Movie and Book title analysis - Can we predict the popularity of a title in books based on the learings from movie titles?

In [1]:

```
import pandas as pd
import collections
import os
import nltk
import numpy as np
import matplotlib.pyplot as plt
import string

nltk.download("punkt")
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ebalgza\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [2]:

```
print(os.listdir("movies"))
```

```
['movies.csv', 'ratings.csv.001', 'ratings.csv.002', 'ratings.csv.00
3', 'ratings.csv.004', 'ratingsorig.csv']
```

In [3]:

```
#Let's load the data
# If you are like me, and the laptop can't process 500 megs of rating, use a slice
# Feel free to use the whole set if you have da powah, just uncomment last line
movies = pd.read_csv("movies/movies.csv", sep=",")
ratings = pd.read_csv("movies/ratings.csv.001")
#ratings = pd.read_csv("movies/ratingsorig.csv") #my laptop can't handle the size
```

In [4]:

```
movies.head()
```

Out[4]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [5]:

```
ratings.head()
```

Out[5]:

	userId	movieId	rating	timestamp
0	1	2	3.5	1112486027
1	1	29	3.5	1112484676
2	1	32	3.5	1112484819
3	1	47	3.5	1112484727
4	1	50	3.5	1112484580

Sum the number of positive reviews for each movie

In [6]:

```
#How do we define a really good rating?
#Not just a plain old - yeah it was alright kinda movie
#Let's make it way above average, 4.5 and 5
positive_ratings = ratings['rating'] > 4
positive_ratings[6001266:6001276]
```

Out[6]:

```
6001266    False
6001267    False
6001268     True
6001269    False
6001270     True
6001271    False
6001272    False
6001273    False
6001274    False
6001275     True
Name: rating, dtype: bool
```

In [7]:

```
#I want to see those raving good ratigs
good_film_ratings = ratings[positive_ratings]
good_film_ratings[105:115]
```

Out[7]:

	userId	movieId	rating	timestamp
414	3	2985	5.0	944919729
416	3	3033	5.0	944919729
417	3	3039	5.0	944919189
422	3	5060	5.0	944917450
439	4	454	5.0	840878944
450	4	733	5.0	840879322
452	5	11	5.0	851527751
455	5	62	5.0	851526935
459	5	141	5.0	851526935
460	5	150	5.0	851527514

In [8]:

```
#What does this look like for one particular movie?
movie1196_filter = good_film_ratings['movieId'] == 1196
movies1196 = good_film_ratings[movie1196_filter]
movies1196.head()
```

Out[8]:

	userId	movieId	rating	timestamp
30	1	1196	4.5	1112484742
190	2	1196	5.0	974821014
286	3	1196	5.0	944917859
512	5	1196	5.0	851617674
602	7	1196	5.0	1011204572

In [9]:

```
good_film_ratings.head()
```

Out[9]:

	userId	movieId	rating	timestamp
30	1	1196	4.5	1112484742
31	1	1198	4.5	1112484624
131	1	4993	5.0	1112484682
142	1	5952	5.0	1112484619
158	1	7153	5.0	1112484633

In [10]:

```
#Feel free to take a look at the way the data currently looks
movies.head()
#ratings.head()
#good_film_ratings.head()
```

Out[10]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [ ]:

```
```

In [11]:

```
#Let's sum up the excellent ratings for the movies
number_of_excellent_ratings = good_film_ratings.groupby(by = "movieId")['rating'].co
sum_of_excellent_ratings = good_film_ratings.groupby(by = "movieId")['rating'].sum()
```

In [12]:

```
number_of_excellent_ratings[35:45]
#sum_of_excellent_ratings[35:45] #uncomment this line to see the overall weight thes
```

Out[12]:

```
movieId
36    2010
37       4
38      28
39    1096
40      55
41     363
42      48
43      95
44     160
45     325
Name: rating, dtype: int64
```

In [13]:

```
# But a movie is not good must because it has positive reviews
# It needs to have a whole lot more excellent reviews than negatives
# Let's create a negative ratings list function
print("For reference, here are the positive ratings:")
print(positive_ratings[6001265:6001270])
```

```
For reference, here are the positive ratings:
6001265    False
6001266    False
6001267    False
6001268     True
6001269    False
Name: rating, dtype: bool
```

In [14]:

```
#Any ratings that is less than 2 is truly a baaad one
print("Negative reviews:")
negative_ratings = ratings['rating'] < 2
negative_ratings.head()
```

```
Negative reviews:
```

Out[14]:

```
0    False
1    False
2    False
3    False
4    False
Name: rating, dtype: bool
```

In [15]:

