Amazon reviews classification project

Introduction

In this project, I explore two different methods for classifying correctly whether product reviews from Amazon are positive or negative, both belonging to naive Bayes method family: the Bernoulli naive Bayes method and the Multinomial naive Bayes method.

I employ different datasets containing the text of product reviews and corresponding product ratings assigned by each reviewer.

Load the Data

I use data from [http://jmcauley.ucsd.edu/data/amazon/ (http://jmcauley.ucsd.edu/data/amazon/)](http://jmcauley.ucsd.edu/data/amazon/) .

To begin, I use the subset of Toys and Games data.

First, I need to import package I will use:

In [186]:
```python
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

from wordcloud import WordCloud, STOPWORDS

from scipy import stats
```

Then, I load the data in Panda DataFrame structure:

In [187]:
```python
#Load dataset in a panda DataFrame
df = pd.read_json('./Dataset/Toys_and_Games_5.json',lines=True)
```

Let's take a first look at the data. Visualize the first three rows of the dataset and get it's dimension:

In [188]:
```python
#Look at the data
print("Dataset dimension: ", df.shape, "\n")
```

```
Dataset dimension:  (167597, 9)
```

In [189]:
```python
df.iloc[0:3]
```

Out[189]:

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary
0	A1VXOAVRGKGEAK	0439893577	Angie	[0, 0]	I like the item pricing. My granddaughter want...	5	Magnetic board

1	A8R62G708TSCM	0439893577	Candace	[1, 1]	Love the magnet easel... great for moving to d...	4	it works pretty good for moving to different a...
2	A21KH420DK0ICA	0439893577	capemaychristy	[1, 1]	Both sides are magnetic. A real plus when you...	5	love this!

Better organize the data

For our project, we focus our attention only on "reviewText" attribute and "overall" attribute. Even "summary" is an interesting attribute to focus on.

Modify the structure of dataset to include only columns of our interest:

In [190]:
```python
dataset = pd.DataFrame()
dataset["overall"] = df["overall"]
dataset["reviewText"] = df["reviewText"]
dataset["summary"] = df["summary"]
```

Our dataset has now the form

In [191]:
```python
dataset.iloc[0:5]
```

Out[191]:

	overall	reviewText	summary
0	5	I like the item pricing. My granddaughter want...	Magnetic board
1	4	Love the magnet easel... great for moving to d...	it works pretty good for moving to different a...
2	5	Both sides are magnetic. A real plus when you...	love this!
3	5	Bought one a few years ago for my daughter and...	Daughters love it
4	4	I have a stainless steel refrigerator therefor...	Great to have so he can play with his alphabet...

Data cleaning and text preprocessing

Text is just a sequence of words (or characters). When we deal with language modelling we are more concerned about the words as a whole, instead of just worrying about character-level depth of our text data. So we need to put text into a form that is more predictable and analyzable. Below we explore the basic steps of text preprocessing. These steps are needed for transferring text from human language to machine-readable format for further processing.

We start to apply this techniques to a single sentence, for visualizing how this affect the text. Then we apply the same techniques to all dataset.

Then we apply the same techniques to all dataset.

Remove non-letter characters and convert reviews to lower Case

It may be important to include some punctuations and numbers. However, for simplicity, we remove both of them.

In the following code, we substitute all the characters different from a-z and from A-Z to spaces. Then we convert any capital letters to lower case.

In [33]:
```
#Remove non-letter characters
review = dataset.iloc[0]["reviewText"]
print("Original review: ", review)

#Convert reviews to lower case
letter = re.sub("[^a-zA-Z]", " ", review)
letter = letter.lower()
print("Processed review: ",letter)
```

```
Original review:  I like the item pricing. My granddaughter wanted t
o mark on it but I wanted it just for the letters.
Processed review:  i like the item pricing  my granddaughter wanted
to mark on it but i wanted it just for the letters
```

Tokenization

Tokenization is the process of splitting a sentence or paragraph into a list of word.

We use the function provided by the Natural Language Toolkit(NLTK) library for doing so:

In [41]:
```
word = nltk.word_tokenize(letter)
print("Tokenized review: ", word)
```

```
Tokenized review:  ['i', 'like', 'the', 'item', 'pricing', 'my', 'gr
anddaughter', 'wanted', 'to', 'mark', 'on', 'it', 'but', 'i', 'wante
d', 'it', 'just', 'for', 'the', 'letters']
```

Remove stop-words

Stop words are generally the most common words in a language.Stop words consume many computational resources (memory space, search time) and do not add any semantic value to the text. So, they are filtered out before processing a text.

There is no single universal list of stop words for all language. We use one provided by the NLTK library.

In [40]:

```python
#Remove stop-words
print("List of english stop-words: \n")
print(stopwords.words("english"), "\n")
wordSW = [w for w in word if not w in set(stopwords.words("english
print("Original review: ", word, "\n")
print("Processed review: ", wordSW, "\n")
```

```
List of english stop-words:

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',
'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this
', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'w
ere', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', '
does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', '
because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', '
about', 'against', 'between', 'into', 'through', 'during', 'before',
'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'h
ere', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor',
'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'coul
dn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn
't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
ghtn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "
shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't
", 'won', "won't", 'wouldn', "wouldn't"]

Original review:  ['i', 'like', 'the', 'item', 'pricing', 'my', 'gra
nddaughter', 'wanted', 'to', 'mark', 'on', 'it', 'but', 'i', 'wanted
', 'it', 'just', 'for', 'the', 'letters']

Processed review:  ['like', 'item', 'pricing', 'granddaughter', 'wan
ted', 'mark', 'wanted', 'letters']
```

Stemming and Lemmization

It is useful to know the difference between these two.

- **Stemming**: Stemming algorithms work by cutting off the end of the word, and in some cases also the beginning while looking for the root.

  example) studying -> study; studied -> studi
- **Lemmatization**: Lemmatization is the process of converting the words of a sentence to its dictionary form.

  example) studying -> study, studied -> study.

  Lemmatization also discerns the meaning of the word by understanding the context of a passage.

  example) if a "meet" is used as a noun then it will print out a "meeting"; however, if it is used as a verb then it will print out "meet".

Usually, either one of them is chosen for text-analysis. I use stemming. In a future work can be interesting evaluate which is the best choice between these two methodologies with this given dataset and naive Bayes method.

There are three major Stemming alghorithms: Porter, Snowball and Lancaster. Lancaster is the most aggressive among the three and Porter is the least aggressive ("aggressive" means how much a working set of words are reduced). The more aggressive the algorithms, the faster it is. However, in some certain circumstances, it will hugely trim down your working set. Therefore, in this project I decide to use Snowball since it is slightly faster than Porter and does not trim down too much information as Lancaster does.

In [43]:
```
#Lemmization
snow = nltk.stem.SnowballStemmer("english")
stems = [snow.stem(w) for w in wordSW]
print("Original review: ", wordSW, "\n")
print("Processed review: ", stems, "\n")
```

```
Original review:  ['like', 'item', 'pricing', 'granddaughter', 'want
ed', 'mark', 'wanted', 'letters']

Processed review:  ['like', 'item', 'price', 'granddaught', 'want',
'mark', 'want', 'letter']
```

Clean all dataset

So far, we have cleaned only one datapoint. Now it's time to apply all the cleaning process to all the data. To make the code reusable, we create a simple function.

In [192]:
```
#Apply data cleaning and text preprocessing to all dataset
def cleaning(rawReview):
    #1. Remove non-letters
    letters = re.sub("[^a-zA-Z]", " ", rawReview)

    #2. Convert to lower case
    letters = letters.lower()

    #3. Tokenize
    tokens = nltk.word_tokenize(letters)

    #4. Convert the stopwords list to "set" data type
    stops = set(nltk.corpus.stopwords.words("english"))
    #5. Remove stop words
    words = [w for w in tokens if not w in stops]

    #6. Stemming
    words = [nltk.stem.SnowballStemmer("english").stem(w) for w ir

    #7. Join the words back into one string separated by space, ar
    return " ".join(words)
```

Then, we use apply() method to execute the above function on all the rows of our dataset. This is an expansive process, so, after the completion of computation, we save our processed dataset in an external file, to retrieve it when needed and save time.

In [193]:
```
#Apply data cleaning and text preprocessing to all dataset
#Add the processed data to the original data.
dataset["cleanReview"] = dataset["reviewText"].apply(cleaning)
dataset["cleanSummary"] = dataset["summary"].apply(cleaning)

#Save processed data set in order to retrieve it
dataset.to_pickle("./Dataset/processedToysAndGames.pkl")

```

In [198]:
```
#Retrieve precedent saved dataset
dataset = pd.read_pickle("./Dataset/processedToysAndGames.pkl")
dataset.head()
```

Out[198]:

	overall	reviewText	summary	cleanReview	cleanSummary
0	5	I like the item pricing. My granddaughter want...	Magnetic board	like item price granddaught want mark want letter	magnet board
1	4	Love the magnet easel... great for moving to d...	it works pretty good for moving to different a...	love magnet easel great move differ area wish ...	work pretti good move differ area
2	5	Both sides are magnetic. A real plus when you...	love this!	side magnet real plus entertain one child four...	love
3	5	Bought one a few years ago for my daughter and...	Daughters love it	bought one year ago daughter love still use to...	daughter love
4	4	I have a stainless steel refrigerator therefor...	Great to have so he can play with his alphabet...	stainless steel refriger therefor much space s...	great play alphabet

Data visualization

Data visualization is the graphic representation of data.

Word frequency

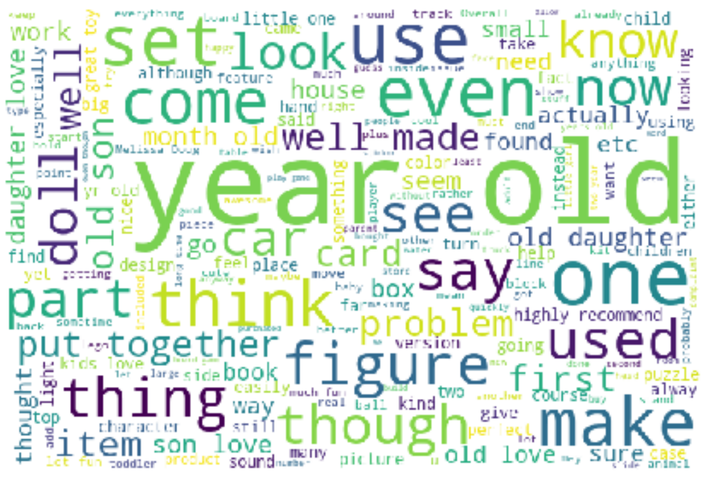
As a tool for visualization by using the frequency of words appeared in text, we use WordCloud. It can give you some general shape of what this text is about quickly and intuitively.

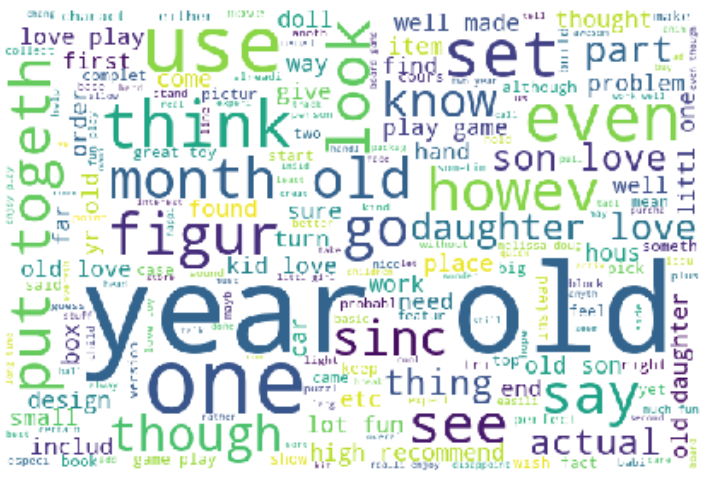
We define a function that do all the work and we apply it at the reviewText and cleanReview columns in order to better visualize the difference.

In [120]:
```
#WorldCloud
def cloud(data,backgroundcolor = 'white', width = 800, height = 60
    wordcloud = WordCloud(stopwords = STOPWORDS, background_color
                          width = width, height = height).generate(
    plt.figure(figsize = (10, 5))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
```

We print the world of cloud of the original and prcessed reviewText field in order to better understand the difference between the two.

In [121]:
```
cloud(' '.join(dataset["reviewText"]))
cloud(' '.join(dataset["cleanReview"]))
```





Distribution

Let's visualize the distribution of the length of the reviews. In order to obtain this, we first count the number of words of each review and add it to our dataset.

