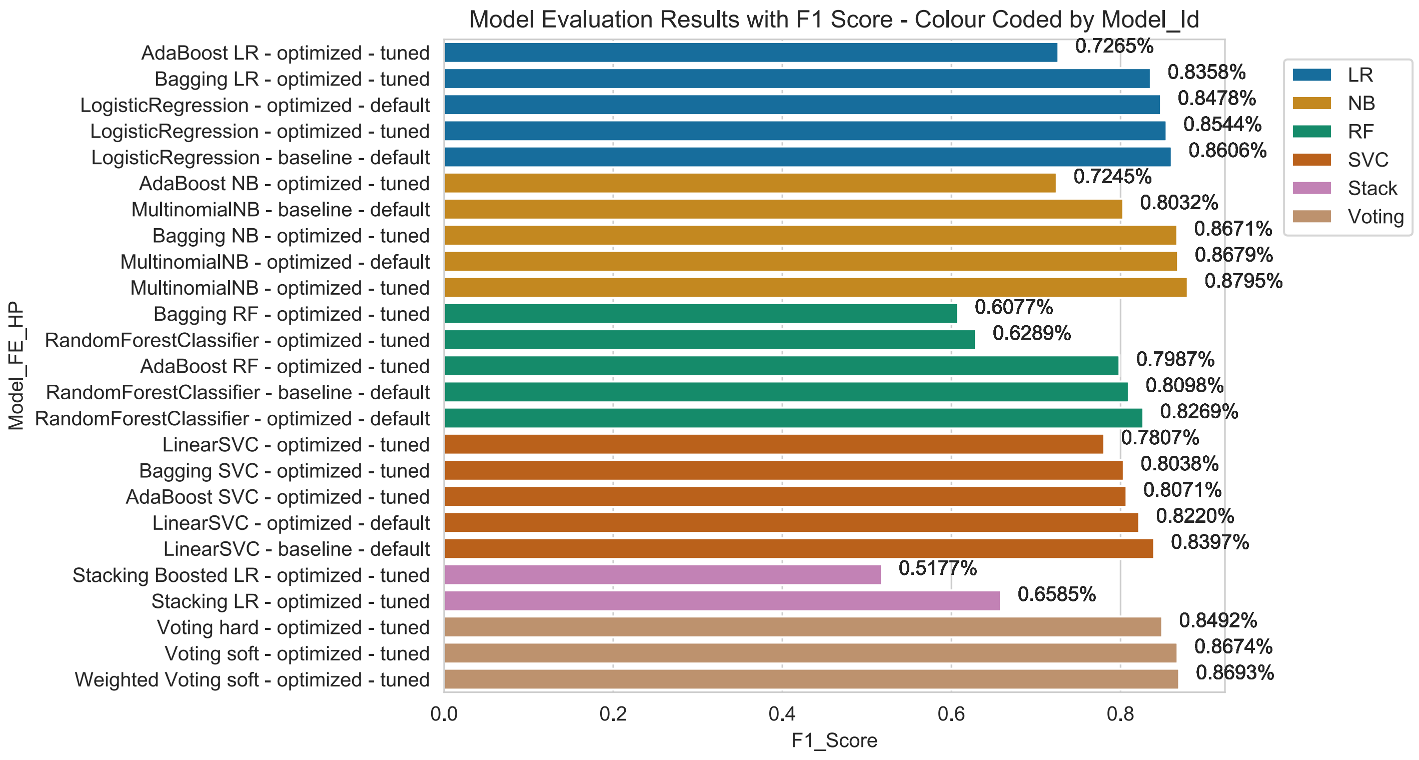
```
In [81]: result_df = cv_df
         result_df[['Model_Id','Model', 'Features', 'Hyper_Param', 'Best_Params', 'F1_Score']]
```

Out[81]:

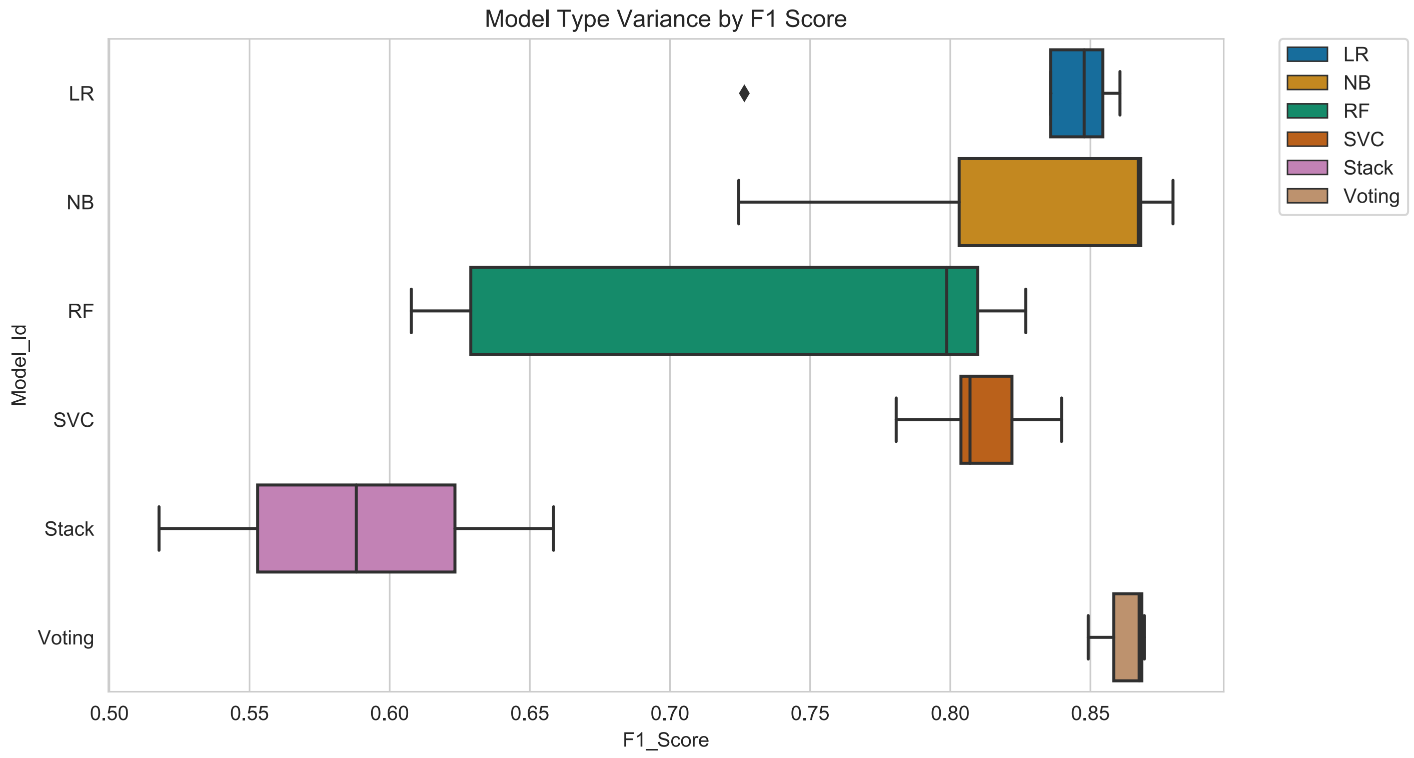
	Model_Id	Model	Features	Hyper_Param	Best_Params	F1_Score
0	RF	RandomForestClassifier	baseline	default		0.8097789
1	SVC	LinearSVC	baseline	default		0.8397106
2	NB	MultinomialNB	baseline	default		0.8031807
3	LR	LogisticRegression	baseline	default		0.8605552
4	RF	RandomForestClassifier	optimized	default		0.8269332
5	SVC	LinearSVC	optimized	default		0.8220224
6	NB	MultinomialNB	optimized	default		0.8679468
7	LR	LogisticRegression	optimized	default		0.8477885
8	RF	RandomForestClassifier	optimized	tuned	{'max_depth': 6, 'n_estimators': 90, 'random_state': 0}	0.6289337
9	SVC	LinearSVC	optimized	tuned	{'C': 1500, 'dual': True, 'loss': 'hinge', 'max_iter': 900, 'penalty': 'l2'}	0.7807117
10	NB	MultinomialNB	optimized	tuned	{'alpha': 0.44, 'fit_prior': False}	0.8794765
11	LR	LogisticRegression	optimized	tuned	{'C': 0.1, 'dual': False, 'multi_class': 'auto', 'penalty': 'l2'}	0.8544467
12	RF	Bagging RF	optimized	tuned		0.6077235
13	SVC	Bagging SVC	optimized	tuned		0.8038190
14	NB	Bagging NB	optimized	tuned		0.8670945
15	LR	Bagging LR	optimized	tuned		0.8357849
16	RF	AdaBoost RF	optimized	tuned		0.7987084
17	SVC	AdaBoost SVC	optimized	tuned		0.8070579
18	NB	AdaBoost NB	optimized	tuned		0.7245388
19	LR	AdaBoost LR	optimized	tuned		0.7265426
20	Stack	Stacking LR	optimized	tuned		0.6584707
21	Stack	Stacking Boosted LR	optimized	tuned		0.5177026
22	Voting	Voting hard	optimized	tuned		0.8492384
23	Voting	Voting soft	optimized	tuned		0.8674039
24	Voting	Weighted Voting soft	optimized	tuned		0.8692584