```
#I want to see those horrendously bad ratigs
bad_film_ratings = ratings[negative_ratings]
bad_film_ratings[105:115]
```

Out[15]:

	userId	movieId	rating	timestamp
3297	28	185	1.0	834093021
3298	28	196	1.0	834093089
3302	28	266	1.0	834093066
3315	28	417	1.0	834092871
3323	28	590	1.0	834092660
3434	29	371	1.0	835638228
3504	30	163	1.0	1204791725
3505	30	168	0.5	1204791801
3510	30	555	1.5	1204791757
3538	31	527	0.5	1424733598

In [16]:

```
#Let's sum up the bad ratings for the movies
number_of_bad_ratings = bad_film_ratings.groupby(by = "movieId")['rating'].count()
sum_of_bad_ratings = bad_film_ratings.groupby(by = "movieId")['rating'].sum()
```

In [17]:

```
number_of_bad_ratings
#sum_of_bad_ratings # Uncomment this line to see the total negative weight
```

Out[17]:

```
movieId
1          274
2          419
3          280
4          106
5          290
6          102
7          175
8           22
9          111
10         246
11         104
12         225
13          26
14          69
15         146
16          78
17         148
18         115
19        1420
20         116
21         218
22         131
23          79
24         191
25         301
26          22
27          17
28          15
29          80
30          12
          ...
128632       1
128736       1
128914       1
128975       2
129015       1
129030       2
129354       2
129370       1
129428       1
129456       1
129514       1
129699       1
```

```
129707       1
129822       1
129834       1
129937       1
130052       1
130075       2
130466       1
130490       2
130496       1
130498       1
130500       1
130502       1
130506       1
130508       1
130510       1
130672       1
130804       1
130836       1
Name: rating, Length: 13088, dtype: int64
```

In [18]:

```
# Let's turn the sums into a combined score that expresses the overall quality, so t
# of each movie based on the most positive and most negative ratings
print(sum_of_excellent_ratings.head())
print(sum_of_bad_ratings.head())
```

```
movieId
1    23501.5
2     2988.0
3     1827.5
4      345.0
5     1477.0
Name: rating, dtype: float64
movieId
1    282.0
2    452.5
3    284.5
4    108.0
5    285.5
Name: rating, dtype: float64
```

In [19]:

```
#Let's leave mediocrity behind.
# A truly wonderful movie is defined by the weight of outstanding reviews
# at least for my purposes
quality_score = sum_of_excellent_ratings - sum_of_bad_ratings

qs = quality_score.to_frame()
qs.head()
```

Out[19]:

	rating
movieId	
1	23219.5
2	2535.5
3	1543.0
4	237.0
5	1191.5

In [20]:

```
qs = qs.rename(columns={"rating":"Quality_Score"})
qs.head()
```

Out[20]:

	Quality_Score
movieId	
1	23219.5
2	2535.5
3	1543.0
4	237.0
5	1191.5

In [21]:

```
#Let's get the movies together with their awesomeness scores
ratings_with_quality_score = movies.merge(qs, on="movieId", how="left")
```

In [22]:

```
ratings_with_quality_score.head()
```

Out[22]:

	movieId	title	genres	Quality_Score
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	23219.5
1	2	Jumanji (1995)	Adventure|Children|Fantasy	2535.5
2	3	Grumpier Old Men (1995)	Comedy|Romance	1543.0
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	237.0
4	5	Father of the Bride Part II (1995)	Comedy	1191.5

In [23]:

```
#What are the first 1000 highest ranking movies based on the overwhelming positive r
ratings_with_quality_score.sort_values(by = "Quality_Score", ascending=False)[:1000]
```

		the...		
602	608	Fargo (1996)	Comedy|Crime|Drama|Thriller	28341.5
4897	4993	Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy	28027.0
1184	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi	25983.5
583	589	Terminator 2: Judgment Day (1991)	Action|Sci-Fi	25691.0
46	47	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller	25615.0

In [24]:

```
# Let's define a movie as excellent if its positive review weight is
# at least 1000 scores higher than its negative weight
excellence_score=999
ratings_with_quality_score["excellent_film"] = np.where(ratings_with_quality_score['
```

In [25]:

```
ratings_with_quality_score.head()
#ratings_with_quality_score.shape
#ratings_with_quality_score.tail()
```

Out[25]:

	movieId	title	genres	Quality_Score	excelle
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	23219.5	
1	2	Jumanji (1995)	Adventure|Children|Fantasy	2535.5	
2	3	Grumpier Old Men (1995)	Comedy|Romance	1543.0	
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	237.0	
4	5	Father of the Bride Part II (1995)	Comedy	1191.5	

In [26]:

```
stop_words = list(string.punctuation)
stop_words += nltk.corpus.stopwords.words("english")
stop_words += ["1","2","3","4","5","6","7","8","9","0","the","The", "'s", "'re"]
stop_words += ["1990","1995","1996","1997","1998","1999","1994","1993","1992","1991'
stop_words += ["1980","1985","1986","1987","1988","1989","1984","1983","1982","1981'
stop_words += ["1950","1955","1956","1957","1958","1959","1954","1953","1952","1951'
stop_words += ["1960","1965","1966","1967","1968","1969","1964","1963","1962","1961'
stop_words += ["1970","1975","1976","1977","1978","1979","1974","1973","1972","1971'
stop_words += ["1940","1945","1946","1947","1948","1949","1944","1943","1942","1941'
stop_words += ["1940","1945","1946","1947","1948","1949","1944","1943","1942","1941'
stop_words += ["1930","1935","1936","1937","1938","1939","1934","1933","1932","1931'
stop_words += ["2000","2005","2006","2007","2008","2009","2004","2003","2002","2001'
stop_words += ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019'
```

In [27]:

```
def build_bag_of_words_features_filtered(words):
    return {
        word:1 for word in words \
        if not word in stop_words}
```

In [28]:

```
nltk.word_tokenize(ratings_with_quality_score.iloc[[1]]["title"].to_string())
#build_bag_of_words_features_filtered(ratings_with_quality_score.iloc[[2]]["title"])
```

Out[28]:

```
['1', 'Jumanji', '(', '1995', ')']
```

In [29]:

```
def tokenize_title(line):
    return nltk.word_tokenize(line)
```

In [30]:

```
tokenize_title(ratings_with_quality_score.iloc[[1]]["title"].to_string())
```

Out[30]:

```
['1', 'Jumanji', '(', '1995', ')']
```

In [31]:

```
def tokenize_and_filter_title(title):
    all_words = tokenize_title(title)
    filtered_words = build_bag_of_words_features_filtered(all_words)
    #print(filtered_words)
    return filtered_words

for title in ratings_with_quality_score["title"]:
    tokenize_and_filter_title(title)

ratings_with_quality_score.head()
```

Out[31]:

	movieId	title	genres	Quality_Score	excelle
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	23219.5	
1	2	Jumanji (1995)	Adventure|Children|Fantasy	2535.5	
2	3	Grumpier Old Men (1995)	Comedy|Romance	1543.0	
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	237.0	
4	5	Father of the Bride Part II (1995)	Comedy	1191.5	

In [32]:

```
list(ratings_with_quality_score)
```

Out[32]:

```
['movieId', 'title', 'genres', 'Quality_Score', 'excellent_film']
```

In [33]:

```
def create_target_df(ratings_with_quality_score):
    data = []
    for movie in ratings_with_quality_score.itertuples(index=True, name='title'):
        title = getattr(movie, "title")
        bow = tokenize_and_filter_title(title)
        is_excellent = getattr(movie, "excellent_film")
        data += [(bow,is_excellent)]

    return data
```

In [34]:

```
target = (create_target_df(ratings_with_quality_score))
```

In [35]:

```
target[8]
```

Out[35]:

```
({'Sudden': 1, 'Death': 1}, False)
```

In [36]:

```
targetlist = list(target)

targetlist[8]
```

Out[36]:

```
({'Sudden': 1, 'Death': 1}, False)
```

In [37]:

```
from nltk.classify import NaiveBayesClassifier
split=10000
```

In [38]:

```
success_classifier = NaiveBayesClassifier.train(target[:split])
```

In [39]:

```
nltk.classify.util.accuracy(success_classifier, target[:split])*100
```

Out[39]:

```
84.38
```

In [40]:

```
import random
def montecarlo_classifier():
    split = random.randint(0,(len(target)-1))
    success_classifier = NaiveBayesClassifier.train(target[:split])
    accuracy = round(nltk.classify.util.accuracy(success_classifier, target[split:]]
    #print("{}{}{}".format("accurary: ",accuracy,"\n\n"))
    #print("{}{}".format("split: ",split))
    return split,accuracy
```

In [71]:

```
def montecarlo_runner(iterations):
    perf=[]
    for iteration in range(iterations):
        perf.append(montecarlo_classifier())

    return perf
```

In [91]:

```
number_of_runs = 10
perf=montecarlo_runner(number_of_runs)
print(perf)
```

```
[(24210, 90.91), (15633, 88.6), (1484, 81.27), (10744, 88.01), (1485
7, 88.78), (17341, 88.88), (21252, 89.4), (26256, 90.7), (9056, 88.7
6), (4908, 85.91)]
```

In [92]:

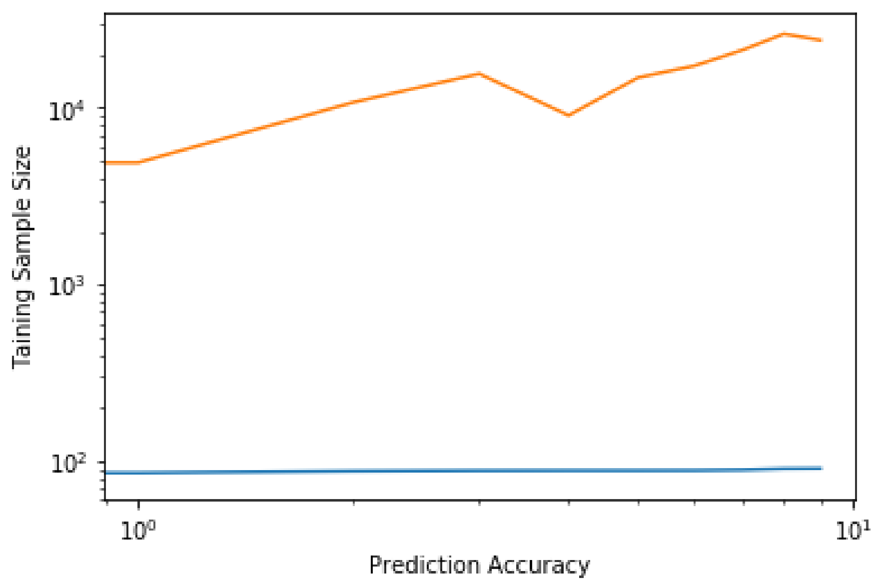
```
sorted_accuracy = sorted(perf,key=lambda x: x[1], reverse=False)
sorted_accuracy[1]

transposed_values=[]
for sample,accuracy in sorted_accuracy:
    transposed_values.append([accuracy,sample])

transposed_values
```

Out[92]:

```
[[81.27, 1484],
 [85.91, 4908],
 [88.01, 10744],
 [88.6, 15633],
 [88.76, 9056],
 [88.78, 14857],
 [88.88, 17341],
 [89.4, 21252],
 [90.7, 26256],
 [90.91, 24210]]
```

In [93]:

```
plt.loglog(transposed_values)
plt.ylabel("Taining Sample Size")
plt.xlabel("Prediction Accuracy")
```

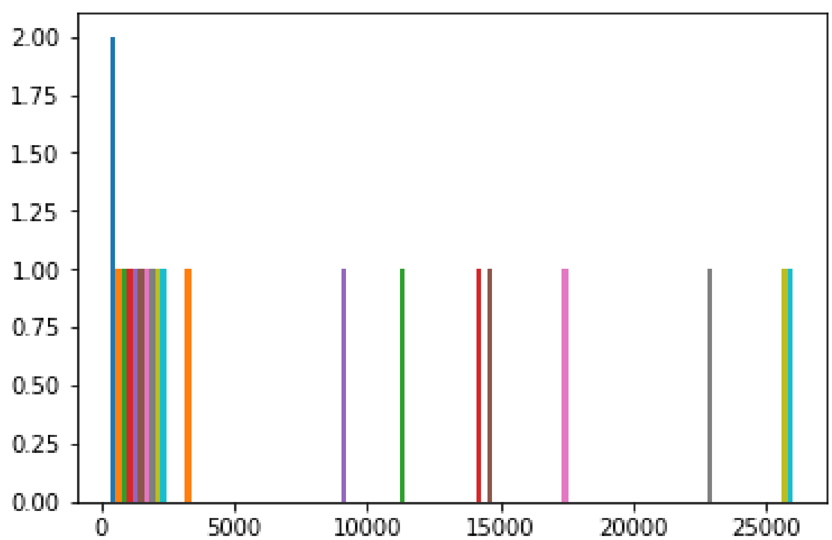


In [94]:

```
plt.hist(transposed_values)
```

Out[94]:

```
([array([2., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
  array([1., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
  array([1., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
  array([1., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
  array([1., 0., 0., 1., 0., 0., 0., 0., 0., 0.]),
  array([1., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
  array([1., 0., 0., 0., 0., 0., 1., 0., 0., 0.]),
  array([1., 0., 0., 0., 0., 0., 0., 0., 1., 0.]),
  array([1., 0., 0., 0., 0., 0., 0., 0., 0., 1.]),
  array([1., 0., 0., 0., 0., 0., 0., 0., 0., 1.])],
 array([   81.27 ,  2698.743,  5316.216,  7933.689, 10551.162, 13168.
635,
        15786.108, 18403.581, 21021.054, 23638.527, 26256.   ]),
 <a list of 10 Lists of Patches objects>)
```



In [95]:

```
for point in sorted_accuracy:
    print(point[0])
    print(point[1])
```

```
1484
81.27
4908
85.91
10744
88.01
15633
88.6
9056
88.76
14857
88.78
17341
88.88
21252
89.4
26256
90.7
24210
90.91
```

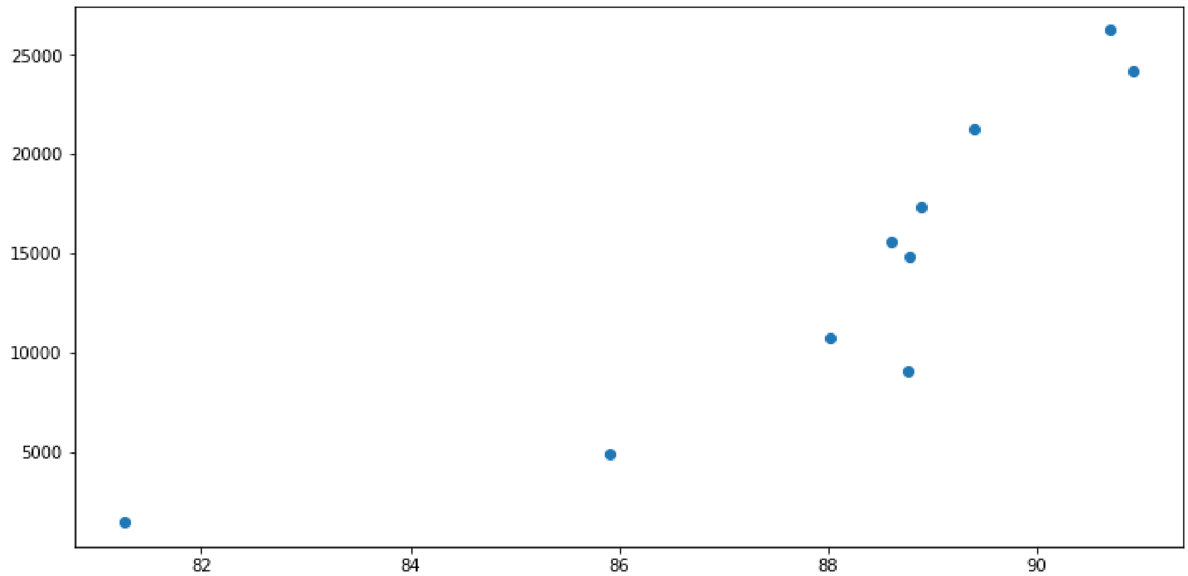
In [96]:

```
print(sorted_accuracy)
print(transposed_values)

retransposed = list(zip(*transposed_values))
print(retransposed)
```

```
[(1484, 81.27), (4908, 85.91), (10744, 88.01), (15633, 88.6), (9056,
88.76), (14857, 88.78), (17341, 88.88), (21252, 89.4), (26256, 90.7),
(24210, 90.91)]
[[81.27, 1484], [85.91, 4908], [88.01, 10744], [88.6, 15633], [88.76,
9056], [88.78, 14857], [88.88, 17341], [89.4, 21252], [90.7, 26256],
[90.91, 24210]]
[(81.27, 85.91, 88.01, 88.6, 88.76, 88.78, 88.88, 89.4, 90.7, 90.91),
(1484, 4908, 10744, 15633, 9056, 14857, 17341, 21252, 26256, 24210)]
```

In [97]:

```
plt.figure(figsize = (12,6))
plt.scatter(retransposed[0],retransposed[1])
plt.show()
```



In [102]:

```
perf=montecarlo_runner(100)
```

In [103]:

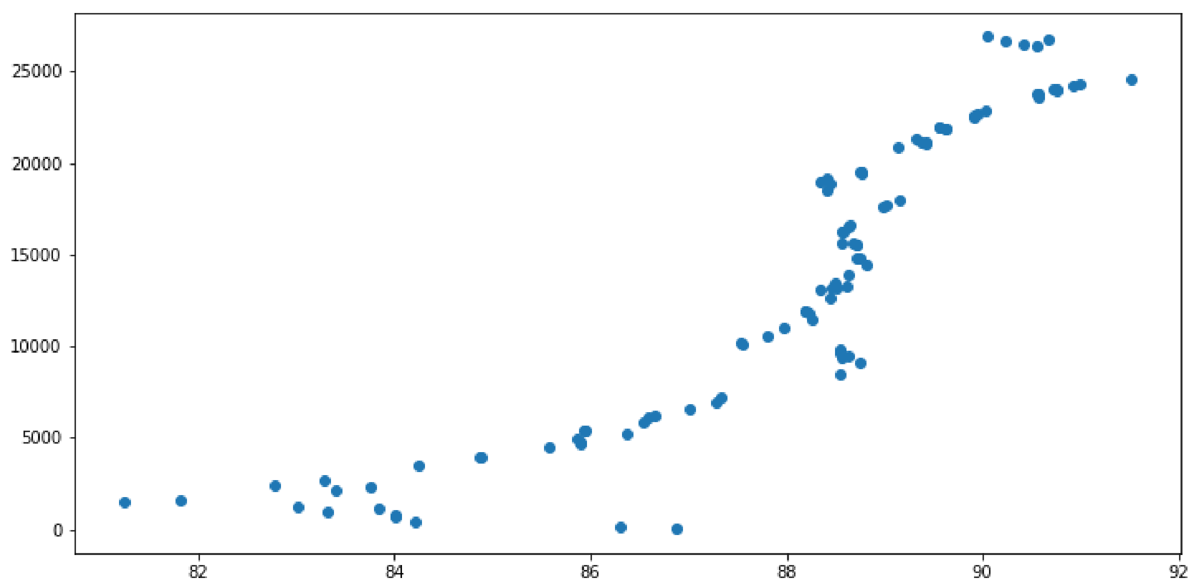
```
data = list(zip(*perf))
data
```

Out[103]:

```
[(21155,
  26886,
  24205,
  21963,
  59,
  11942,
  3981,
  979,
  19438,
  11830,
  15492,
  24533,
  13406,
  23991,
  1137,
  4685,
  11947,
  6615,
```

In [104]:

```
plt.figure(figsize = (12,6))
plt.scatter(data[1],data[0])
plt.show()
```

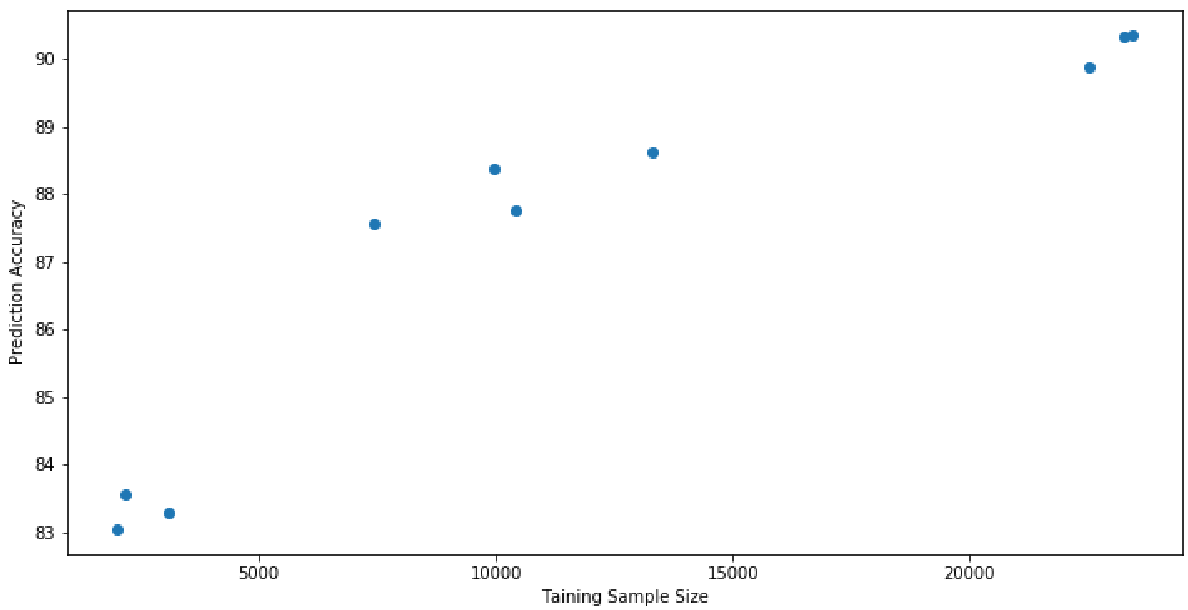


In [110]:

```
def go_the_whole_hog(iterations):
    raw_data=montecarlo_runner(iterations)
    serialized_data = list(zip(*raw_data))
    plt.figure(figsize = (12,6))
    plt.xlabel("Taining Sample Size")
    plt.ylabel("Prediction Accuracy")
    plt.scatter(serialized_data[0],serialized_data[1])
    plt.show()
```

In [111]:

```
go_the_whole_hog(10)
```



In [115]:

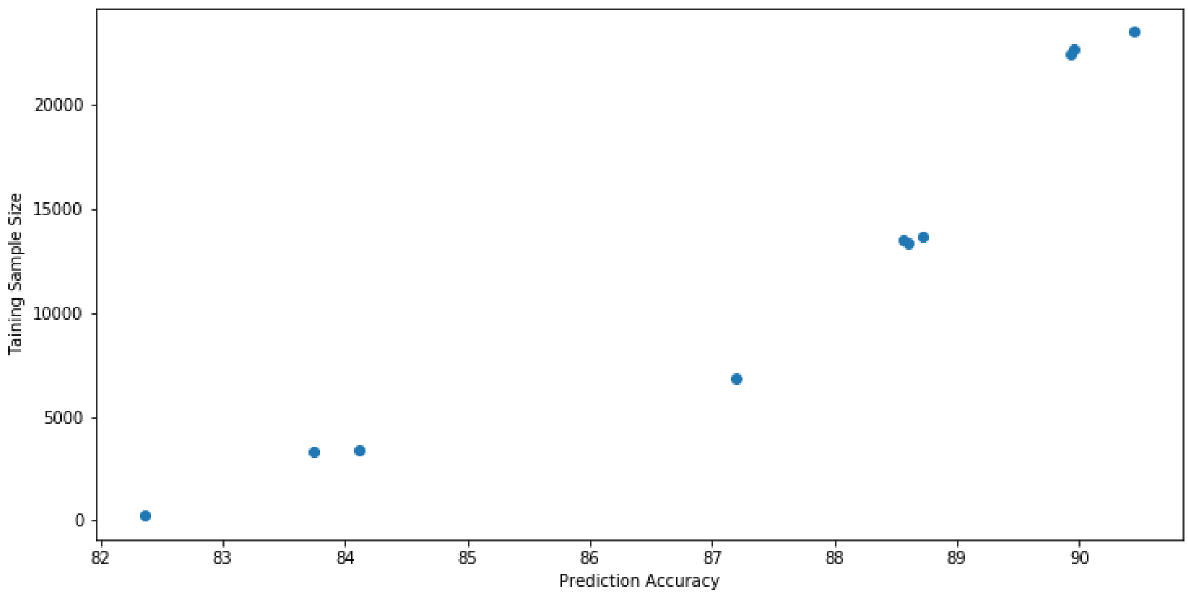
```
def generate_data_for_(iterations):
    raw_data=montecarlo_runner(iterations)
    serialized_data = list(zip(*raw_data))
    return serialized_data

def visualize_data(data):
    plt.figure(figsize = (12,6))
    plt.ylabel("Taining Sample Size")
    plt.xlabel("Prediction Accuracy")
    plt.scatter(data[1],data[0])
    plt.show()
```

In [117]:

```
def how_big_should_my_sample_be(iterations):
    data = generate_data_for_(iterations)
    visualize_data(data)
```

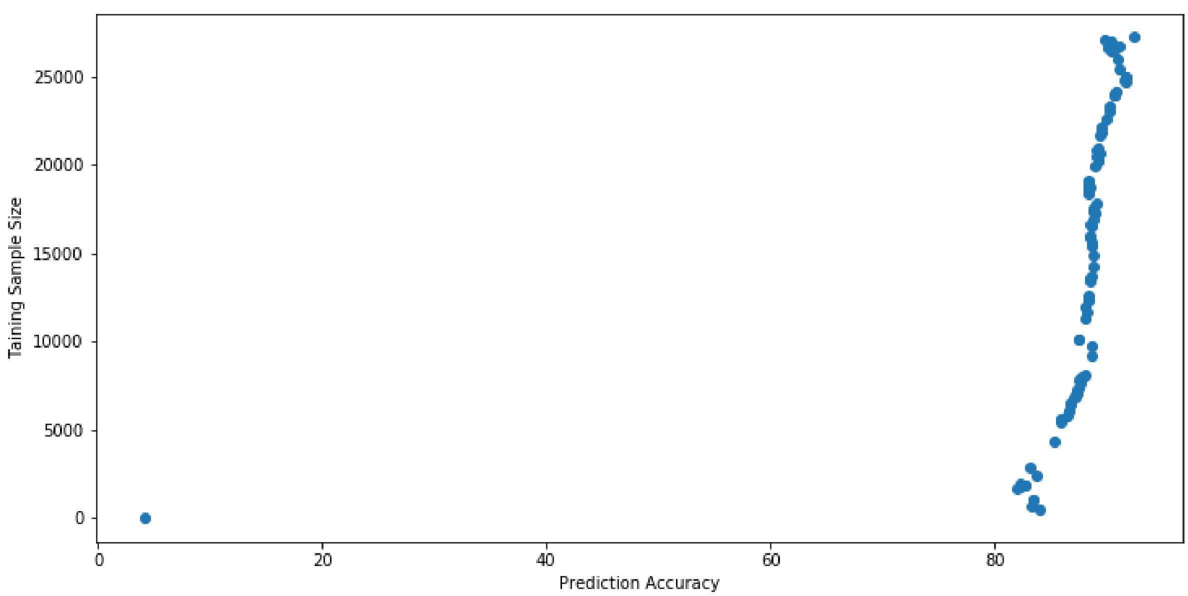
In [119]:

```
how_big_should_my_sample_be(10)
```



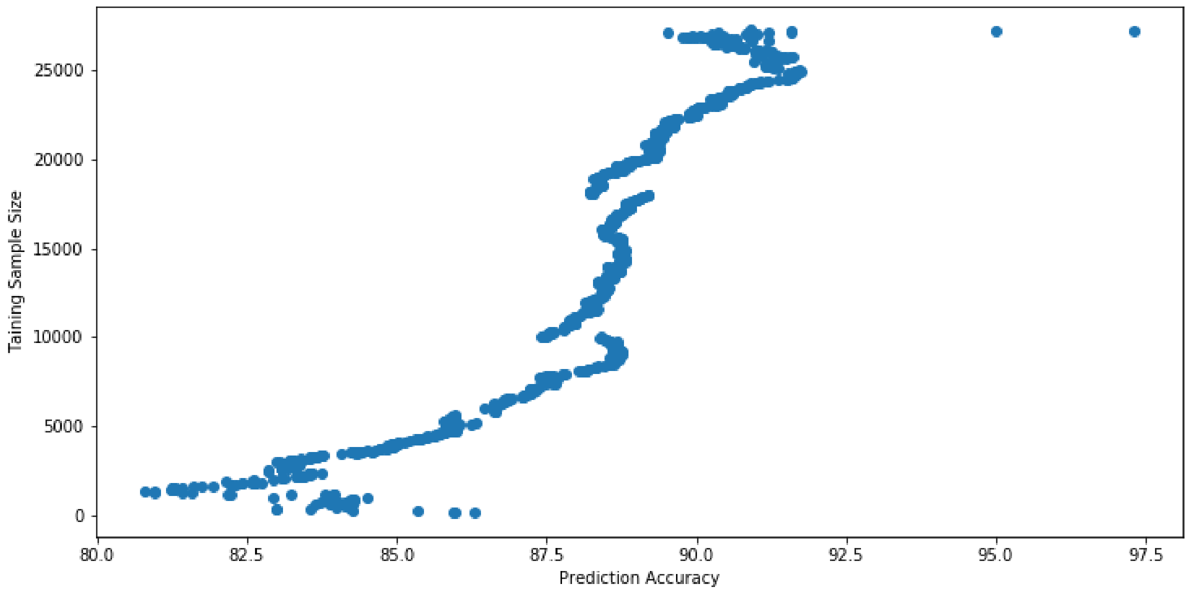
In [120]:

```
how_big_should_my_sample_be(100)
```



In [121]:

```
how_big_should_my_sample_be(1000)
```



In [122]:

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-122-b6142bf73f77> in <module>()
----> 1 how_big_should_my_sample_be(5000)

<ipython-input-117-8436a268a806> in how_big_should_my_sample_be(iterations)
      1 def how_big_should_my_sample_be(iterations):
----> 2     data = generate_data_for_(iterations)
      3     visualize_data(data)

<ipython-input-115-e771b36aad0d> in generate_data_for_(iterations)
      1 def generate_data_for_(iterations):
----> 2     raw_data=montecarlo_runner(iterations)
      3     serialized_data = list(zip(*raw_data))
      4     return serialized_data
      5 

<ipython-input-71-86355268ae3f> in montecarlo_runner(iterations)
      2     perf=[]
      3     for iteration in range(iterations):
----> 4         perf.append(montecarlo_classifier())
      5 
      6     return perf

<ipython-input-40-3ef488773873> in montecarlo_classifier()
      2 def montecarlo_classifier():
      3     split = random.randint(0,(len(target)-1))
----> 4     success_classifier = NaiveBayesClassifier.train(target[:split])
      5     accuracy = round(nltk.classify.util.accuracy(success_classifier, target[split:])*100,2)
      6     #print("{}{}{}".format("accurary: ",accuracy,"\n\n"))

c:\users\ebalgza\appdata\local\programs\python\python3\lib\site-packages\nltk\classify\naivebayes.py in train(cls, labeled_featuresets, estimator)
    199             for fname, fval in featureset.items():
    200                 # Increment freq(fval|label, fname)
--> 201                 feature_freqdist[label, fname][fval] += 1
    202                 # Record that fname can take the value fval.
    203                 feature_values[fname].add(fval)

c:\users\ebalgza\appdata\local\programs\python\python3\lib\site-packages\nltk\probability.py in __setitem__(self, key, val)
```

```
    130         """
    131         self._N = None
--> 132         super(FreqDist, self).__setitem__(key, val)
    133 
    134     def __delitem__(self, key):

KeyboardInterrupt: 
```

In [ ]:

```

```