In [103]:
```
#Distribution
# We need to split each words in cleaned review and then count the
dataset["reviewLenght"] = dataset["cleanReview"].apply(lambda x: l
dataset["summaryLenght"] = dataset["cleanSummary"].apply(lambda x:
```

Now dataset looks like

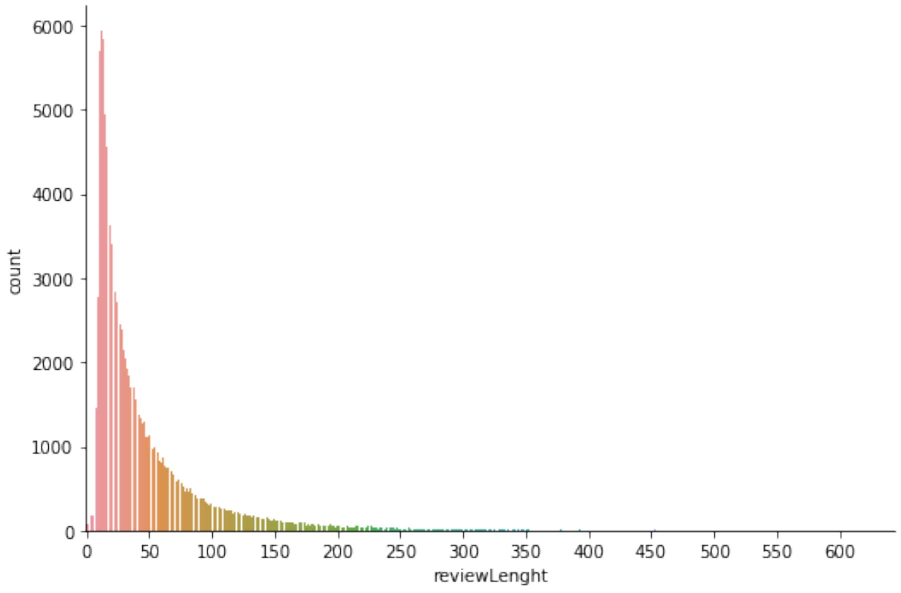
In [84]:
```
dataset.head()
```

Out[84]:

	overall	reviewText	summary	cleanReview	cleanSummary	reviewLenght	summaryLen
0	5	I like the item pricing. My granddaughter want...	Magnetic board	like item price granddaught want mark want letter	magnet board	8	
1	4	Love the magnet easel... great for moving to d...	it works pretty good for moving to different a...	love magnet easel great move differ area wish ...	work pretti good move differ area	14	
2	5	Both sides are magnetic. A real plus when you...	love this!	side magnet real plus entertain one child four...	love	40	
3	5	Bought one a few years ago for my daughter and...	Daughters love it	bought one year ago daughter love still use to...	daughter love	14	
4	4	I have a stainless steel refrigerator therefor...	Great to have so he can play with his alphabet...	stainless steel refriger therefor much space s...	great play alphabet	19	

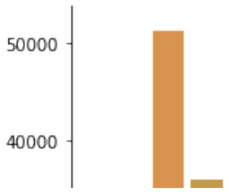
Then, we use the catplot() function of seaborn library to visualize the distribution of the length of the reviews:

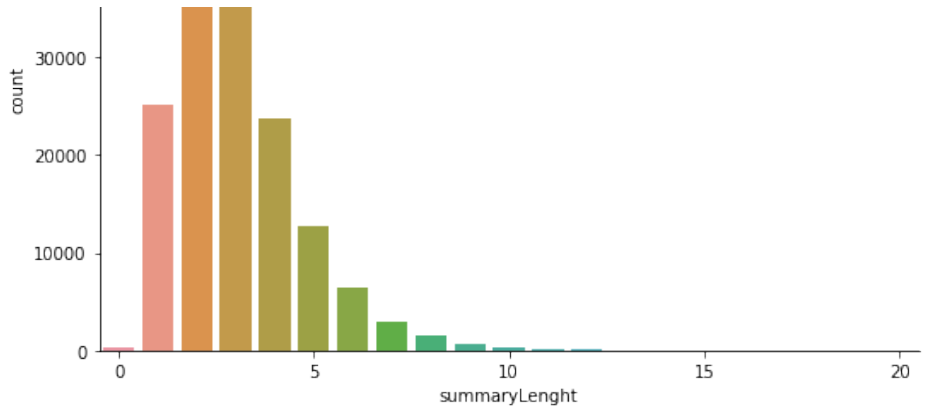
In [181]:
```
sns.catplot(x='reviewLenght',kind='count',data=dataset,orient="h",
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%d'))
ax.xaxis.set_major_locator(ticker.MultipleLocator(base=50))
plt.show()
print("Median of review lenght: ", dataset["reviewLenght"].median(
```



```
Median of review lenght:  29.0
```

In [183]:
```
sns.catplot(x='summaryLenght',kind='count',data=dataset,orient="h'
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%d'))
ax.xaxis.set_major_locator(ticker.MultipleLocator(base=5))
plt.show()
print("Median of review lenght: ", dataset["summaryLenght"].mediar
```





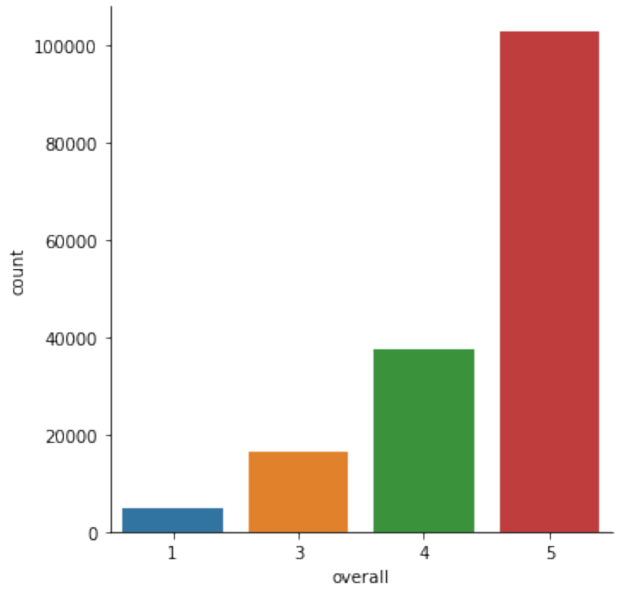
```
Median of review lenght:  3.0
```

Notice how the reviews in the summary column are noticeably shorter than the ones on the summaryText column.

The last step is to visualize the distribution of the overall score:

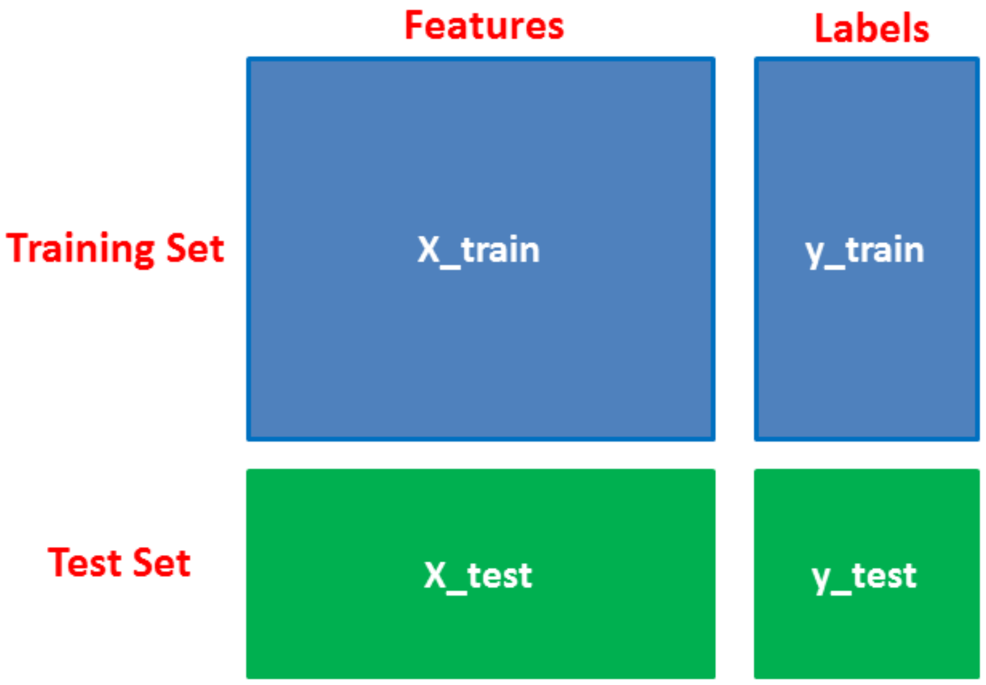
In [184]:
```
dataset['overall'].value_counts()
sns.catplot(x='overall',kind='count',data=dataset,orient="h")
```

Out[184]:
```
<seaborn.axisgrid.FacetGrid at 0x1b8b833210>
```



Split the dataset

Usually, dataset is split into random train and test subsets. We use the train_test_split() function from scikit-learn library to do so.



In [200]:
```
#Create validation and training set
xTrain, xTest, yTrain, yTest = train_test_split(dataset["cleanRevi
                                                dataset["overall'
                                                 test_size=0.2,
                                                  random_state=1)
```

Feature extraction and Bag Of Words

Even though we cleaned the data with many steps, we still have one more step to create machine learning-friendly input. One common approach is called a Bag of Words. It is simply the matrix that counts how many each word appears in documents,(or only if the word appears in document) disregarding grammar and word order. In order to do that, we use CountVectorizer() method in sklearn library.

The CountVectorizer has its own built-in preprocessor (text cleaner) and tokenizer (word splitter). In this case, we deal with documents that have already been preprocessed. We need to disable the built-in preprocessor.

In [194]:
```
def dummy(word):
    return word
```

In [201]:
```
#Build vectorizer
vect = []
vect.append(CountVectorizer(binary = True, preprocessor = dummy))
vect.append(CountVectorizer(preprocessor = dummy))
```

We built two vectorizer: one for Bernoulli method and one for Multinomial method. In fact, Bernoulli expect a binary bag of words, while Multinomial expect a word counting frequency bag of words.

In [202]:
```
#Create bag of words representation for bernoulli model
xTrainBern = vect[0].fit_transform(xTrain) #Learn the vocabulary c
xTestBern = vect[0].transform(xTest) #Transform documents to docum

#Create bag of words representation for binomial model
xTrainBin = vect[1].fit_transform(xTrain)
```

```
xTestBin = vect[1].transform(xTest)

print("Number of words in train set vocabulary:", len(vect[0].get_
```

```
Number of words in train set vocabulary: 47807
```

Shed some light on the bag of words representation returned from the CountVectorizer class

The following function create a dataframe from a word matrix return by CountVectorizer:

In [203]:
```
def wm2df(wordMatrix, featureNames):
    # create an index for each row
    docNames = ['Doc{:d}'.format(idx) for idx, _ in enumerate(word
    dataFrame = pd.DataFrame(data=wordMatrix.toarray(), index= doc
                    columns=featureNames)
    return(dataFrame)
```

In [204]:
```
#set of documents
documents = ['The quick brown fox.','The the Jumps over the lazy d
#instantiate the vectorizer object
cvec = CountVectorizer(preprocessor = dummy)
#convert the documents into a document-term matrix
wordMatrix = cvec.fit_transform(documents)
#retrieve the terms found in the corpora
featureNames = cvec.get_feature_names()
#create a dataframe from the matrix
wm2df(wordMatrix, featureNames)
```

Out[204]:

	Jumps	The	brown	dog	fox	lazy	over	quick	the
Doc0	0	1	1	0	1	0	0	1	0
Doc1	1	1	0	1	0	1	1	0	2

Notice how CountVectorizer can do most of the preprocessing work done above by hand (see documentation for more options).

Modelling

As you can see from the matrix above, text data usually is very sparse and has a high dimensionality. As described in the introduction, we use Bernoulli and Multinomial naive Bayes method for classifying our review.

In [205]:
```
#Build models
models = []
models.append(BernoulliNB())
models.append(MultinomialNB())

#Train models
models[0].fit(xTrainBern, yTrain)
models[1].fit(xTrainBin, yTrain)

#Make class predictions
yPredBern = models[0].predict(xTestBern)
yPredBin = models[1].predict(xTestBin)
```

We used BernoulliNB and MultinomialNB method provided by the Scikit-learn library for instantiate and train our models.

Then we perform a series of predictions using the test sets defined above.

Analysis

Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

It works well only if there are equal number of samples belonging to each class. For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A. When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the test accuracy would drop down to 60%. Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

We use the accuracy_score() function provide by scikit-learn for calculate it:

In [206]:
```
print('Bernoulli Accuracy:', metrics.accuracy_score(yTest, yPredBe
print('Multinomial Accuracy:', metrics.accuracy_score(yTest, yPrec
```

```
Bernoulli Accuracy: 0.5884844868735084
Multinomial Accuracy: 0.6524463007159904
```

We now train our models on non-preprocessed reviews and evaluate how the processing work influenced the accuracy of our models

First, we define a simple function to avoid code repetition and perform a series of accuracy measurements on both processed and non-processed text:

In [208]:
```
def analysis(feature, target, maxFeatures = None, minDf = 1,
                             ngramRange = (1,1)):
    #Build vectorizer
    vect = []
    vect.append(CountVectorizer(binary = True, max_features = maxF
    vect.append(CountVectorizer(max_features = maxFeatures, min_df

    xTrain, xTest, yTrain, yTest = train_test_split(feature, targe

    #Create bag of words representation for bernoulli model
    xTrainBern = vect[0].fit_transform(xTrain) #Learn the vocabula
    xTestBern = vect[0].transform(xTest) #Transform documents to c

    #Create bag of words representation for binomial model
    xTrainBin = vect[1].fit_transform(xTrain)
    xTestBin = vect[1].transform(xTest)

    print("Number of words in vocabulary:", len(vect[0].get_featur
    #Build models
    models = []
    models.append(BernoulliNB())
    models.append(MultinomialNB())

    #Train models
    models[0].fit(xTrainBern, yTrain)
    models[1].fit(xTrainBin, yTrain)

    #Make class predictions
    yPredBern = models[0].predict(xTestBern)
    yPredBin = models[1].predict(xTestBin)

    print('Bernoulli Accuracy:', metrics.accuracy_score(yTest, yPr
    print('Multinomial Accuracy:', metrics.accuracy_score(yTest, y
```

In [209]:
```
analysis(dataset['reviewText'],dataset['overall'])
```

```
Number of words in vocabulary: 95039

Bernoulli Accuracy: 0.5792064439140812
Multinomial Accuracy: 0.6510739856801909
```

Clearly, even if the number of feature is smaller (47807 vs 95039), the preprocessing work had only a small impact on the accuracy of both model.