Model Evaluation Results with F1 Score

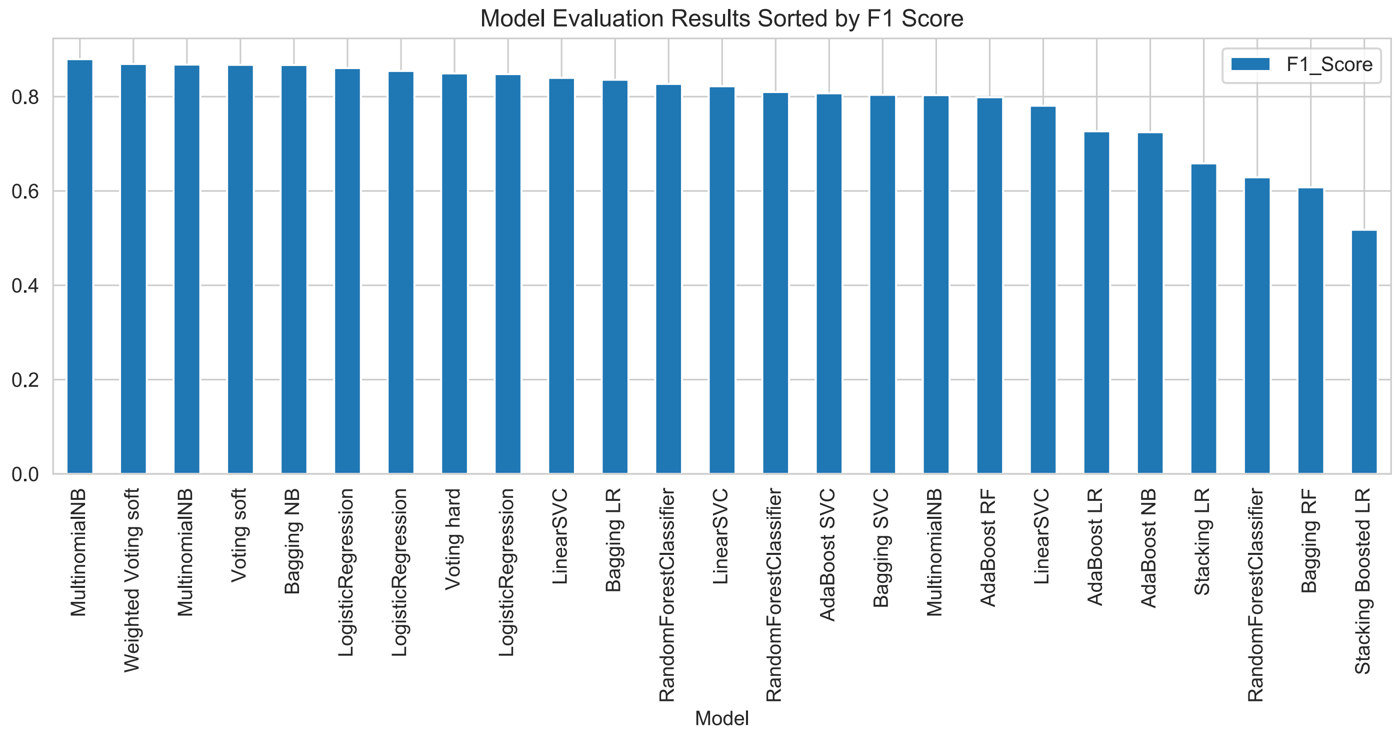
In [82]:
```
result_df['Model_FE_HP'] = result_df['Model'] + ' - ' + result_df['Features'] + ' - ' + result_df['Hyper_Param']
#fig = plt.subplots(figsize=(6, 6))
fig, (ax1) = plt.subplots(figsize=(7, 6), ncols=1)
g = sns.barplot(x='F1_Score', y='Model_FE_HP', data=result_df.sort_values(by=['Model_Id','F1_Score']), palette="colorblind", h
plt.title("Model Evaluation Results with F1 Score - Colour Coded by Model_Id")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=1.)

total = len(result_df['Model_FE_HP'])
def annotateBars(row, ax=ax):
    for p in ax.patches:
        val = '{:.4f}%'.format(p.get_width())
        #percentage = '{:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(val, (x, y))

plot = result_df.apply(annotateBars, ax=ax1, axis=1)
plt.show()
```



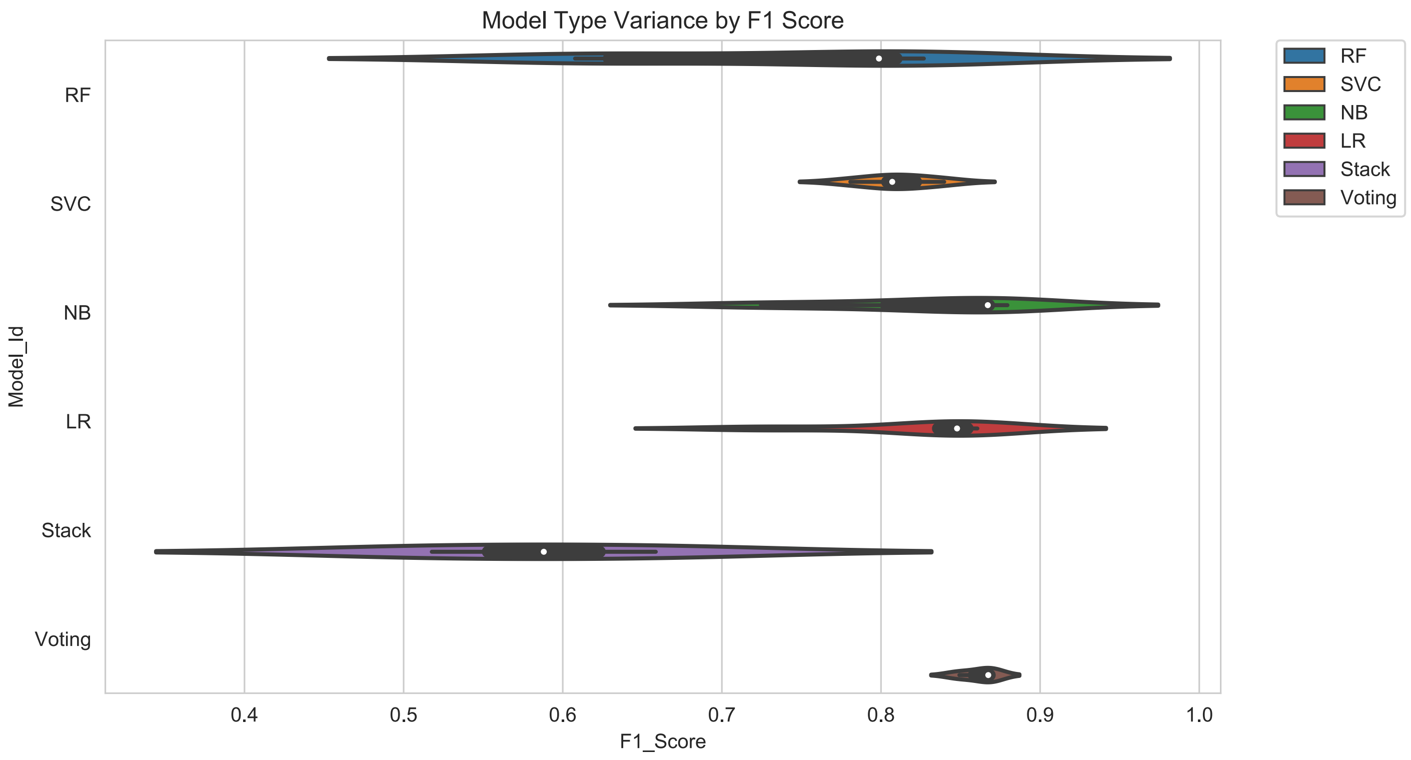
In [83]:
```
fig = plt.subplots(figsize=(10, 6))
sns.boxplot(x='F1_Score', y='Model_Id', data=result_df.sort_values(by='Model_Id'), palette="colorblind", hue='Model_Id', dodge
plt.title("Model Type Variance by F1 Score")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
```



In [84]:
```
result_df.sort_values(by='F1_Score', ascending=False).\
plot(y="F1_Score", x="Model", kind='bar', title="Model Evaluation Results Sorted by F1 Score", figsize=(12, 4));
```



```
In [85]: fig = plt.subplots(figsize=(10, 6))
         #sns.boxplot(x='F1_Score', y='Model_Id', data=result_df.sort_values(by='Model_Id'), palette="colorblind", hue='Model_Id', dodg
         plt.title("Model Type Variance by F1 Score")
         sns.violinplot(x='F1_Score', y='Model_Id', data=result_df, hue='Model_Id', size=6, jitter=True, edgecolor="gray", linewidth=2)
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
```

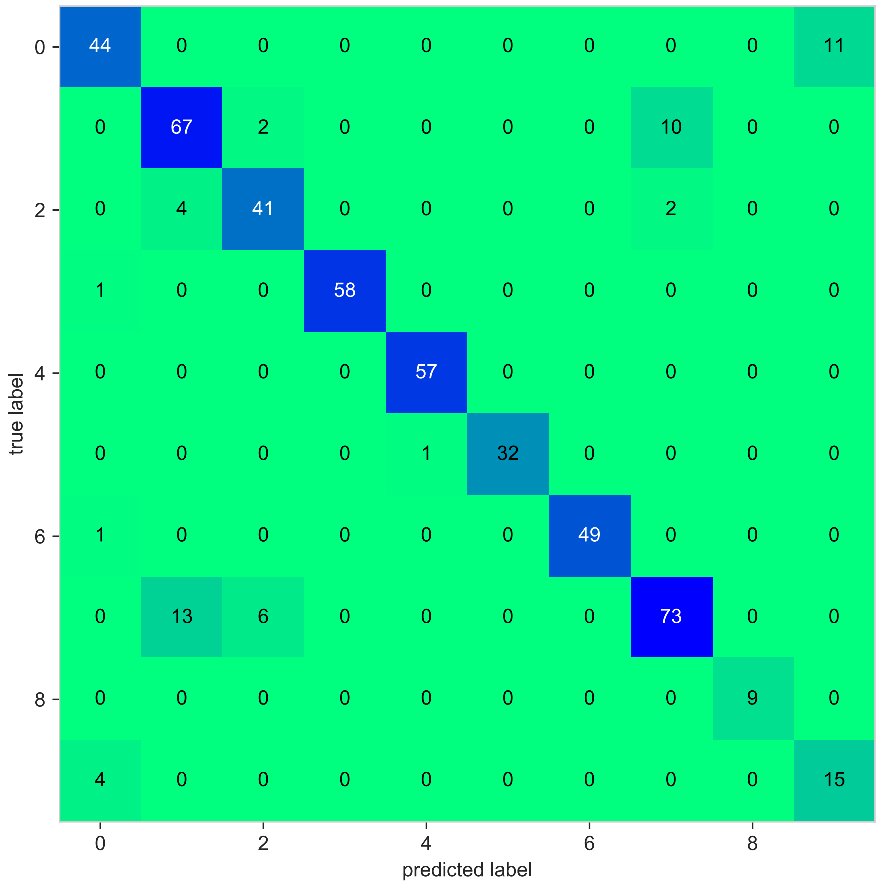


```
In [86]: clf_final_results = Build_Model(eclf2, rm_chi_opt_bow.x_train_sel, y_train, rm_chi_opt_bow.x_test_sel, y_test)
```

```
In [87]: category_id_df = corpus_df[['Instruction_id', 'category']].drop_duplicates().sort_values('category')
         category_to_id = dict(category_id_df.values)
         id_to_category = dict(category_id_df[['category', 'Instruction_id']].values)
```

In [88]:
```
from mlxtend.plotting import plot_confusion_matrix

plot_confusion_matrix(clf_final_results.cm,cmap = 'winter_r',figsize = (7.5, 7.5))
```