What if we drastically reduce the number of features?

In [441]:
```
analysis(dataset['cleanReview'],dataset['overall'], maxFeatures =
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.6014569125852449
Multinomial Accuracy: 0.6255114693118413
```

In [453]:
```
analysis(dataset['reviewText'],dataset['overall'], maxFeatures = 5
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.547255369928401
Multinomial Accuracy: 0.6060560859188544
```

What if we use our classifiers on shorter reviews? Let's try to use the summary field instead of reviewText field:

In [362]:
```
analysis(dataset['cleanSummary'],dataset['overall'])
```

```
Number of words in vocabulary: 12009

Bernoulli Accuracy: 0.645346062052506
Multinomial Accuracy: 0.6481801909307876
```

In [361]:
```
analysis(dataset['summary'],dataset['overall'])
```

```
Number of words in vocabulary: 17224

Bernoulli Accuracy: 0.6506563245823389
Multinomial Accuracy: 0.6587410501193317
```

In this case, the performance of the Bernoulli model is considerably increased.

We also notice that the preprocessing work has had a negative impact on the accuracy scores.

Moreover, in case the reviews are short and the number of elements in the vocabulary are few, the Bernoulli model outperform for the first time the Multinomial model:

In [364]:
```python
analysis(dataset['cleanSummary'],dataset['overall'], maxFeatures =
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.6158412887828162
Multinomial Accuracy: 0.6131861575178997
```

In [363]:
```python
analysis(dataset['summary'],dataset['overall'], maxFeatures = 50)
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.63645584725537
Multinomial Accuracy: 0.634307875894988
```

Sentiment analysis: predict only if the reviews are positive or negative

Suppose we now want to evaluate only if a review is positive or negative, rather than the overall score.

We consider a positive review if it has a score strictly higher than 2, negative if it has a score strictly less than 2. We ignore the reviews with a score equal to 2.

We then add a new "sentiment" attribute to our dataset.

In [212]:
```python
def sentiment(x):
  if x == 2:
    return np.nan
  if x<2:
    return 0
  if x>2:
    return 1

#Add sentiment of each review
dataset["sentiment"] = dataset["overall"].apply(sentiment)

#Delete review with score equal to 2
dataset = dataset.dropna()

print("New dataset look like: \n")
dataset.iloc[:2]
```

```
New dataset look like:
```

Out[212]:

	overall	reviewText	summary	cleanReview	cleanSummary	sentiment
0	5	I like the item pricing. My granddaughter want...	Magnetic board	like item price granddaught want mark want letter	magnet board	1.0
1	4	Love the magnet easel... great for moving to d...	it works pretty good for moving to different a...	love magnet easel great move differ area wish ...	work pretti good move differ area	1.0

We repeat all the tests carried out previously and observe, as is natural, how the best accuracy increase and how some result differ:

In [368]:
```
analysis(dataset['cleanReview'],dataset['sentiment'])
```

```
Number of words in vocabulary: 46798

Bernoulli Accuracy: 0.957470551766894
Multinomial Accuracy: 0.9751394916305022
```

In [369]:
```
analysis(dataset['reviewText'],dataset['sentiment'])
```

```
Number of words in vocabulary: 72064

Bernoulli Accuracy: 0.9484500929944203
Multinomial Accuracy: 0.9758524488530688
```

In [371]:
```
analysis(dataset['cleanReview'],dataset['sentiment'], maxFeatures
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.971264724116553
Multinomial Accuracy: 0.9590514569125852
```

In [372]:
```
analysis(dataset['reviewText'],dataset['sentiment'], maxFeatures =
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.9614693118412895
Multinomial Accuracy: 0.9523248605083695
```

In this case results differ: notice how Benoulli method outperform the Multinomial one when the number of features is low, even if the review is sufficiently long.

In [373]:
```
analysis(dataset['summary'],dataset['sentiment'])
```

```
Number of words in vocabulary: 16936

Bernoulli Accuracy: 0.971264724116553
Multinomial Accuracy: 0.9731556106633602
```

In [377]:
```
analysis(dataset['cleanSummary'],dataset['sentiment'])
```

```
Number of words in vocabulary: 11821

Bernoulli Accuracy: 0.9713887166769993
Multinomial Accuracy: 0.9730006199628022
```

In [375]:
```
analysis(dataset['summary'],dataset['sentiment'], maxFeatures = 5
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.9712957222566646
Multinomial Accuracy: 0.9690638561686299
```

In [376]:
```
analysis(dataset['cleanSummary'],dataset['sentiment'], maxFeature
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.971264724116553
Multinomial Accuracy: 0.9705207687538747
```

Here we have the same result: Benoulli method outperform the Multinomial one when the number of features is low, and the review is sufficiently short.

Playing with parameters

Below we do some test with the CountVector function, trying to improve accuracy score with overall label.

We set min_df = 2 and ngram_range = (1,3): min_df = 2 means that, in order to include the vocabulary in the matrix, one word must appear in at least two documents. ngram_range means that we cut one sentence by number of ngram. Let's say we have one sentence, I am a boy. If we cut the sentence by digram (ngram=2) then the sentence would be cut like this ["I am","am a", "a boy"]. The result of accuracy can be highly dependent on parameters.

In [463]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 2)
```

```
Number of words in vocabulary: 22591

Bernoulli Accuracy: 0.607997520148791
Multinomial Accuracy: 0.6719466831990081
```

In [467]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 3)
```

```
Number of words in vocabulary: 17678

Bernoulli Accuracy: 0.6088964662120273
Multinomial Accuracy: 0.6709237445753254
```

In [464]:
```
analysis(dataset['cleanReview'],dataset['overall'], ngramRange = (
```

```
Number of words in vocabulary: 6554026

Bernoulli Accuracy: 0.6351828890266584
Multinomial Accuracy: 0.6398946063236206
```

In [468]:
```
analysis(dataset['cleanReview'],dataset['overall'], ngramRange = (
```

```
Number of words in vocabulary: 12080894

Bernoulli Accuracy: 0.634903905765654
Multinomial Accuracy: 0.6384686918784873
```

In [460]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 2, ngr
```

```
Number of words in vocabulary: 1027090

Bernoulli Accuracy: 0.6311531308121513
Multinomial Accuracy: 0.6775573465592064
```

In [461]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 2, ngr
```

```
Number of words in vocabulary: 1164417

Bernoulli Accuracy: 0.6317420954742715
Multinomial Accuracy: 0.6756354618722876
```

In [475]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 2, ngr
```

```
Number of words in vocabulary: 50

Bernoulli Accuracy: 0.6004959702417855
Multinomial Accuracy: 0.6245815251084935
```

In [462]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 3, ngr
```
```
Number of words in vocabulary: 524095

Bernoulli Accuracy: 0.6253254804711718
Multinomial Accuracy: 0.6831990080595164
```

In [469]:
```
analysis(dataset['cleanReview'],dataset['overall'], minDf = 3, ngr
```
```
Number of words in vocabulary: 562252

Bernoulli Accuracy: 0.6252324860508369
Multinomial Accuracy: 0.6810601363918165
```

In [476]:
```
analysis(dataset['cleanSummary'],dataset['overall'], minDf = 2)
```
```
Number of words in vocabulary: 6749

Bernoulli Accuracy: 0.6697148171109734
Multinomial Accuracy: 0.6735275883446993
```

In [490]:
```
analysis(dataset['cleanSummary'],dataset['overall'], minDf = 5)
```
```
Number of words in vocabulary: 3950

Bernoulli Accuracy: 0.6718226906385617
Multinomial Accuracy: 0.6736205827650341
```

In [482]:
```
analysis(dataset['cleanSummary'],dataset['overall'],  ngramRange =
```
```
Number of words in vocabulary: 233460

Bernoulli Accuracy: 0.6442653440793552
Multinomial Accuracy: 0.675759454432734
```

In [483]:
```
analysis(dataset['cleanSummary'],dataset['overall'],  ngramRange =
```
```
Number of words in vocabulary: 306440

Bernoulli Accuracy: 0.6401425914445134
Multinomial Accuracy: 0.6745195288282703
```

In [485]:
```
analysis(dataset['cleanSummary'],dataset['overall'], minDf = 2, ng
```
```
Number of words in vocabulary: 41218

Bernoulli Accuracy: 0.6704587724736516
Multinomial Accuracy: 0.6769993800371977
```

In [486]:
```
analysis(dataset['cleanSummary'],dataset['overall'], minDf = 2, ng
```
```
Number of words in vocabulary: 43493

Bernoulli Accuracy: 0.6703657780533168
Multinomial Accuracy: 0.6771853688778673
```

In [487]:
```
analysis(dataset['cleanSummary'],dataset['overall'], minDf = 3, ng
```
```
Number of words in vocabulary: 22129

Bernoulli Accuracy: 0.670117792932424
Multinomial Accuracy: 0.6744885306881587
```

```
In [488]: analysis(dataset['cleanSummary'],dataset['overall'], minDf = 3, ng
```

```
Number of words in vocabulary: 22833

Bernoulli Accuracy: 0.6697768133911965
Multinomial Accuracy: 0.6744265344079355
```

Summing up

1. **For the Bernoulli model, we obtain on this dataset an improvement of almost ten points compared to the original test (the one performed at the beginning of this chapter with only the cleanReview column and the overall target) as regards the accuracy using short reviews and playing with ngrams, and including in the vocabulary only most frequent words.**
2. **For the Multinomial model, we obtain on this dataset an improvement of three points compared to the original test (the one performed at the beginning of this chapter with only the cleanReview column and the overall target) as regards the accuracy by using both long and short reviews and playing with ngrams, and including in the vocabulary only most frequent words.**
3. **The above test provide evidence that the multinomial model tends to outperform the multi-variate Bernoulli model in all the test cases, except the one where the reviews are short and the number of word in vocabulary is low.**
4. **When the number of word in vocabulary is restricted we note a small improvement in Bernoulli model and a small deterioration in Multinomial model, in agreement with what has been done in** *Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification* **: empirical comparisons provide evidence that the multinomial model tends to outperform the multi-variate Bernoulli model if the vocabulary size is relatively large**
5. **However, they also show that in text classification, large differences in performance can be attributed to the choices of stop word removal, stemming, token-length, ngrams range, number of features, choice of the words to be included in the vocabulary, in accordance with the most complete tests performed in** *Lawrence M Rudner and Tahung Liang. Automated essay scoring using bayes' theorem.*

It's evidente that the choice between a multi-variate Bernoulli or Multinomial model for text classification should precede comparative studies including different combinations of the parameters discussed above.

These first results will find more evidence when we will compare the learning curves of the two methods.

Learning curve

Let's say we have some data and split it into a training set and validation set. We take one single instance from the training set and use it to estimate a model. Then we measure the model's error on the validation set and on that single training instance. The error on the training instance will be 0, since it's quite easy to perfectly fit a single data point. The error on the validation set, however, will be very large.

That's because the model is built around a single instance, and it almost certainly won't be able to generalize accurately on data that hasn't seen before. Now let's say that instead of one training instance, we take ten and repeat the error measurements. Then we take fifty, one hundred, five hundred, until we use our entire training set. The error scores will vary more or less as we change the training set. We thus have two error scores to monitor: one for the validation set, and one for the training sets. If we plot the evolution of the two error scores as training sets change, we end up with two curves. These are called learning curves. A learning curve shows how error changes as the training set size increases.

To draw the learning curve we use the function learning_curve provided by the Sci-kit library: estimator is the model with which we wish to perform the tests, X the complete set of features, y the complete set of targets, train_sizes is a list containing training set sizes we want to use for generating the learning curves. **As default, the function split X in training set and test set, with an 80:20 ratio, so the maximum value we could enter in this field will be 80% of the overall size of X. An important thing to be aware of is that the function is using k-cross-validation, so there will be five splits. For each split, an estimator is trained for every training set size specified (where k is the number of folds used for cross-validation).**

This also affects the values returned by the learning_curve function: since k tests are performed for each dimension of the train set (a list of numeric values specified in the train_sizes parameter) , the function returns as a value a table with shape (#train sizes * k).

To print the learning curve we need only one value per dimension we calculate the average of the 5 test values for each.

All these operations, together with the preparation of the dictionaries, are carried out in the two underlying functions.