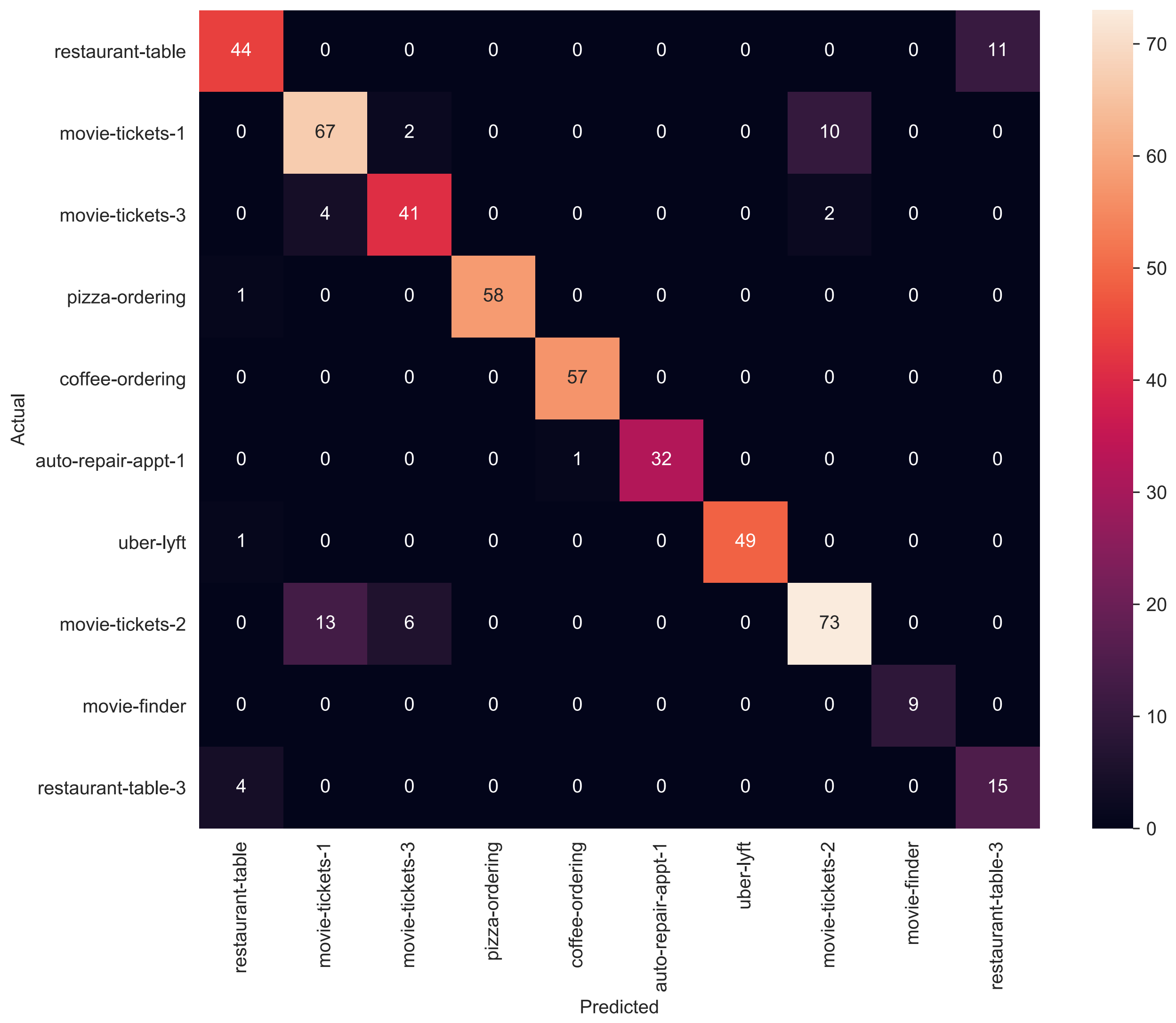
Out[88]:
```
(<Figure size 750x750 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x20d5993a3c8>)
```



In [89]:
```
print("Label" + clf_final_results.report)
```

```
Label              precision    recall  f1-score   support

           0       0.80      0.88      0.84        50
           1       0.85      0.80      0.82        84
           2       0.87      0.84      0.85        49
           3       0.98      1.00      0.99        58
           4       1.00      0.98      0.99        58
           5       0.97      1.00      0.98        32
           6       0.98      1.00      0.99        49
           7       0.79      0.86      0.82        85
           8       1.00      1.00      1.00         9
           9       0.79      0.58      0.67        26

    accuracy                           0.89       500
   macro avg       0.90      0.89      0.90       500
weighted avg       0.89      0.89      0.89       500
```

In [90]:
```
fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(clf_final_results.cm, annot=True, fmt='d',
            xticklabels=category_id_df.Instruction_id.values, yticklabels=category_id_df.Instruction_id.values)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [ ]:

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)

Dataset:

The dataset used in this project is the **Taskmaster-1** dataset from Google. 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)

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	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)
df_all = df_all[df_all.selfdialog_norm != '']
print(df_all.shape)
```

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

In [6]:
```
print (df_all.groupby('Instruction_id').size())
```

```
Instruction_id
auto-repair-appt-1    1160
coffee-ordering       1376
movie-finder            54
movie-tickets-1        678
movie-tickets-2        377
movie-tickets-3        195
pizza-ordering        1467
restaurant-table      1198
restaurant-table-3     102
uber-lyft             1098
dtype: int64
```

In [7]:
```
#weight_higher = ['restaurant-table-2', 'movie-tickets-1', 'movie-tickets-3','uber-lift-2','coffee-ordering-1','coffee-orderin
class_sample_size_dict = {
    "auto-repair-appt-1": 230,
    "coffee-ordering": 230,
    "movie-finder": 54,
    "movie-tickets-1": 250,
    "movie-tickets-2": 250,
    "movie-tickets-3": 195,
    "pizza-ordering": 230,
    "restaurant-table": 230,
    "restaurant-table-3": 101,
    "uber-lyft": 230
}
sum(class_sample_size_dict.values())
```

Out[7]:
```
2000
```

Get a Sample of records.

In [8]:
```
# Function to Get balanced Sample - Get a bit more than needed then down sample
def sampling_k_elements(group):
    name = group['Instruction_id'].iloc[0]
    k = class_sample_size_dict[name]
    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)
print (corpus_df.groupby('Instruction_id').size(), corpus_df.shape)
```

```
Instruction_id
auto-repair-appt-1    230
coffee-ordering       230
movie-finder           54
movie-tickets-1       250
movie-tickets-2       250
movie-tickets-3       195
pizza-ordering        230
restaurant-table      230
restaurant-table-3    101
uber-lyft             230
dtype: int64 (2000, 3)
```

Generate Corpus List

In [9]:
```
doc_lst = []
for i, row in corpus_df.iterrows():
    doc_lst.append(row.selfdialog_norm)

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

```
2000
```

Out[9]:
```
['hi im issue car help sure whats problem light came saying headlight ok want get fixed right away today would ideal already know want take yes intelligent auto solutions ok let pull website online scheduler see today ok im looks like two appointments open today could minutes im least minutes away ok time would pm tonight tell able fix spot call confirm makemodel car kia soul ok said parts done appointment thats great news please book yes booked online thanks give info yes text youll phone thank big help',
 'hi schedule appointment car okay auto repair shop would like check check intelligent auto solutions car bringing lexus im driving put name cell phone number yes put jeff green cell phone number seems problem car makes sound step brakes anything else would like check like oil change maintenance yes think im due oil change well got let check online see available check bring mins able make appointment bring car time pm great thanks initial cost brake checkup oil change okay accept credit card yes great thanks bye youre welcome bye',
 'assistant favor yes course whats going car making weird rattly noises think checked find good mechanic certainly im checking google right moment ok appears auto shop near work star rating want give call yes please ok ill put hold moment see say great thanks ok im back said bring tomorrow ok long going keep depends whats going said could problem muffler wont know look gave number theyll give call alright make sure get uber tomorrow morning yes time well probably need leave house ok ill house get car ill make sure uber arrives well thank much youre welcome need anything else ok see tomorrow',
 'gail need help schedule appointment intelligent auto solutions car whats wrong car need schedule appointment look radiator see drops fluid time park ground ok year model car bmw series sure name use use name scolar timer address miklan road forest hills new mexico bring car tomorrow see get earlier situation annoying time bring work pm take abut minutes ok let check would prefer bring tomorrow morning let check time slots way please reserve car use mean time case car kept overnight well checked time bring pm today ok let confirm everything bring car today pm check leaking radiator get car ise case car stays overnight thats correct repair shop need initial inspection thats ok go right ahead book appointment sure everything booked requested thanks help talk later']
```

In [10]:
```
category_id_df = corpus_df[['Instruction_id', 'category']].drop_duplicates().sort_values('category')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category', 'Instruction_id']].values)
```

Split Data into Train and Test Sets

In [11]:
```
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 = 0)
X_train, X_test, y_train, y_test = train_test_split(doc_lst, corpus_df['Instruction_id'], test_size=0.25, random_state = 0)
```

In [12]:
```
# from __future__ import print_function
# import lime
# import sklearn
# import numpy as np
# import sklearn
# import sklearn.ensemble
# import sklearn.metrics
```

In [13]:
```
# vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lowercase=False)
# train_vectors = vectorizer.fit_transform(X_train)
# test_vectors = vectorizer.transform(X_test)
```

In [14]:
```
# from sklearn.naive_bayes import MultinomialNB
# nb = MultinomialNB(alpha=.01)
# nb.fit(train_vectors, y_train)
```

In [15]:
```
# pred = nb.predict(test_vectors)
# sklearn.metrics.f1_score(y_test, pred, average='weighted')
```

In [16]:
```
# from lime import lime_text
# from sklearn.pipeline import make_pipeline
# c = make_pipeline(vectorizer, nb)
```

In [17]:
```
# cats = set(corpus_df['Instruction_id'])
```

In [18]:
```
# class_names = list(cats)
# class_names
```

In [19]:
```
# from lime.lime_text import LimeTextExplainer
# explainer = LimeTextExplainer(class_names=class_names)
```

In [20]:
```
# idx = 3
# print('Document id: %d' % idx)
# exp = explainer.explain_instance(X_test[idx], c.predict_proba, num_features=6, top_labels=5)
# pred_class = pred[idx] # nb.predict(test_vectors[idx])
# print('Predicted class =', pred_class)
# print('True class:', y_test.iloc[idx])
```

In [21]:
```
# pred_cat = category_to_id[pred_class]
# print ('Explanation for class %s' % pred_class, 'Category', pred_cat)
# print (exp.as_list(label=pred_cat))
# # print ('\n'.join(map(str, exp.as_list(label=0))))
```

In [22]:
```
# exp = explainer.explain_instance(X_test.iloc[idx], c.predict_proba, num_features=6, top_labels=3)
# print(exp.available_labels())
```

In [23]:
```
# exp.show_in_notebook(text=False)
```

In [24]:
```
# exp.show_in_notebook(text=X_test[idx])
```

Build Vocabulary

In [25]:
```
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: 7209
Vocabulary Sample: [('like', 1), ('would', 2), ('ok', 3), ('okay', 4), ('pm', 5), ('yes', 6), ('tickets', 7), ('want', 8),
('order', 9), ('time', 10)]
```

Bag of Words Feature Extraction

In [26]:
```python
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[26]:
```
array([[0, 5, 4, ..., 0, 0, 0],
       [0, 4, 4, ..., 0, 0, 0],
       [0, 2, 2, ..., 0, 0, 0],
       ...,
       [0, 3, 3, ..., 0, 0, 0],
       [0, 3, 4, ..., 1, 1, 1],
       [0, 2, 2, ..., 0, 0, 0]], dtype=int64)
```

In [27]:
```python
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
X_train_features = pd.DataFrame(cv_matrix, columns=vocab)
X_train_features
```

Out[27]:

	PAD	like	would	ok	okay	pm	yes	tickets	want	order	time	thank	see	movie	please	...	xs	yeppers	pinkitzel	jessie	librarys	wednesda
0	0	5	4	0	3	2	1	3	1	0	4	1	1	3	0	...	0	0	0	0	0	
1	0	4	4	0	0	1	0	4	0	1	0	2	0	1	1	...	0	0	0	0	0	
2	0	2	2	0	0	0	0	0	1	0	0	0	0	0	1	...	0	0	0	0	0	
3	0	1	1	0	7	3	1	4	1	0	1	0	0	1	0	...	0	0	0	0	0	
4	0	1	1	7	0	3	0	5	2	0	1	3	2	2	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
1495	0	4	4	1	7	1	3	1	0	2	3	2	1	1	2	...	0	0	0	0	0	
1496	0	1	1	3	0	0	1	0	0	2	0	0	1	0	1	...	0	0	0	0	0	
1497	0	3	3	0	1	0	1	0	3	0	1	0	1	0	0	...	0	0	0	0	0	
1498	0	3	4	5	0	3	2	3	1	0	0	0	1	6	1	...	0	0	0	0	0	
1499	0	2	2	0	6	3	2	2	1	0	1	0	2	2	0	...	0	0	0	0	0	

1500 rows × 7209 columns

In [28]:
```python
# 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)
```

```
(1500, 7209)
(500, 7209)
(500,)
```

In [29]:
```python
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
```

Interpretability - Features Importances

In [30]:
```python
from sklearn.ensemble import RandomForestClassifier

model_rf_bow = RandomForestClassifier(max_depth=6, n_estimators=90, random_state=2)
rm_rf_bow = Build_Model(model_rf_bow, X_train_bow, y_train, X_test_bow, y_test)
```

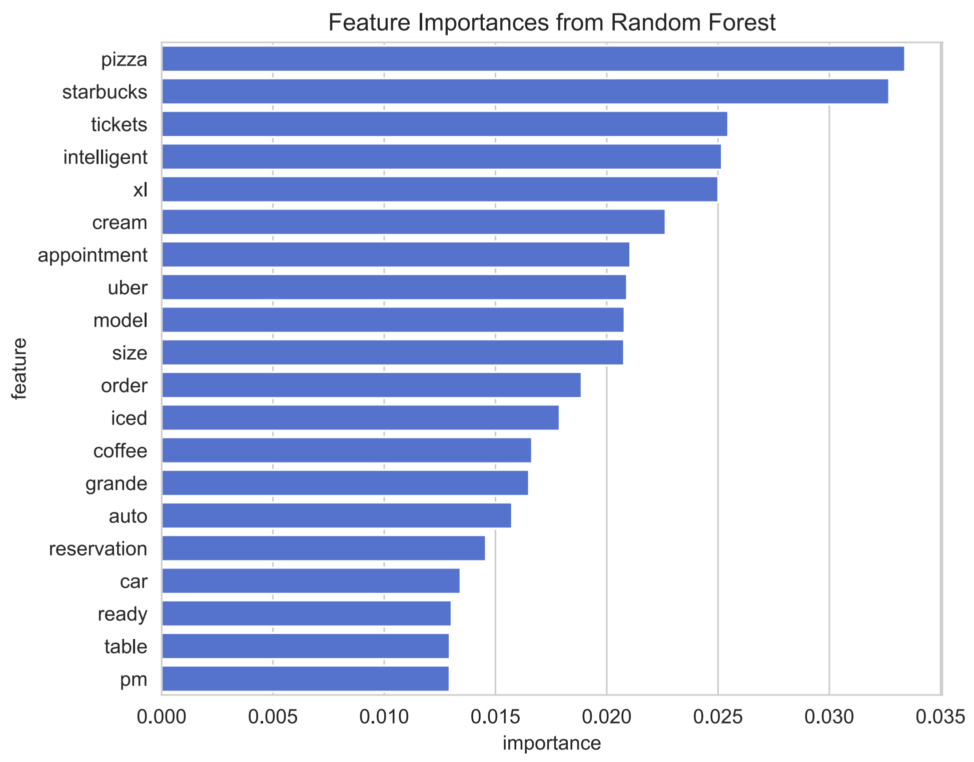
```
In [31]: importances = model_rf_bow.feature_importances_

         # train_features is the dataframe of training features
         feature_list = list(X_train_features.columns)

         # Extract the feature importances into a dataframe
         feature_results = pd.DataFrame({'feature': feature_list, 'importance': importances})

         # Show the top 20 most important
         feature_results = feature_results.sort_values('importance', ascending = False).reset_index(drop=True)

         feature_results.head(20)
```

Out[31]:

	feature	importance
0	pizza	0.0334075
1	starbucks	0.0326899
2	tickets	0.0254586
3	intelligent	0.0251725
4	xl	0.0250139
5	cream	0.0226447
6	appointment	0.0210595
7	uber	0.0209102
8	model	0.0208026
9	size	0.0207771
10	order	0.0188729
11	iced	0.0178846
12	coffee	0.0166441
13	grande	0.0165070
14	auto	0.0157454
15	reservation	0.0145711
16	car	0.0134314
17	ready	0.0130299
18	table	0.0129445
19	pm	0.0129402