In [215]:

```python
def dataPlotPreparation( X, y, maxFeatures = None, minDf = 1,ngram
    #Build vectorizer
    vect = []
    vect.append(CountVectorizer(binary = True, max_features= maxFe
    vect.append(CountVectorizer( max_features= maxFeatures, min_df

    #Convert in bag of words representation the entire dataset
    XBern = vect[0].fit_transform(X)
    XBin = vect[1].fit_transform(X)
    y = y

    print("Number of words in vocabulary :", len(vect[0].get_featu

    return XBern, XBin, y



```

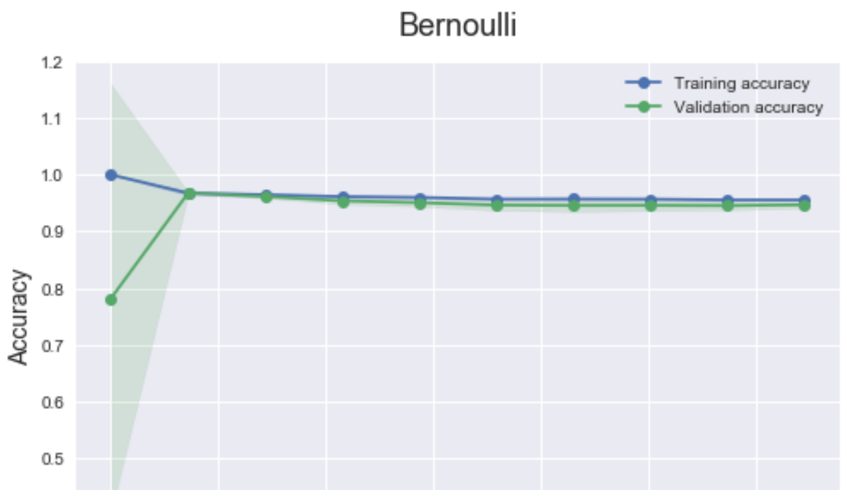
In [216]:

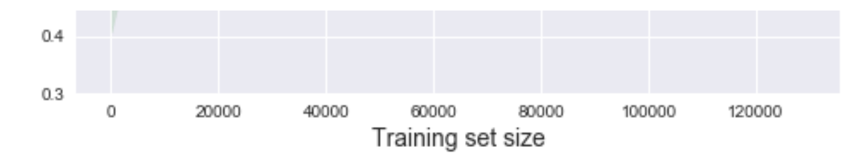
```python
#Generate learning curve
def plotLearningCurve(estimator, X, y, trainSizes, title):
    #generate learning curve for bernoulli model
    trainSizes, trainScores, validationScores = learning_curve(est
                                                               X = X,
                                                                y = y
                                                                 trai
                                                                   sc
                                                                    r


    #Print scores
    #print('Training scores:\n\n', trainScores)
    #print('\nValidation scores:\n\n', validationScores)

    #Calculate the mean
    trainScoresMean = trainScores.mean(axis = 1)
    validationScoresMean = validationScores.mean(axis = 1)
    #print('Mean training scores\n\n', pd.Series(trainScoresMean,
    #print('\nMean validation scores\n\n',pd.Series(validationScor
    trainScoresStd = np.std(trainScores, axis=1)
    validationScoresStd = np.std(validationScores, axis=1)

    #Plot
    plt.style.use('seaborn')
    plt.plot(trainSizes, trainScoresMean, 'o-', label = 'Training
    plt.plot(trainSizes, validationScoresMean, 'o-', label = 'Vali
    plt.fill_between(trainSizes, trainScoresMean - trainScoresStd,
                     trainScoresMean + trainScoresStd, alpha=0.1,
                     color="b")
    plt.fill_between(trainSizes, validationScoresMean - validatior
                     validationScoresMean + validationScoresStd, a
    plt.ylabel('Accuracy', fontsize = 14)
    plt.xlabel('Training set size', fontsize = 14)
    plt.title(title, fontsize = 18, y = 1.03)
    plt.legend()
    plt.ylim(0.3,1.2)


```

As an example, we plot the learning curve of **both model** taking as feature vector the **cleanReview** columns and as target vector the **sentiment** and the **overall** column:

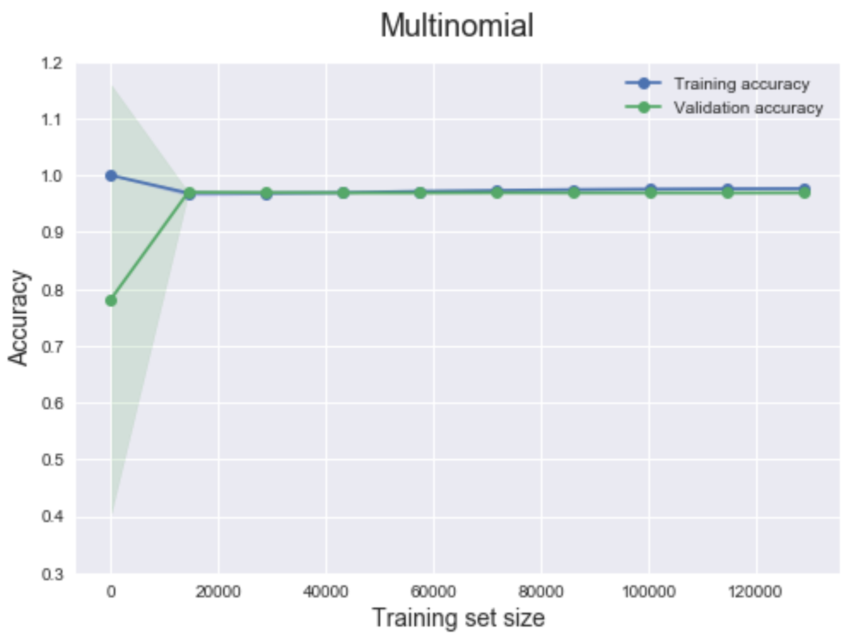
In [219]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanReview"], data

plotLearningCurve(estimator = models[0],
                  X = XBern,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Bernoulli")
```

```
Number of words in vocabulary : 52421
```

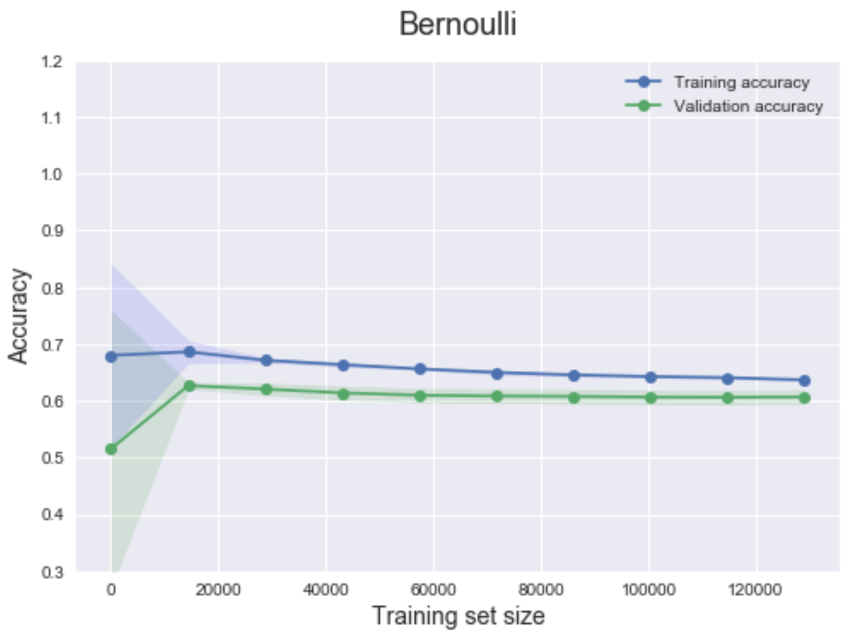




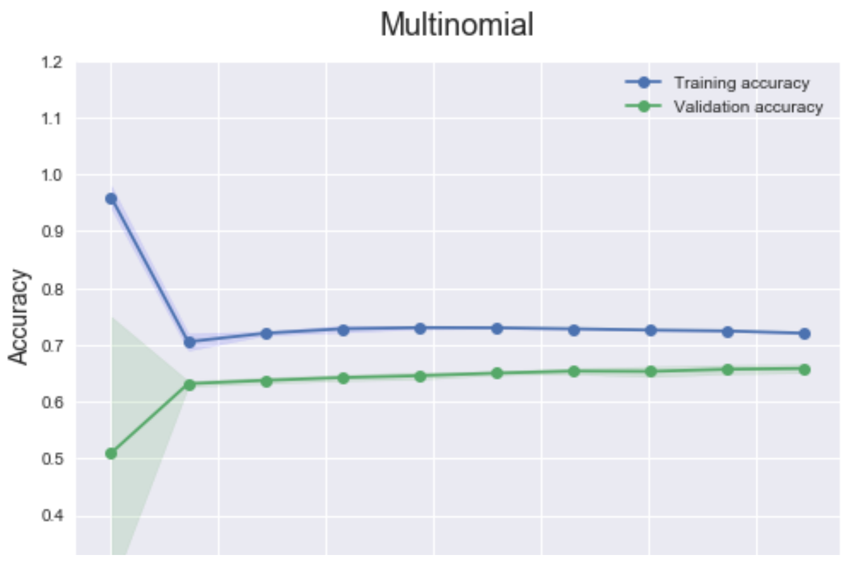
In [420]:
```
plotLearningCurve(estimator = models[1],
                  X = XBin,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Multinomial")
```



In [492]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanReview"], data

plotLearningCurve(estimator = models[0],
                  X = XBern,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Bernoulli")
```

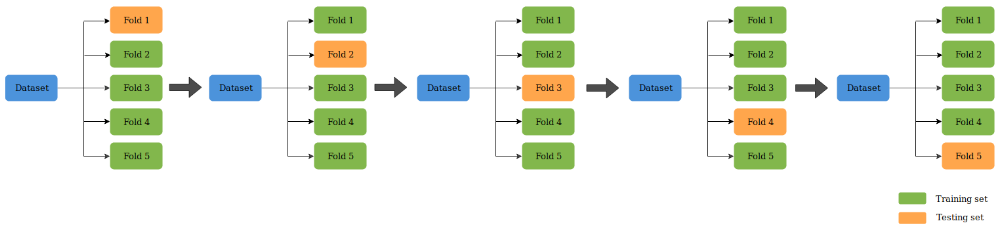


In [493]:
```
plotLearningCurve(estimator = models[1],
                  X = XBin,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Multinomial")
```



A note on K-fold cross validation

Learning_curve() use the k-fold cross validation methodology for performing test on training set. K-Fold CV is where a given data set is split into a K number of sections/folds where each fold is used as a testing set at some point. Lets take the scenario of 5-Fold cross validation(K=5). Here, the data set is split into 5 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.



A note on Laplace smoothing

Additive smoothing, also called Laplace smoothing, or Lidstone smoothing, is a technique used to smooth categorical data.

Given an observation x = (x1, …, xd) from a multinomial distribution with N trials and parameter vector θ = (θ1, …, θd), a "smoothed" version of the data gives the estimator:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

where the pseudocount α > 0 is the smoothing parameter (α = 0 corresponds to no smoothing).

BernoulliNB() and MultinomialNB() implementation of Naive Bayes method in Scikit-learn set the value of a to 1 by default.

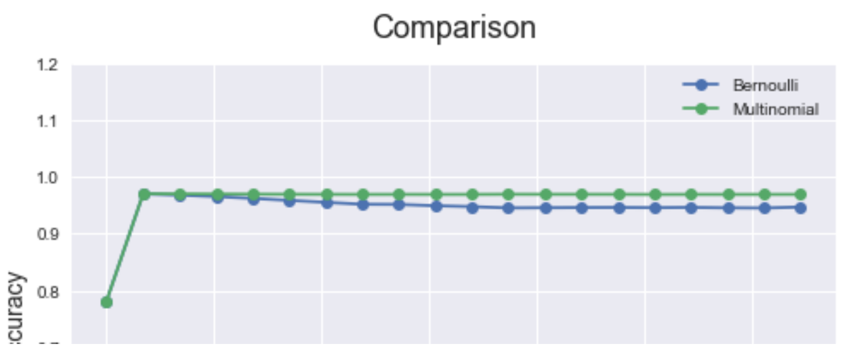
A comparison between the Bernoulli and the Multinomial model

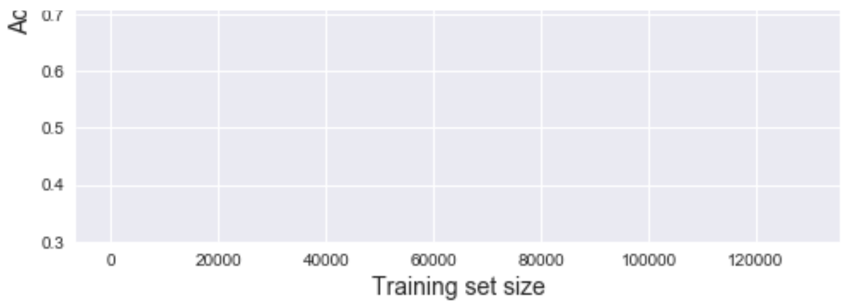
To better compare the two models, we can compare their accuracy validation curve on all previously defined tests, taking as a reference the cleanReview column for the features, and both sentiment and overall column for the target.

In [221]:
```
#Generate validation curve
def plotValidationCurve(estimator, X, y, trainSizes, title):
    #generate learning curve for bernoulli model
    trainSizes, trainScores, validationScores = learning_curve(est
                                                            X = X,
                                                             y = y
                                                              trai
                                                                sc
                                                                 r


    #Print scores
    #print('Training scores:\n\n', trainScores)
    #print('\nValidation scores:\n\n', validationScores)

    #Calculate the mean
    trainScoresMean = trainScores.mean(axis = 1)
    validationScoresMean = validationScores.mean(axis = 1)
    #print('Mean training scores\n\n', pd.Series(trainScoresMean,
    #print('\nMean validation scores\n\n',pd.Series(validationScor
    trainScoresStd = np.std(trainScores, axis=1)
    validationScoresStd = np.std(validationScores, axis=1)

    #Plot
    plt.style.use('seaborn')
    plt.plot(trainSizes, validationScoresMean, 'o-', label = title

    plt.ylabel('Accuracy', fontsize = 14)
    plt.xlabel('Training set size', fontsize = 14)
    plt.title("Comparison", fontsize = 18, y = 1.03)
    plt.legend()
    plt.ylim(0.3,1.2)
```

In [223]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanReview"], data

plotValidationCurve(estimator = models[0],
                  X = XBin,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Bernoulli")
plotValidationCurve(estimator = models[1],
                  X = XBin,
                   y = y,
                    trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Multinomial")
```

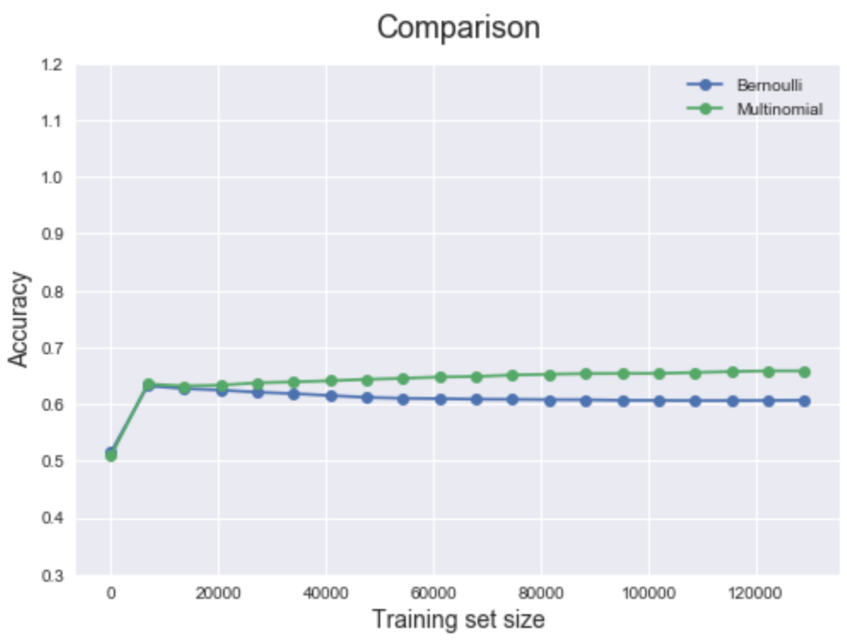
```
Number of words in vocabulary : 52421
```





In [224]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanReview"], data

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

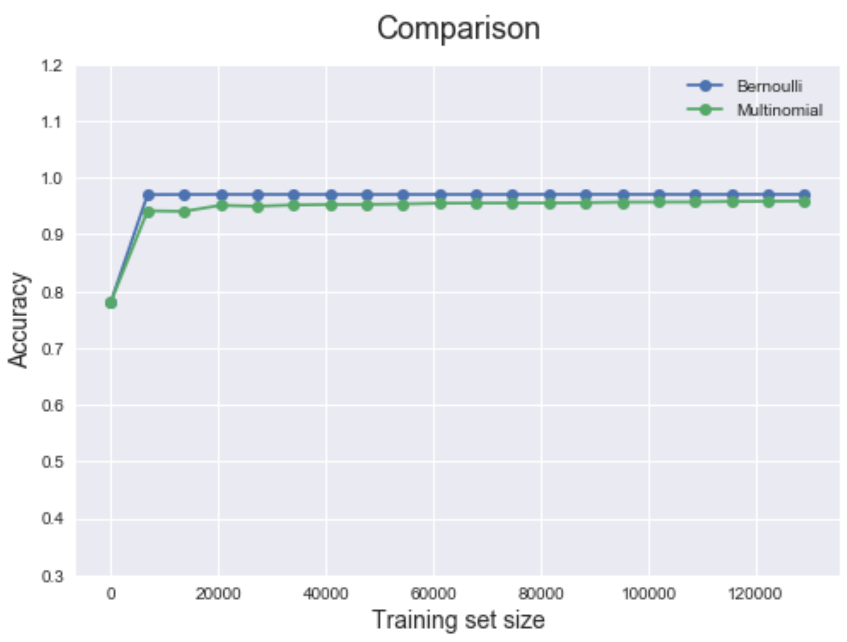
```
Number of words in vocabulary : 52421
```



In [225]:

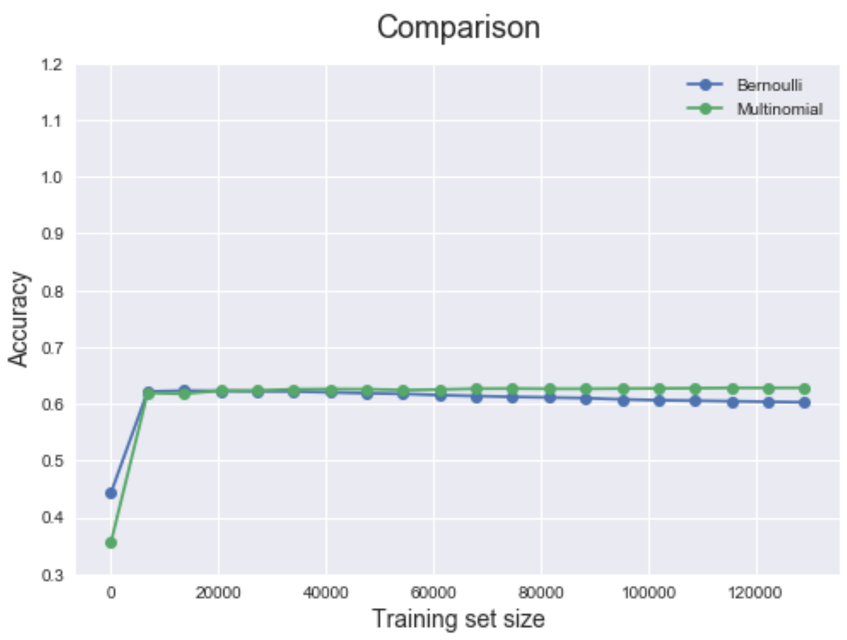
```
XBern, XBin, y = dataPlotPreparation( dataset['cleanReview'],datas

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

```
Number of words in vocabulary : 50
```



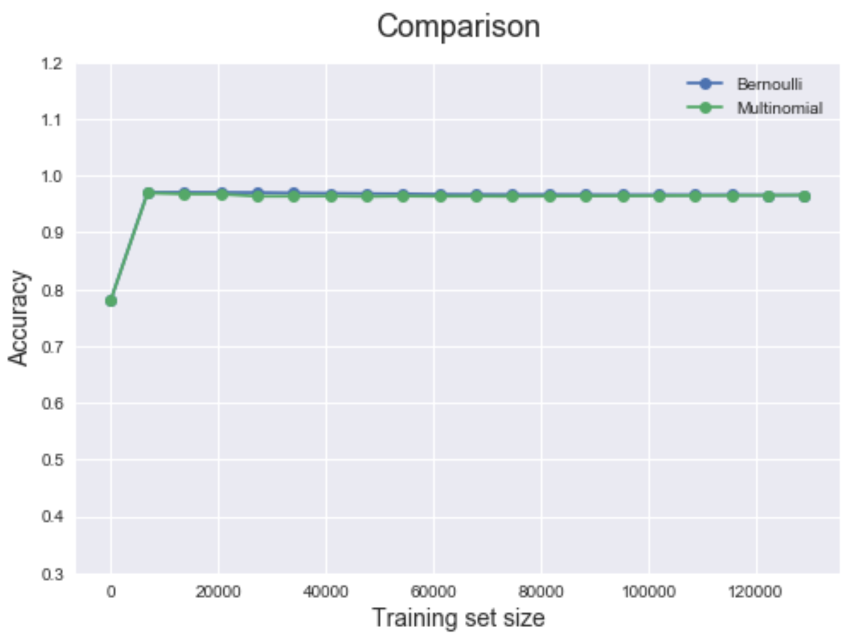
In [226]:
```
XBern, XBin, y = dataPlotPreparation( dataset['cleanReview'],datas

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

```
Number of words in vocabulary : 50
```



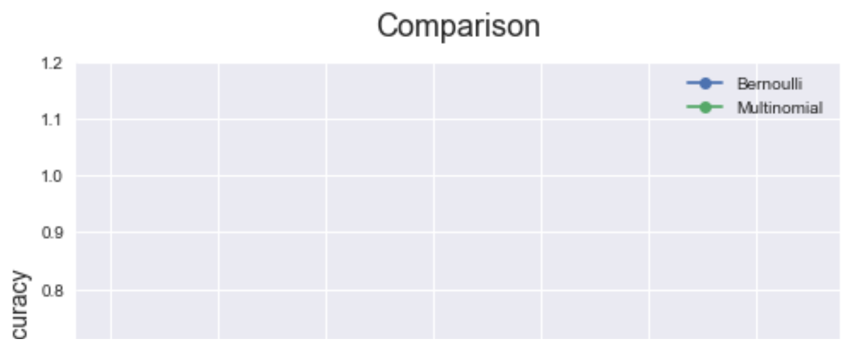
In [227]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanSummary"], dat

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

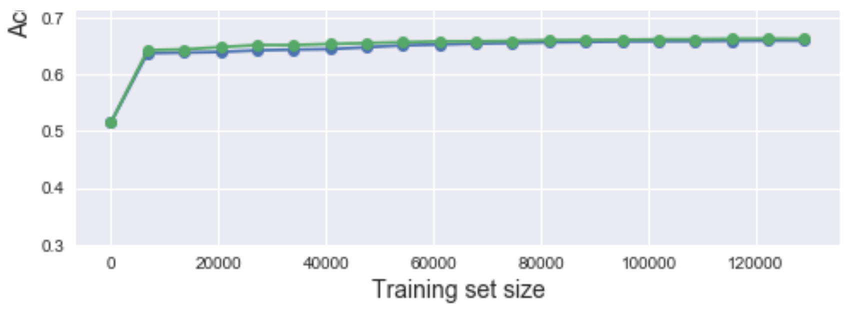
```
Number of words in vocabulary : 12977
```



In [228]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanSummary"], dat

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

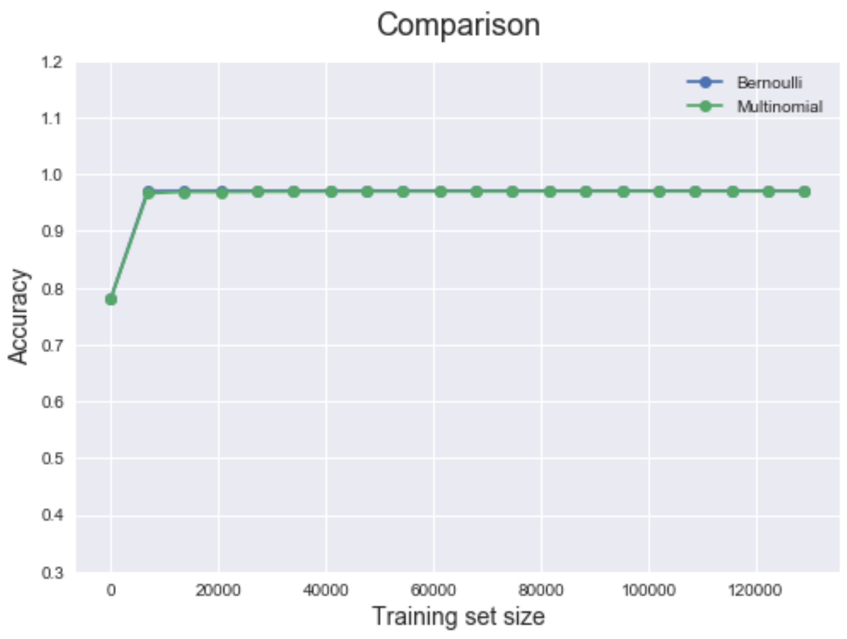
```
Number of words in vocabulary : 12977
```





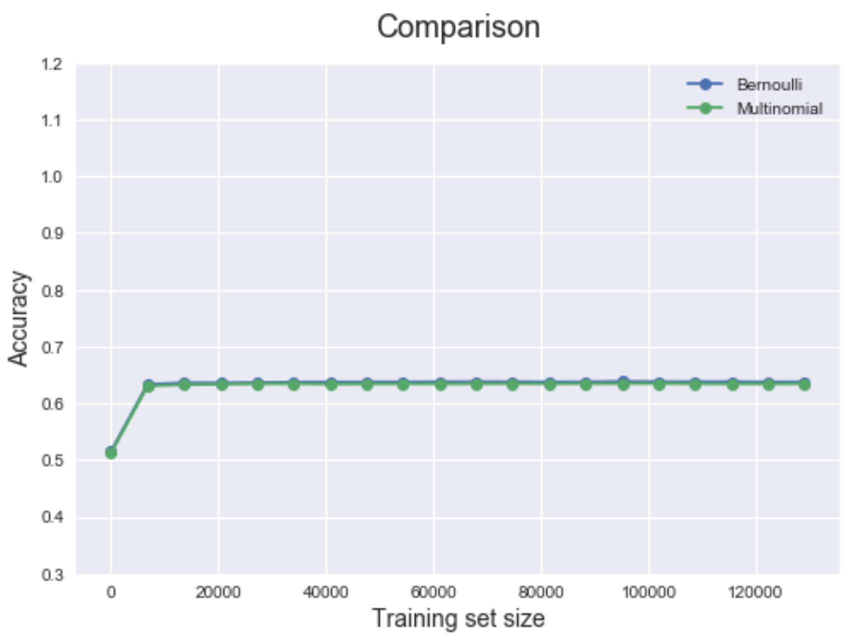
In [229]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanSummary"], dat

plotValidationCurve(estimator = models[0],
                   X = XBin,
                    y = y,
                     trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Bernoulli")
plotValidationCurve(estimator = models[1],
                   X = XBin,
                    y = y,
                     trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 50
```



In [230]:
```
XBern, XBin, y = dataPlotPreparation( dataset["cleanSummary"], dat

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