In [32]:
```
fig, (ax1) = plt.subplots(figsize=(7, 6), ncols=1)
g = sns.barplot(x='importance', y='feature', data=feature_results.head(20), color='royalblue', ci=None, ax=ax1)
plt.title("Feature Importances from Random Forest")
```

Out[32]:
```
Text(0.5, 1.0, 'Feature Importances from Random Forest')
```

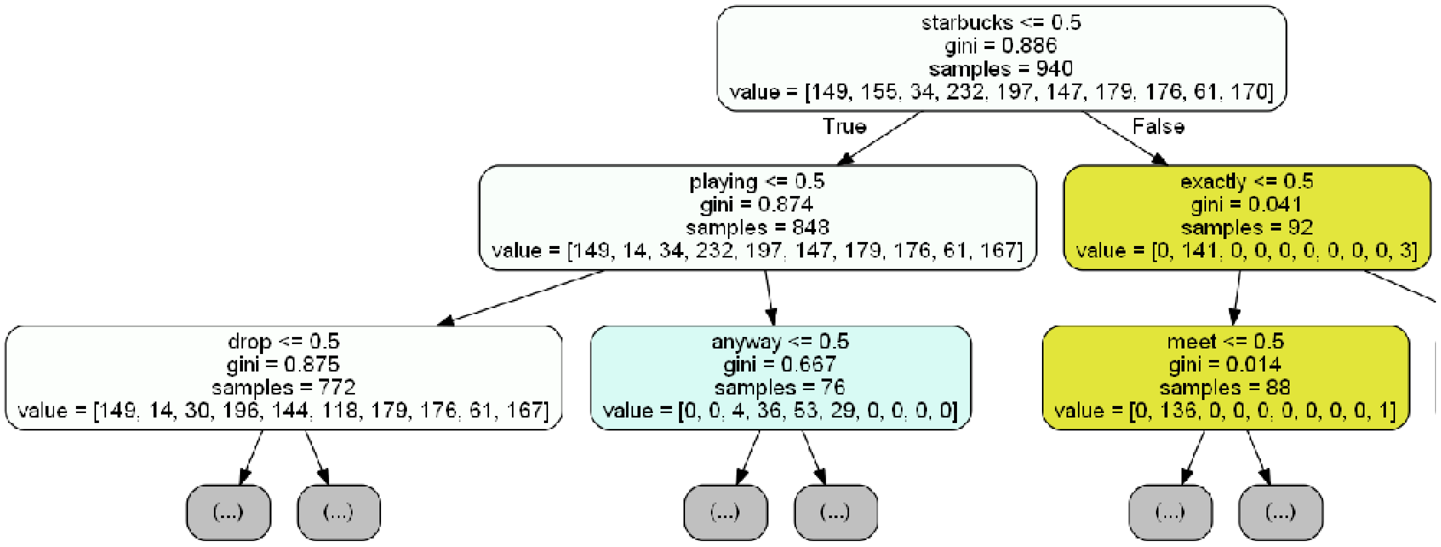


In [33]:
```
from sklearn import tree

# Extract a single tree (number 105)
single_tree = model_rf_bow.estimators_[6]

# Save the tree to a dot file
tree.export_graphviz(single_tree, out_file = 'images/tree.dot',
                     feature_names = feature_list)

# Convert to a png from the command line
# This requires the graphviz visualization library (https://www.graphviz.org/)
!dot -Tpng images/tree.dot -o images/tree.png

single_tree
```

Out[33]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=6, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1869695442, splitter='best')
```

In [34]:
```
tree.export_graphviz(single_tree, out_file = 'images/tree_small.dot', rounded = True, feature_names = feature_list, filled = T
!dot -Tpng images/tree_small.dot -o images/tree_small.png
```

In [35]:
```
import matplotlib.image as mpimg

img=mpimg.imread('images/tree_small.png')
fig, ax = plt.subplots(figsize=(16, 10))
imgplot = ax.imshow(img)
ax.grid(False)
ax.axis('off')
plt.show()
```


starbucks <= 0.5
gini = 0.886
samples = 940
value = [149, 155, 34, 232, 197, 147, 179, 176, 61, 170]
True
False
playing <= 0.5
gini = 0.874
samples = 848
value = [149, 14, 34, 232, 197, 147, 179, 176, 61, 167]
exactly <= 0.5
gini = 0.041
samples = 92
value = [0, 141, 0, 0, 0, 0, 0, 0, 0, 3]
drop <= 0.5
gini = 0.875
samples = 772
value = [149, 14, 30, 196, 144, 118, 179, 176, 61, 167]
anyway <= 0.5
gini = 0.667
samples = 76
value = [0, 0, 4, 36, 53, 29, 0, 0, 0, 0]
meet <= 0.5
gini = 0.014
samples = 88
value = [0, 136, 0, 0, 0, 0, 0, 0, 0, 1]
(...)
(...)
(...)
(...)
(...)
(...)

Build Ensemble Model

In [36]:
```
import random
import pandas as pd
import IPython
import xgboost
import keras

import eli5
from eli5.lime import TextExplainer
from lime.lime_text import LimeTextExplainer
print('ELI5 Version:', eli5.__version__)
print('XGBoost Version:', xgboost.__version__)
print('Keras Version:', keras.__version__)
```

```
ELI5 Version: 0.10.1
XGBoost Version: 0.90
Keras Version: 2.3.1
```

In [37]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from xgboost import XGBClassifier
```

In [38]:
```
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin

from keras.models import Model, Input
from keras.layers import Dense, LSTM, Dropout, Embedding, SpatialDropout1D, Bidirectional, concatenate
from keras.layers import GlobalAveragePooling1D, GlobalMaxPooling1D
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

class KerasTextClassifier:
    __author__ = "Edward Ma"
    __copyright__ = "Copyright 2018, Edward Ma"
    __credits__ = ["Edward Ma"]
    __license__ = "Apache"
    __version__ = "2.0"
    __maintainer__ = "Edward Ma"
    __email__ = "makcedward@gmail.com"

    OOV_TOKEN = "UnknownUnknown"

    def __init__(self,
                 max_word_input, word_cnt, word_embedding_dimension, labels,
                 batch_size, epoch, validation_split,
                 verbose=0):
        self.verbose = verbose
        self.max_word_input = max_word_input
        self.word_cnt = word_cnt
        self.word_embedding_dimension = word_embedding_dimension
        self.labels = labels
        self.batch_size = batch_size
        self.epoch = epoch
        self.validation_split = validation_split

        self.label_encoder = None
        self.classes_ = None
        self.tokenizer = None

        self.model = self._init_model()
        self._init_label_encoder(y=labels)
        self._init_tokenizer()

    def _init_model(self):
        input_layer = Input((self.max_word_input,))
        text_embedding = Embedding(
            input_dim=self.word_cnt+2, output_dim=self.word_embedding_dimension,
            input_length=self.max_word_input, mask_zero=False)(input_layer)

        text_embedding = SpatialDropout1D(0.5)(text_embedding)

        bilstm = Bidirectional(LSTM(units=256, return_sequences=True, recurrent_dropout=0.5))(text_embedding)
        x = concatenate([GlobalAveragePooling1D()(bilstm), GlobalMaxPooling1D()(bilstm)])
        x = Dropout(0.5)(x)
        x = Dense(128, activation="relu")(x)
        x = Dropout(0.5)(x)

        output_layer = Dense(units=len(self.labels), activation="softmax")(x)
        model = Model(input_layer, output_layer)
        model.compile(
            optimizer="adam",
            loss="sparse_categorical_crossentropy",
            metrics=["accuracy"])
        return model

    def _init_tokenizer(self):
        self.tokenizer = Tokenizer(
            num_words=self.word_cnt+1, split=' ', oov_token=self.OOV_TOKEN)

    def _init_label_encoder(self, y):
        self.label_encoder = LabelEncoder()
        self.label_encoder.fit(y)
        self.classes_ = self.label_encoder.classes_

    def _encode_label(self, y):
        return self.label_encoder.transform(y)

    def _decode_label(self, y):
        return self.label_encoder.inverse_transform(y)

    def _get_sequences(self, texts):
        seqs = self.tokenizer.texts_to_sequences(texts)
        return pad_sequences(seqs, maxlen=self.max_word_input, value=0)
    def _preprocess(self, texts):
        # Placeholder only.
        return [text for text in texts]

    def _encode_feature(self, x):
        self.tokenizer.fit_on_texts(self._preprocess(x))
        self.tokenizer.word_index = {e: i for e,i in self.tokenizer.word_index.items() if i <= self.word_cnt}
        self.tokenizer.word_index[self.tokenizer.oov_token] = self.word_cnt + 1
        return self._get_sequences(self._preprocess(x))

    def fit(self, X, y):
        """
            Train the model by providing x as feature, y as label

            :params x: List of sentence
            :params y: List of label
        """

        encoded_x = self._encode_feature(X)
        encoded_y = self._encode_label(y)

        self.model.fit(encoded_x, encoded_y,
                       batch_size=self.batch_size, epochs=self.epoch,
                       validation_split=self.validation_split)

    def predict_proba(self, X, y=None):
        encoded_x = self._get_sequences(self._preprocess(X))
        return self.model.predict(encoded_x)

    def predict(self, X, y=None):
        y_pred = np.argmax(self.predict_proba(X), axis=1)
        return self._decode_label(y_pred)
```