```
Number of words in vocabulary : 50
```



Summing up

1. **The multinomial model always behaves generally better than the Bernoulli model. The difference between the two increases with the size of the training set.**
2. **When the number of fetures (words in the dictionary) is reduced, the bernoulli has slightly higher performance than the multinomial, but the gap between the two is reduced as the size of the training set increases.**
3. **From the previous two we can deduce that the multinomial model benefits from larger dimensions of the training set, while the bernoulli model does not seem to show improvements from this point of view.**
4. **When the number of targets (overall score or sentiment score) is high, the multinomial is always better than the bernoulli. This is made evident by the graph in which at reduced number of fetures (where the bernoulli model normally behaves better) is accompanied a greater number of targets (the overall field is used): in this case the multinomial is able to adapt his self better than the bernoulli model and overcome the latter slightly.**
5. **When the length of reviews is short, both models generally offer the same performance. However, even when few words are available in the dictionary, the bernoulli seems to have a slight advantage over the multinomial.**

A comparison between the Bernoulli and the Multinomial model - other dataset

I will use the **Digital Music** subset and the **Grocery and Gourmet** subset available from http://jmcauley.ucsd.edu/data/amazon/ (http://jmcauley.ucsd.edu/data/amazon/) .

First, to make the result comparable with the first Toys and Games dataset, i will do the same preprocessing works for both dataset.

In [26]:
```
#Load dataset in a panda DataFrame
df = pd.read_json('./Dataset/Digital_Music_5.json',lines=True)

digitalMusicDs = pd.DataFrame()
digitalMusicDs["overall"] = df["overall"]
digitalMusicDs["reviewText"] = df["reviewText"]
digitalMusicDs["summary"] = df["summary"]

#Apply data cleaning and text preprocessing to all dataset
#Add the processed data to the original data.
digitalMusicDs["cleanReview"] = digitalMusicDs["reviewText"].apply
digitalMusicDs["cleanSummary"] = digitalMusicDs["summary"].apply(c
#Add sentiment of each review
digitalMusicDs["sentiment"] = digitalMusicDs["overall"].apply(sent

#Review length
digitalMusicDs["reviewLenght"] = digitalMusicDs["cleanReview"].app
digitalMusicDs["summaryLenght"] = digitalMusicDs["cleanSummary"].a

#Add sentiment of each review
digitalMusicDs["sentiment"] = digitalMusicDs["overall"].apply(sent

#Delete review with score equal to 2
digitalMusicDs = digitalMusicDs.dropna()



#Save processed data set in order to retrieve it
digitalMusicDs.to_pickle("./Dataset/processedDigitalMusic.pkl")
```

In [28]:

```
#Load dataset in a panda DataFrame
df = pd.read_json('./Dataset/Grocery_and_Gourmet_Food_5.json',line

groceryAndGourmetFoodDs = pd.DataFrame()
groceryAndGourmetFoodDs["overall"] = df["overall"]
groceryAndGourmetFoodDs["reviewText"] = df["reviewText"]
groceryAndGourmetFoodDs["summary"] = df["summary"]

#Apply data cleaning and text preprocessing to all dataset
#Add the processed data to the original data.
groceryAndGourmetFoodDs["cleanReview"] = groceryAndGourmetFoodDs['
groceryAndGourmetFoodDs["cleanSummary"] = groceryAndGourmetFoodDs[

#Add sentiment of each review
groceryAndGourmetFoodDs["sentiment"] = groceryAndGourmetFoodDs["ov

#Review length
groceryAndGourmetFoodDs["reviewLenght"] = groceryAndGourmetFoodDs[
groceryAndGourmetFoodDs["summaryLenght"] = groceryAndGourmetFoodDs

#Add sentiment of each review
groceryAndGourmetFoodDs["sentiment"] = groceryAndGourmetFoodDs["ov

#Delete review with score equal to 2
groceryAndGourmetFoodDs = groceryAndGourmetFoodDs.dropna()

#Save processed data set in order to retrieve it
```

```
groceryAndGourmetFoodDs.to_pickle("./Dataset/processedGroceryAndGo
```

Some statistics for both dataset

Digital music dataset:

In [234]:
```
#Save processed data set in order to retrieve it
digitalMusicDs = pd.read_pickle("./Dataset/processedDigitalMusic.p
print("Dataset dimension: ",digitalMusicDs.shape)
print("Median of review lenght: ", digitalMusicDs["reviewLenght"].
print("First two rows of data:")
digitalMusicDs.iloc[:2]

```

```
Dataset dimension:  (61696, 8)
Median of review lenght:  76.0
First two rows of data:
```

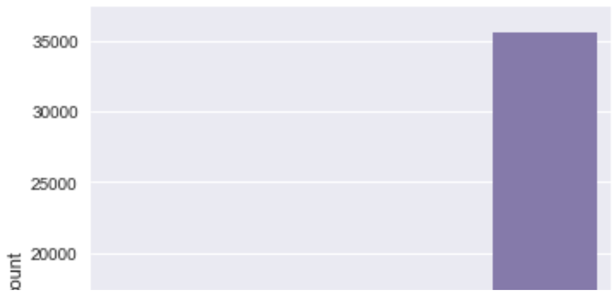
Out[234]:

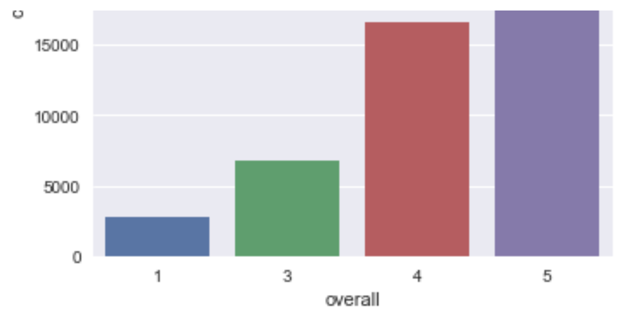
	overall	reviewText	summary	cleanReview	cleanSummary	sentiment	reviewLenght	sumr
0	5	It's hard to believe "Memory of Trees" came ou...	Enya's last great album	hard believ memori tree came year ago held wel...	enya last great album	1.0	89	
1	5	A clasically-styled and introverted album, Mem...	Enya at her most elegant	clasic style introvert album memori tree maste...	enya eleg	1.0	54	

In [236]:
```
print("Distribution of overall score:")
sns.catplot(x='overall',kind='count',data=digitalMusicDs,orient="h
```

```
Distribution of overall score:
```

Out[236]:
```
<seaborn.axisgrid.FacetGrid at 0x279a28f310>
```





This dataset has fewer reviews than the previous one, but a much longer average length of the latter.

Grocery and food dataset:

In [237]:
```
#Save processed data set in order to retrieve it
groceryAndGourmetFoodDs = pd.read_pickle("./Dataset/processedGroce
print("Dimension of data:", groceryAndGourmetFoodDs.shape)
print("Median of review lenght: ", groceryAndGourmetFoodDs["review
print("First two rows of data:")
groceryAndGourmetFoodDs.iloc[:2]

```

```
Dimension of data: (143337, 8)
Median of review lenght:  32.0
First two rows of data:
```

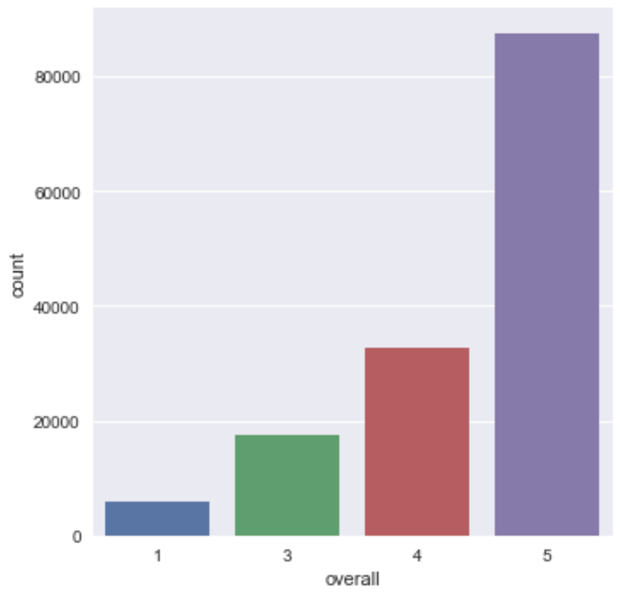
Out[237]:

	overall	reviewText	summary	cleanReview	cleanSummary	sentiment	reviewLenght	sumr
0	4	Just another flavor of Kit Kat but the taste i...	Good Taste	anoth flavor kit kat tast uniqu bit differ thi...	good tast	1.0	14	
1	3	I bought this on impulse and it comes from Jap...	3.5 stars, sadly not as wonderful as I had hoped	bought impuls come japan amus famili weird sta...	star sad wonder hope	1.0	50	

In [238]:
```
print("Distribution of overall score:")
sns.catplot(x='overall',kind='count',data=groceryAndGourmetFoodDs,
```

```
Distribution of overall score:
```

Out[238]:
```
<seaborn.axisgrid.FacetGrid at 0x1a8fd5c250>
```



This dataset is comparable to the previous one, with about the same number of reviews and about the same average length.

Let's do the test

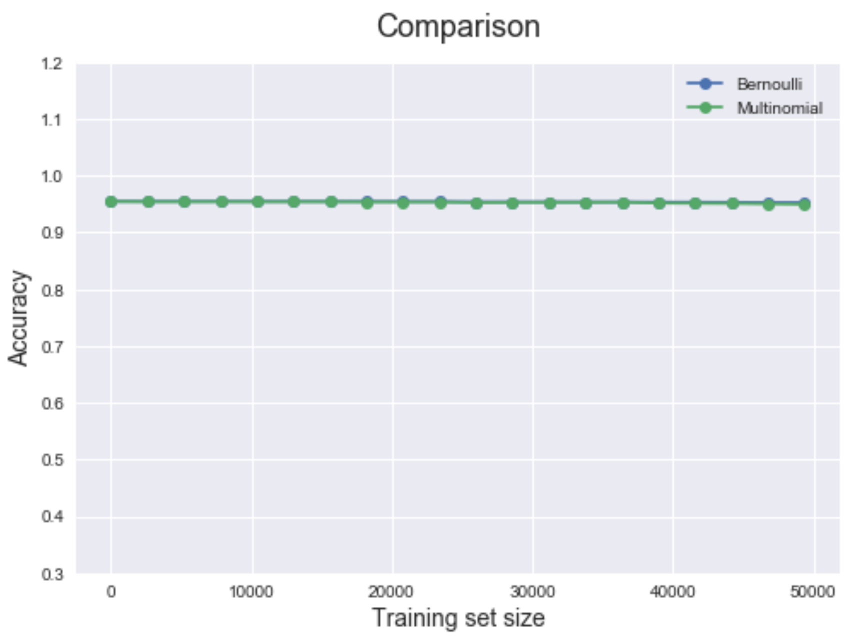
We will repeat the test in the same order of the previus chapter

Digital music dataset

In [32]:
```
models = []
models.append(BernoulliNB())
models.append(MultinomialNB())
```

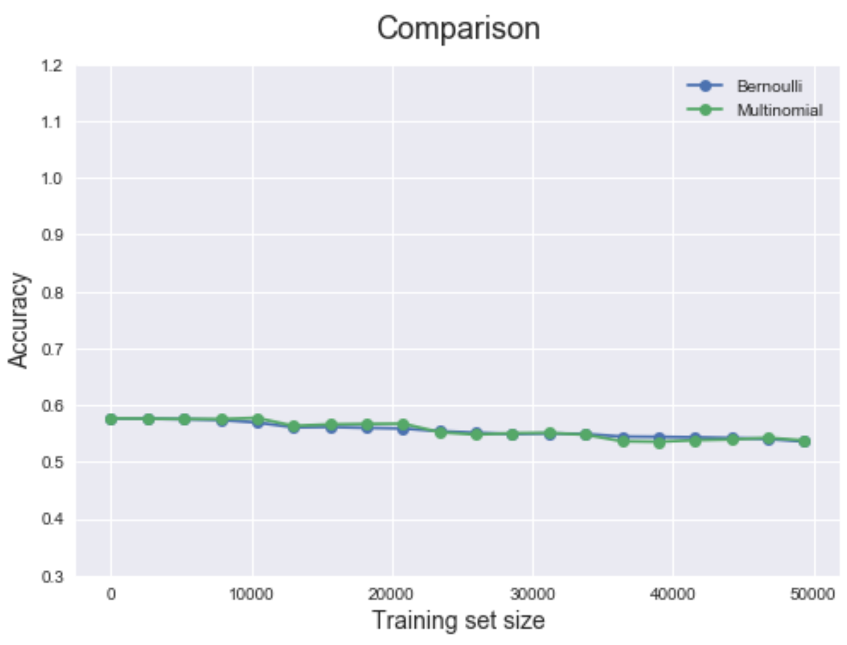
In [239]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanReview'

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