In [39]:
```
names_rf_svc = ['Random Forest', 'Linear SVC']
names = ['Random Forest', 'Linear SVC Prob', 'Multinomial NB', 'Logistic Regression']
```

In [40]:
```
from sklearn import svm

def build_model(names, x, y):
    pipelines = []
    vec = TfidfVectorizer()
    vec.fit(x)

    for name in names:
        print('train %s' % name)

        if name == 'Random Forest':
            estimator = RandomForestClassifier(n_jobs=-1)
            pipeline = make_pipeline(vec, estimator)
        elif name == 'Linear SVC Prob':
            estimator = svm.SVC(kernel='linear', probability=True)
            pipeline = make_pipeline(vec, estimator)
        elif name == 'Linear SVC':
            estimator = LinearSVC()
            pipeline = make_pipeline(vec, estimator)
        elif name == 'Multinomial NB':
            estimator = MultinomialNB()
            pipeline = make_pipeline(vec, estimator)
        elif name == 'Logistic Regression':
            LogisticRegression(n_jobs=-1)
            pipeline = make_pipeline(vec, estimator)

        pipeline.fit(x, y)
        pipelines.append({
            'name': name,
            'pipeline': pipeline
        })

    return pipelines, vec
```

In [41]:
```
pipelines, vec = build_model(names, X_train, y_train)
pipelines_rf_svc, vec = build_model(names_rf_svc, X_train, y_train)
```

```
train Random Forest
train Linear SVC Prob
train Multinomial NB
train Logistic Regression
train Random Forest
train Linear SVC
```

ELI5 - Global Interpretation

In [42]:
```
for pipeline in pipelines_rf_svc:
    print('Estimator: %s' % (pipeline['name']))
    labels = pipeline['pipeline'].classes_.tolist()

    estimator = pipeline['pipeline']

    IPython.display.display(
        eli5.show_weights(estimator=estimator, top=10, target_names=labels, vec=vec))
```

```
Estimator: Random Forest
```

Weight	Feature
0.0272 ± 0.0696	tickets
0.0177 ± 0.0599	pizza
0.0124 ± 0.0558	auto
0.0123 ± 0.0399	reservation
0.0122 ± 0.0445	ride
0.0120 ± 0.0351	movie
0.0114 ± 0.0416	starbucks
0.0112 ± 0.0500	milk
0.0111 ± 0.0378	car
0.0109 ± 0.0350	uber
… 7180 more …	

```
Estimator: Linear SVC
```

y=auto-repair-appt-1 top features		**y=coffee-ordering** top features		**y=movie-finder** top features		**y=movie-tickets-1** top features		**y=movie-tickets-2** top features		**y=movie-tickets-3** top features		**y=pizza-** fea
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**
+2.060	appointment	+2.207	starbucks	+1.363	seen	+2.049	tickets	+2.161	us	+1.973	sorry	+2.525
+1.415	car	+1.653	milk	+1.278	comedy	+1.736	glass	+1.644	wonder	+1.797	pet	+1.175
+1.366	auto	+1.392	coffee	+1.247	movies	+1.370	theatre	+1.443	captain	+1.743	shazam	+1.127
+1.076	intelligent	+1.331	latte	+1.184	action	+1.171	popcorn	+1.432	enjoy	+1.585	hellboy	+1.035
+1.013	solutions	+1.072	caramel	+0.980	movie	+1.092	purchase	+1.275	tickets	+1.369	movie	+1.019
+0.891	tomorrow	+1.031	venti	+0.873	master	*… 1582 more positive …*		+1.204	sent	+1.202	buy	+0.966
+0.843	number	+1.025	cream	+0.770	something	*… 3377 more negative …*		+1.199	marvel	+1.182	cancel	+0.897
+0.780	tune	+0.890	drink	+0.679	watch	-1.449	sorry	+1.095	oclock	+1.169	dont	+0.889
+0.746	tires	+0.846	size	*… 994 more positive …*		-1.473	sold	*… 1512 more positive …*		*… 1091 more positive …*		+0.787
… 1291 more positive …		*… 1061 more positive …*		*… 3357 more negative …*		-1.593	shazam	*… 3016 more negative …*		*… 2956 more negative …*		*… 1020 mo*
… 3492 more negative …		*… 3650 more negative …*		-0.682	tickets	-1.783	dumbo	-1.363	buy	-1.159	text	*… 3703 mo*
-0.777	<BIAS>	-1.036	pizza	-0.755	pm	-2.074	us	-1.643	glass	-1.416	sent	-0.760

In [43]:
```
labels
```

Out[43]:
```
['auto-repair-appt-1',
 'coffee-ordering',
 'movie-finder',
 'movie-tickets-1',
 'movie-tickets-2',
 'movie-tickets-3',
 'pizza-ordering',
 'restaurant-table',
 'restaurant-table-3',
 'uber-lyft']
```