```
Number of words in vocabulary : 77050
```



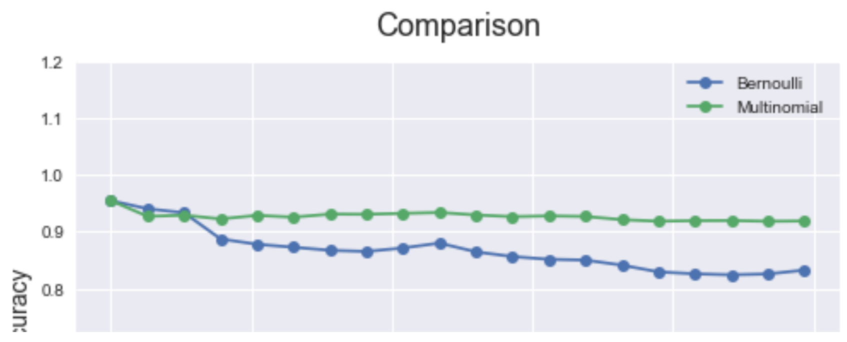
In [240]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanReview'

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

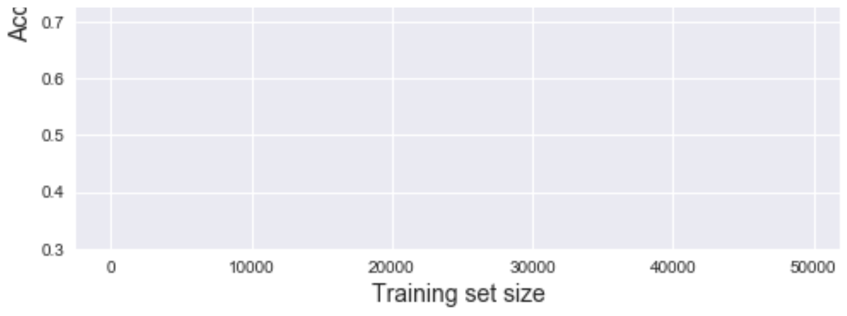
```
Number of words in vocabulary : 77050
```



In [241]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanReview'

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

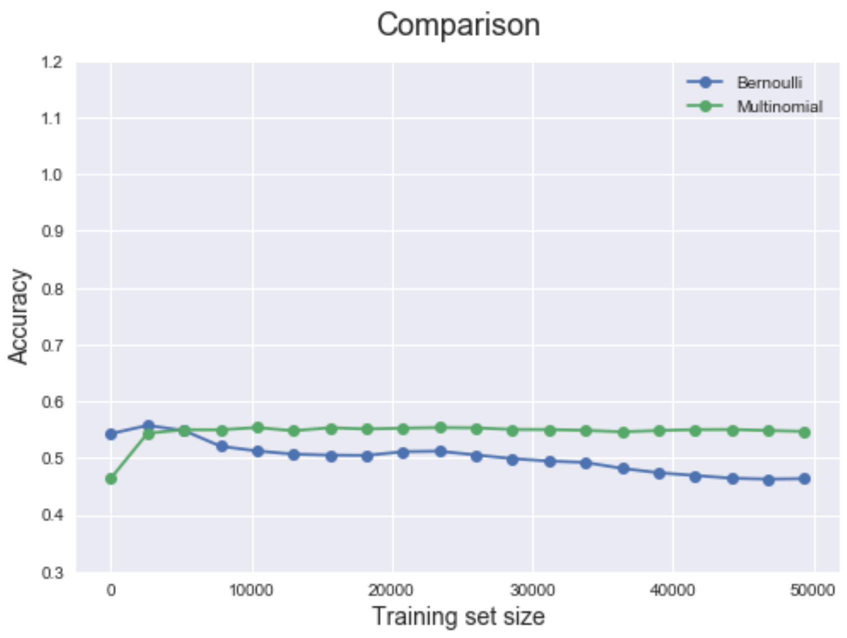
```
Number of words in vocabulary : 50
```





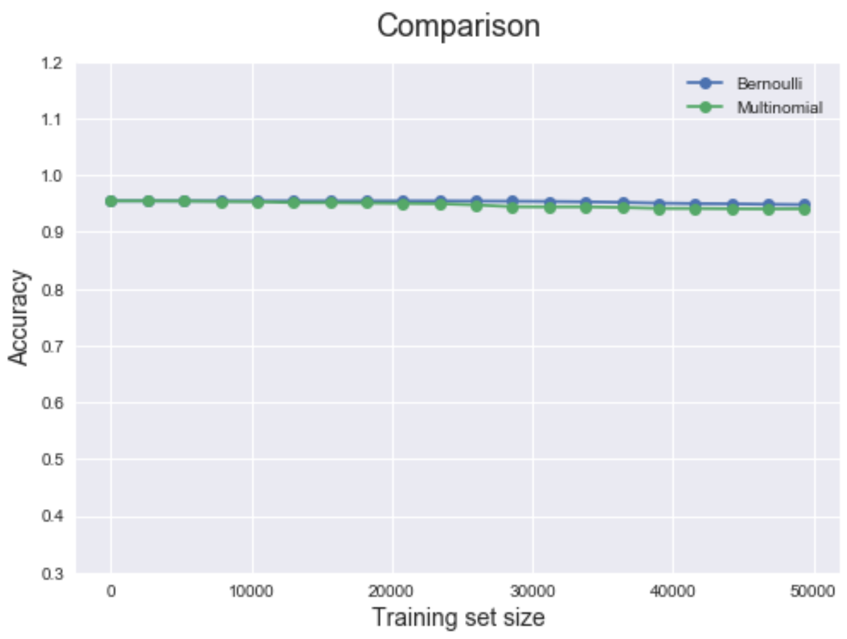
In [242]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanReview'

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Multinomial")
```

```
Number of words in vocabulary : 50
```



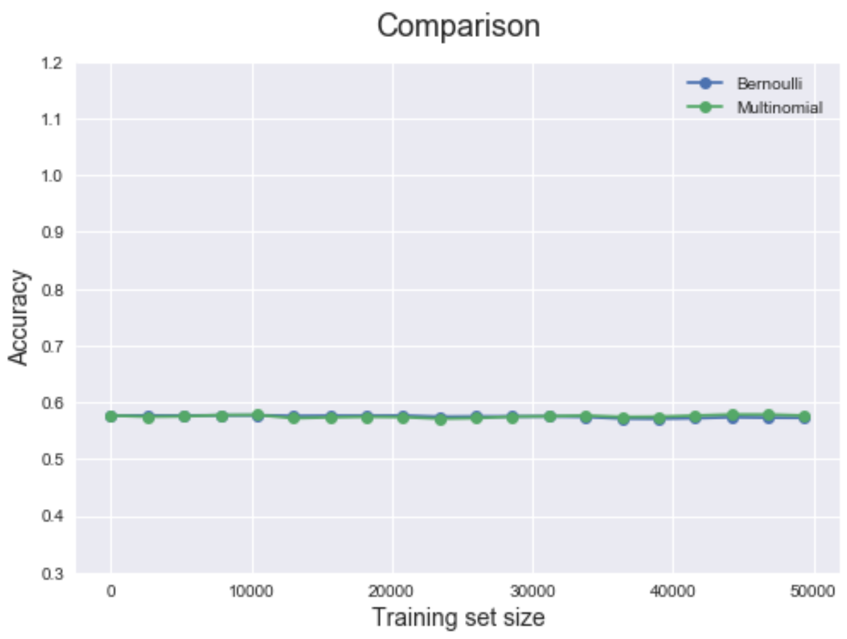
In [243]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanSummary

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 12704
```



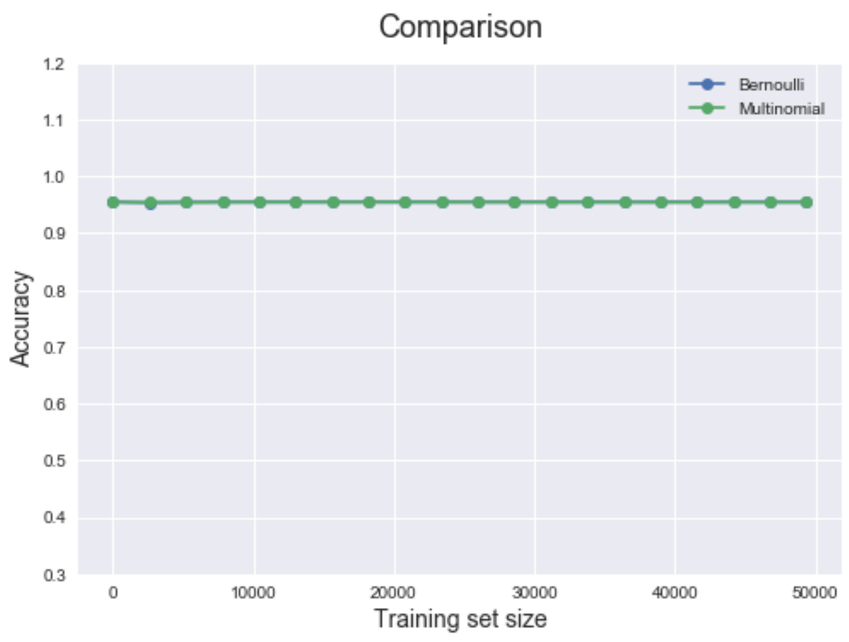
In [244]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanSummary

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 12704
```



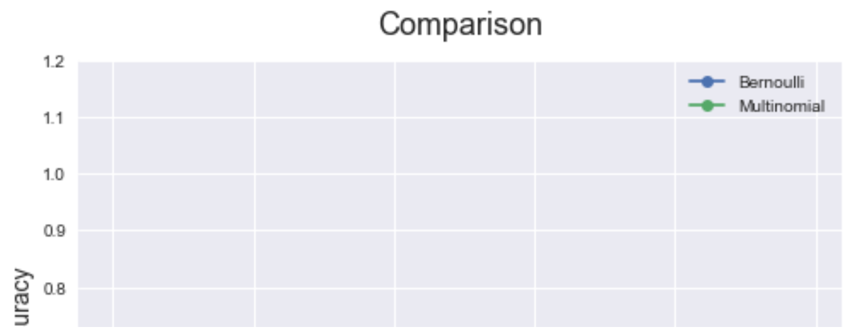
In [245]:
```
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanSummary

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

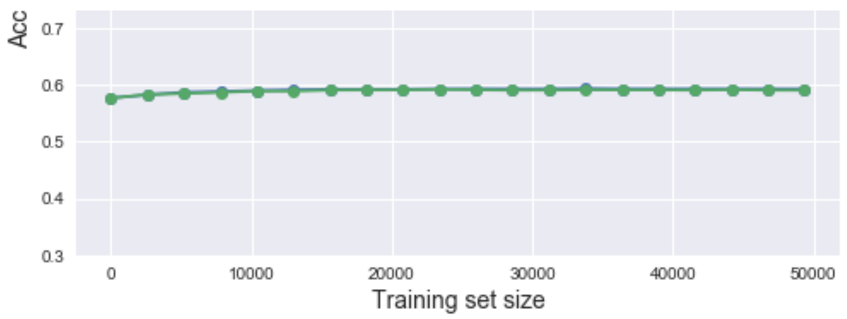
```
Number of words in vocabulary : 50
```



In [246]:
```python
XBern, XBin, y = dataPlotPreparation( digitalMusicDs["cleanSummary
                                      
plotValidationCurve(estimator = models[0],
                   X = XBin,
                    y = y,
                     trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Bernoulli")
plotValidationCurve(estimator = models[1],
                   X = XBin,
                    y = y,
                     trainSizes = np.linspace(20, XBern.shape[0]-(X
                     title="Multinomial")
```

```
Number of words in vocabulary : 50
```





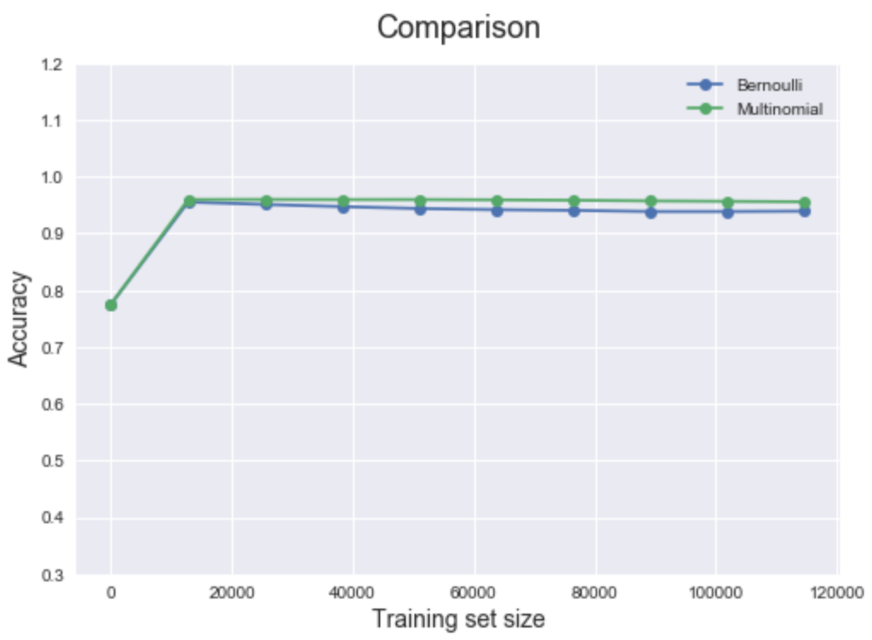
Summing up

1. We note that if the number of targets is greater than 2 (when the overall field is used) the complessive accuracy score is lower. This is probably due to the major average length of the reviews compared to the previous dataset.
2. When decreasing the number of features, the bernoulli model in this case does not perform better than the multinomial model. Indeed, it behaves much worse and the difference between the two widens as the size of the dataset increases. Once again in support of the fact that the multinomial model is the only one of the two that receives advantages from the larger dimensions of the training set.
3. When using short reviews, the bernoulli model seems to behave slightly better than the binomial
4. As the number of targets increases, the performance of both models decreases, but the multinomial responds better, also taking advantage of the larger size of the training set.
5. When the reviews are short and the number of words in the dictionary is low, both models behave the same way.

Grocery and gourmet food dataset

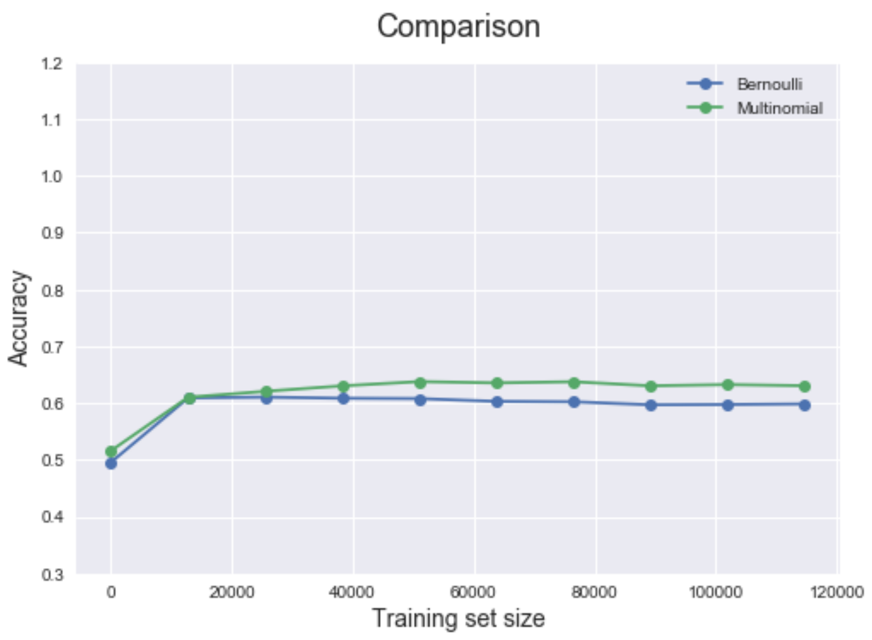
In [42]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 50848
```



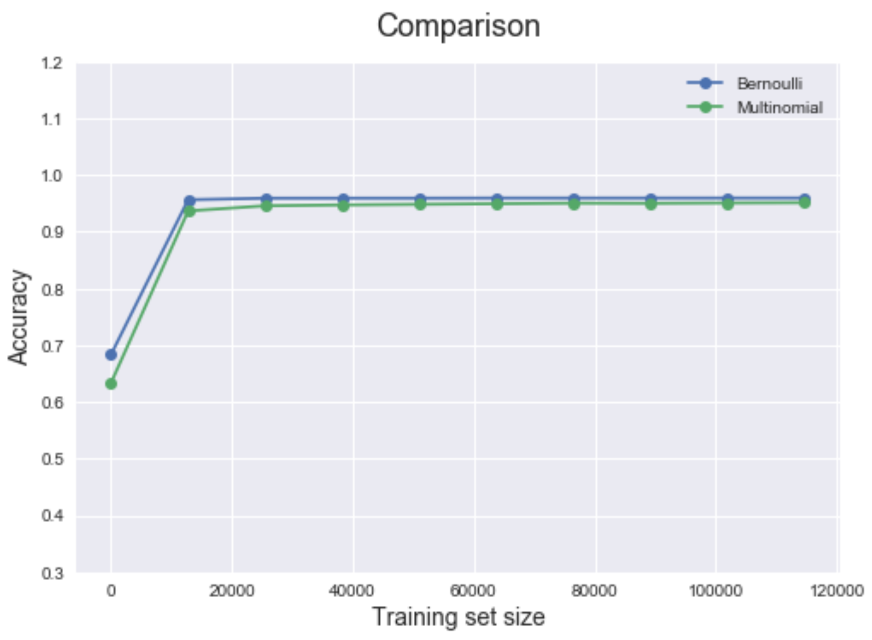
In [43]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 50848
```



In [44]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

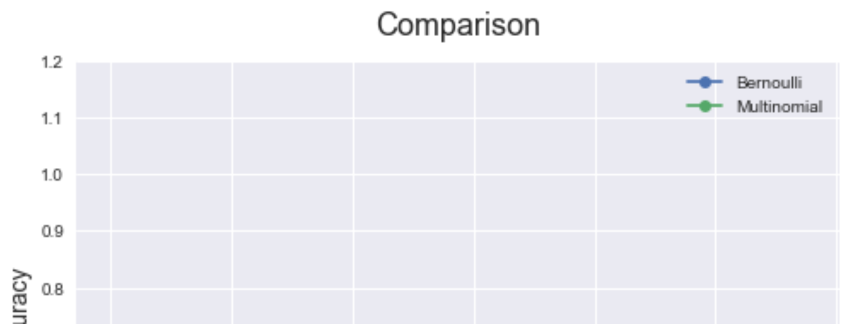
```
Number of words in vocabulary : 50
```

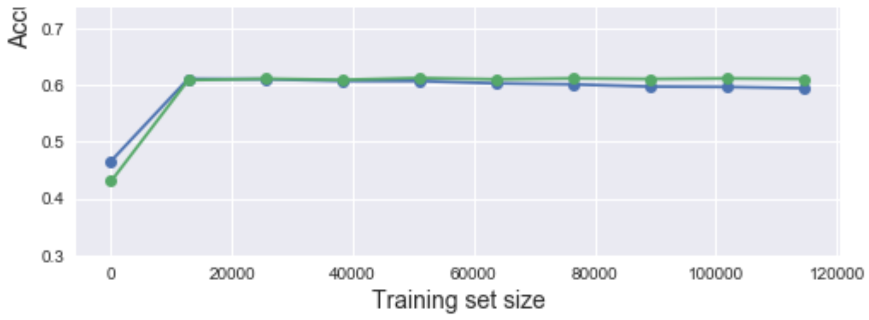


In [45]:

```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

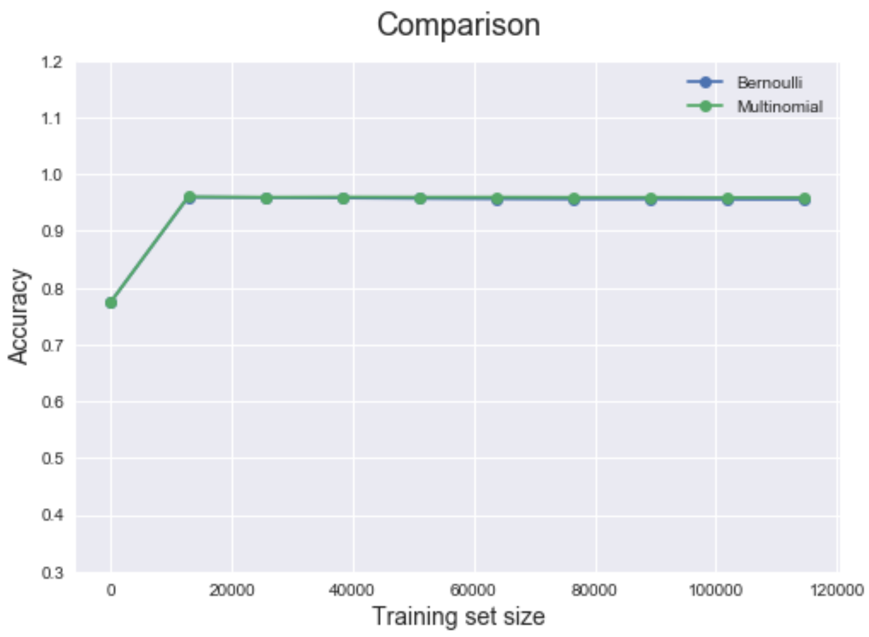
```
Number of words in vocabulary : 50
```





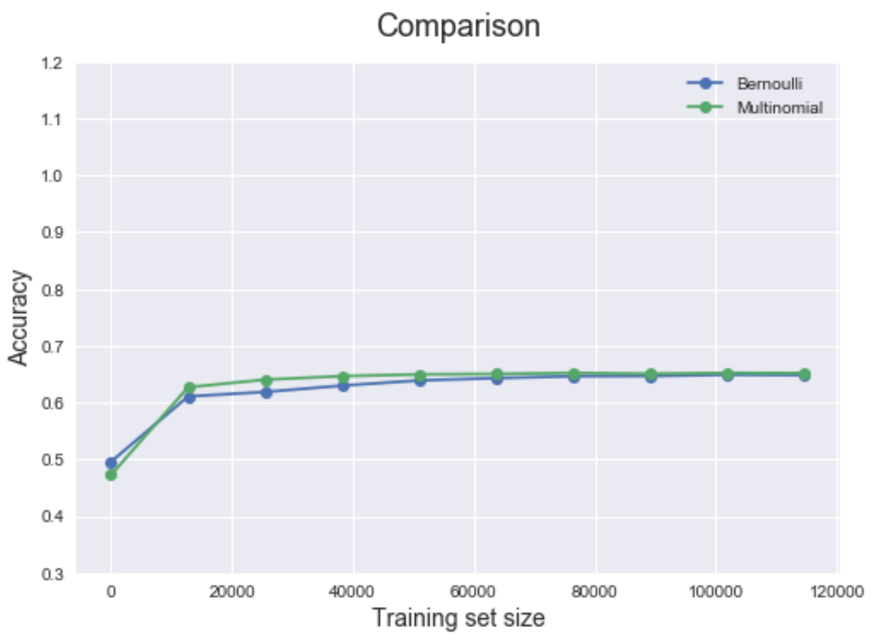
In [46]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 11685
```



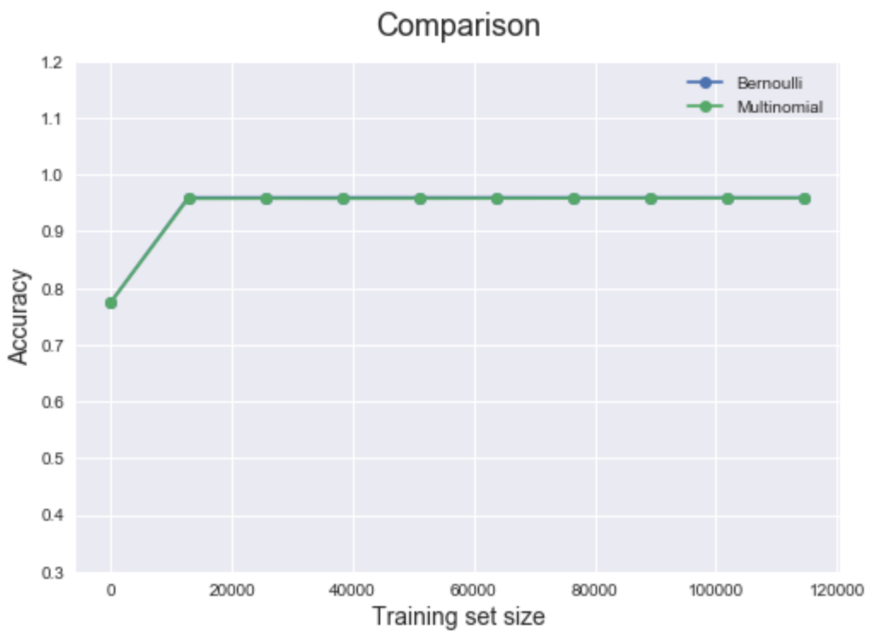
In [47]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 11685
```



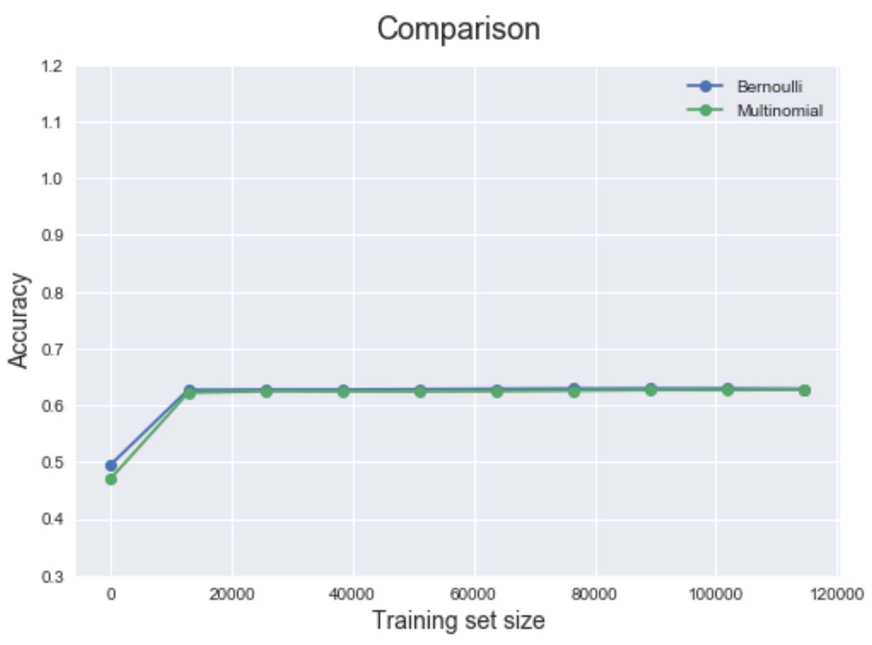
In [48]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 50
```



In [49]:
```
XBern, XBin, y = dataPlotPreparation( groceryAndGourmetFoodDs["cle

plotValidationCurve(estimator = models[0],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                       title="Bernoulli")
plotValidationCurve(estimator = models[1],
                    X = XBin,
                     y = y,
                      trainSizes = np.linspace(20, XBern.shape[0]-(X
                      title="Multinomial")
```

```
Number of words in vocabulary : 50
```



Summing up

1. **Also in this case, the Multinomial model can benefit from the size of the training test, and this allows it to work better even when the number of targets grows.**
2. **When the number of features is limited, the Bernoulli model returns to behave like the first dataset. This can be explained by the fact that the two datasets are very similar, and testifies how the length of the reviews plays a decisive role in the increase / decrease of the performance of the bernoulli model, which in case the reviews are very short, behaves exactly like the bernoulli model.**

```
In [ ]:
```