ELI5 - Local Interpretation

In [44]:
```python
number_of_sample = 1
sample_ids = [random.randint(0, len(X_test) -1 ) for p in range(0, number_of_sample)]

for idx in sample_ids:
    print('Index: %d' % (idx))
    print(number_of_sample)
#     print('Index: %d, Feature: %s' % (idx, x_test[idx]))
    for pipeline in pipelines_rf_svc:
        print('-' * 50)
        print('Estimator: %s' % (pipeline['name']))

        print('True Label: %s, Predicted Label: %s' % (y_test[idx], pipeline['pipeline'].predict([X_test[idx]])[0]))
        labels = pipeline['pipeline'].classes_.tolist()


        estimator = pipeline['pipeline'].steps[1][1]

        IPython.display.display(
            eli5.show_prediction(estimator, X_test[idx], top=10, vec=vec, target_names=labels))
```

```
Index: 241
1
--------------------------------------------------
Estimator: Random Forest
True Label: movie-tickets-1, Predicted Label: movie-tickets-2
```

y=auto-repair-appt-1 (probability **0.000**) top features		**y=coffee-ordering** (probability **0.010**) top features		**y=movie-finder** (probability **0.010**) top features		**y=movie-tickets-1** (probability **0.400**) top features		**y=movie-tickets-2** (probability **0.430**) top features		**y=movie-tickets-3** (probability **0.100**) top features	
Contribution?	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**
+0.112	<BIAS>	+0.114	<BIAS>	+0.026	<BIAS>	+0.125	<BIAS>	+0.128	<BIAS>	+0.096	<BIAS>
+0.008	make	+0.010	nashville	+0.010	perfect	+0.049	tickets	+0.036	regal	+0.022	something
… 356 more positive …		+0.009	order	+0.007	movie	+0.041	two	+0.035	tickets	+0.019	showing
… 95 more negative …		*… 351 more positive …*		+0.003	something	+0.031	need	+0.030	movie	+0.014	tickets
-0.007	two	*… 93 more negative …*		*… 455 more positive …*		+0.023	theater	+0.028	showing	+0.014	movie
-0.008	appointment	-0.008	movie	*… 81 more negative …*		+0.021	showing	+0.026	us	+0.012	theater
-0.008	car	-0.008	size	-0.003	available	+0.017	many	+0.023	cinemas	+0.010	problem
-0.008	movie	-0.008	two	-0.003	need	+0.016	tonight	+0.018	evening	*… 515 more positive …*	
-0.009	showing	-0.009	showing	-0.004	two	+0.015	middle	+0.017	two	*… 201 more negative …*	
-0.009	number	-0.009	theater	-0.005	showing	*… 544 more positive …*		*… 572 more positive …*		-0.010	another
-0.011	tickets	-0.013	tonight	-0.007	tickets	*… 201 more negative …*		*… 168 more negative …*		-0.014	else
-0.011	intelligent	-0.015	tickets	-0.010	watch	-0.031	us	-0.022	need	-0.016	sorry

```
--------------------------------------------------
Estimator: Linear SVC
True Label: movie-tickets-1, Predicted Label: movie-tickets-1
```

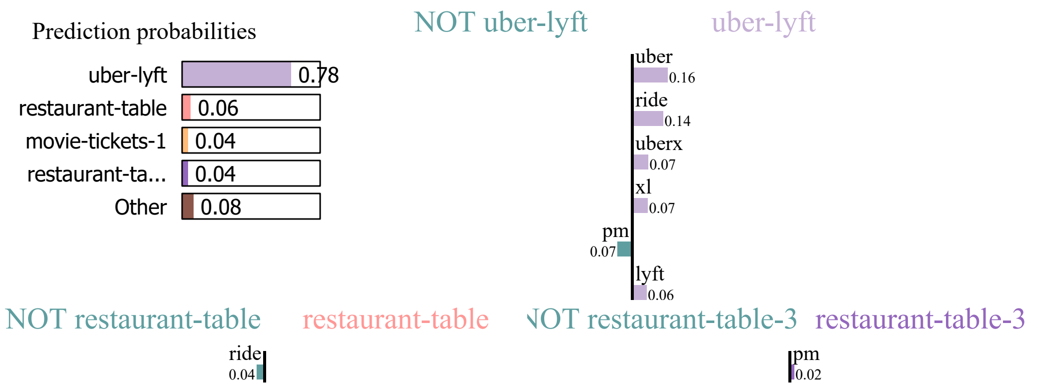
y=auto-repair-appt-1 (score **-1.300**) top features		**y=coffee-ordering** (score **-1.209**) top features		**y=movie-finder** (score **-1.382**) top features		**y=movie-tickets-1** (score **0.555**) top features		**y=movie-tickets-2** (score **0.025**) top features		**y=movie-tickets-3** (score **-1.393**) top features	
Contribution?	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**	**Contribution?**	**Feature**
+0.035	get	+0.045	order	+0.066	something	+0.712	tickets	+0.443	tickets	+0.082	movie
+0.025	problem	*… 11 more positive …*		+0.059	movie	+0.161	two	+0.191	us	+0.068	showing
+0.022	make	*… 26 more negative …*		*… 6 more positive …*		+0.146	upside	+0.173	cinemas	+0.064	location
… 10 more positive …		-0.017	cinemas	*… 31 more negative …*		+0.097	middle	+0.143	regal	*… 17 more positive …*	
… 26 more negative …		-0.019	tonight	-0.029	upside	+0.080	perfect	+0.120	enjoy	*… 20 more negative …*	
-0.026	cinemas	-0.019	pm	-0.030	time	+0.079	cinemas	+0.088	showing	-0.050	somewhere
-0.030	movie	-0.028	movie	-0.031	get	+0.075	hollywood	+0.083	two	-0.052	hollywood
-0.031	order	-0.030	showing	-0.036	pm	+0.075	nashville	*… 18 more positive …*		-0.069	perfect
-0.045	showing	-0.031	need	-0.046	showing	*… 21 more positive …*		*… 19 more negative …*		-0.070	enjoy
-0.069	two	-0.056	two	-0.080	two	*… 16 more negative …*		-0.076	need	-0.092	tickets
-0.236	tickets	-0.188	tickets	-0.237	tickets	-0.184	us	-0.078	upside	-0.098	two
-0.777	<BIAS>	-0.739	<BIAS>	-0.612	<BIAS>	-0.801	<BIAS>	-1.014	<BIAS>	-1.085	<BIAS>

LIME

LIME - Local Interpretation

In [45]:
```
number_of_sample = 1
sample_ids = [random.randint(0, len(X_test) -1 ) for p in range(0, number_of_sample)]

for idx in sample_ids:
    print('Index: %d' % (idx))
    for pipeline in pipelines:
        #if pipeline['name'] != 'Linear SVC':
        print('-' * 50)
        print('Estimator: %s' % (pipeline['name']))
        print('True Label: %s, Predicted Label: %s' % (y_test[idx], pipeline['pipeline'].predict([X_test[idx]])[0]))
        labels = pipeline['pipeline'].classes_.tolist()
        explainer = LimeTextExplainer(class_names=labels)
        exp = explainer.explain_instance(X_test[idx], pipeline['pipeline'].predict_proba, num_features=6, top_labels=3)
        IPython.display.display(exp.show_in_notebook(text=True))
```

```
Index: 116
--------------------------------------------------
Estimator: Random Forest
True Label: uber-lyft, Predicted Label: uber-lyft
```



Skater

Skater - Global Interpretation

In [46]:
```
# Super slow when there is lots of feature(word in this case).......
pipelines, vec = build_model(names, X_train[:2], y_train[:2])
```

```
train Random Forest
train Linear SVC Prob
train Multinomial NB
train Logistic Regression
```

In [47]:
```python
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt

from skater.model import InMemoryModel
from skater.core.explanations import Interpretation

transfromed_x_test = vec.transform(X_test[:2]).toarray()

interpreter = Interpretation(transfromed_x_test, feature_names=vec.get_feature_names())

for pipeline in pipelines:
    #if pipeline['name'] != 'Linear SVC':
    print('-' * 50)
    print('Estimator: %s' % (pipeline['name']))

    estimator = pipeline['pipeline'].steps[1][1]

    print(estimator)
    pyint_model = InMemoryModel(estimator.predict_proba, examples=transfromed_x_test)

    f, axes = plt.subplots(1, 1, figsize = (12, 18))
    ax = axes
    interpreter.feature_importance.plot_feature_importance(pyint_model, ascending=False, ax=ax)
    plt.show()
```

```
--------------------------------------------------
Estimator: Random Forest
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
[74/74] features ████████████████████ Time elapsed: 8 seconds
```

In [ ]:

In [ ]:

In [ ]: