Introduction

We will work on a dataset which consists of Women's fashion online shop reviews. This dataset contains a title, a review text, and whether the review author would recommend the product. We are trying to determine whether a reviewer will recommend a product or not based on review title and review.

First, we will take a look of the train dataset. We will use this set to fit our models.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('C:/Users/alex_/Desktop/college/DS630-MachineLear/hw5_data_train.csv')
dataset.head()
```

Out[1]:

	Title	Review	Recommended
0	Beautiful unique dress	Wore this to my sons wedding. found it last mi...	1
1	Had high hopes but...	Gals, if you absolutely must have this top wai...	0
2	Buttons -buttons so cute!	I hardly believe i have not reviewed htis yet....	1
3	Love this dress	This dress is so cute and comfortable. i'm nor...	1
4	Perfect summer pants!	The linen- cotton blend breathes so well for a...	1

In [2]:

```
dataset.isnull().values.any()
```

Out[2]:

```
True
```

It contains missing values. We will fill them out with space character to not lose the rows.

In [3]:

```
dataset.isnull().sum()
```

Out[3]:

```
Title          2852
Review          629
Recommended       0
dtype: int64
```

In [4]:

```
dataset=dataset.fillna(' ') #filling out missing values
```

In [5]:

```
dataset.isnull().values.any()
```

Out[5]:

```
False
```

False

Now, we can see how big is our dataset.

In [6]:

```
dataset.shape
```

Out[6]:

```
(17614, 3)
```

In [7]:

```
dataset[dataset['Recommended']==1].shape
```

Out[7]:

```
(14485, 3)
```

In [8]:

```
dataset[dataset['Recommended']==0].shape
```

Out[8]:

```
(3129, 3)
```

There's a big difference between the recommended products and not recommended products. We can infere we are working with a imbalanced text classification problem.

Now, we are going to read our test dataset to make the predictions using the models we create using the training set.

In [9]:

```
dataset_test = pd.read_csv('C:/Users/alex_/Desktop/college/DS630-MachineLear/hw5_data_test.csv')
dataset_test.head()
```

Out[9]:

	Title	Review	Recommended
0	NaN	NaN	0
1	Perfect pair!!	This pair of age stevie capris is everything t...	1
2	Talk about creature comforts!	This is a beautifully designed jacket that eve...	1
3	NaN	NaN	1
4	So comfotable	I ordered this dress in 0p since i am 5ft. it ...	1

In [10]:

```
dataset_test=dataset_test.fillna(' ') #filling out missing values
dataset_test.isnull().values.any()
```

Out[10]:

```
False
```

In [11]:

```
dataset_test.shape
```

Out[11]:

```
(5872, 3)
```

In [12]:

```
dataset_test[dataset_test['Recommended']==1].shape
```

Out[12]:

```
(4829, 3)
```

In [13]:

```
dataset_test[dataset_test['Recommended']==0].shape
```

Out[13]:

```
(1043, 3)
```

In [14]:

```
title_test=dataset_test.iloc[:,0].values
recom_test=dataset_test.iloc[:,2].values
```

Now, we are ready to make our machine learning models. We will try to experiment how different our models can be if we use different parts of our dataset. First, we will try to use only the title of our dataset and see how good can be to predict the product. Then, we will use the review body. After that, we will use both columns, title and review body (this would be the most recommended). Last one, we will concatenate the feature vectors of the title and review body and compare with the rest of the models how good (or bad) could be using this approach.

Using title

In [16]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegressionCV
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from joblib import Memory

location = './cachedir'
memory = Memory(location, verbose=10)
```

We are going to compare most frequent words in our models and the scores, to figure out which model is the best.

In [17]:

```
title=dataset.iloc[:,0].values
recom=dataset.iloc[:,2].values

vect_pipe = make_pipeline(CountVectorizer(),LogisticRegressionCV(),memory=memory)
vect_pipe.fit(title,recom)
predictions = vect_pipe.predict(title_test)

print ('Accuracy score is:',vect_pipe.score(title,recom))
print ('ROC AUC score is: ', roc_auc_score(recom, vect_pipe.predict(title)))
```

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Beautiful unique dress', ..., 'Beautiful, flowy,'], dtype=object), array([1, ..., 1], dtyp
e=int64), None)
_____________________________________________fit_transform_one - 0.1s, 0.0min
```

```
Accuracy score is: 0.9144998296809356
ROC AUC score is:  0.8040719215269143
```

In [18]:

```
vect=CountVectorizer()
title=dataset.iloc[:,0].values
recom=dataset.iloc[:,2].values
title_vect=vect.fit_transform(title)

lr=LogisticRegressionCV(cv=5).fit(title_vect,recom)

print ('Accuracy score is:',lr.score(title_vect,recom))
print ('ROC AUC score is: ', roc_auc_score(recom, lr.predict(title_vect)))
```

```
Accuracy score is: 0.9145566027023958
ROC AUC score is:  0.8041064399942944
```

If we compare both models, we are using a pipeline in the 'first model' with Logistic Regression and CountVectorizer. In the 'second model', we are using separate both functions. However, we obtained the same scores in both. This is what we want to show because these are different steps to obtain the same model. We need both steps because we need to show the most frequent words from the model and record the scores for a different dataset, in this case, test data.

In [19]:

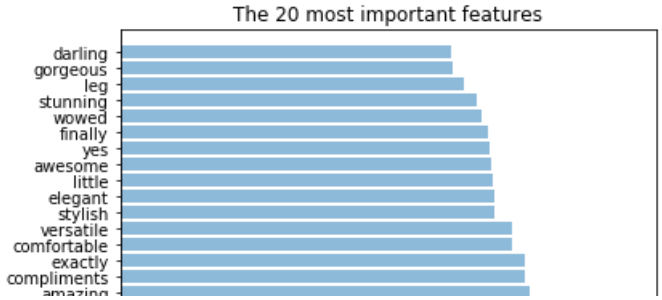
```
print ('Accuracy score is:',vect_pipe.score(title_test,recom_test))
print ('ROC AUC score is: ', roc_auc_score(recom_test, vect_pipe.predict(title_test)))
```

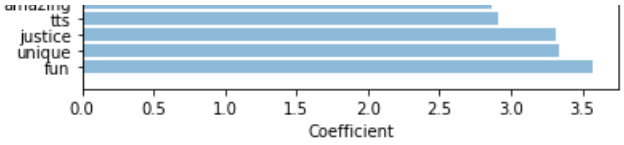
```
Accuracy score is: 0.8773841961852861
ROC AUC score is:  0.739406990404529
```

We need to make the predictions with another set of data to avoid overfitting. In this case, we see that using the training set we obtained scores of 0.91 and 0.8. However, using the test set, we obtained 0.87 and 0.74

In [20]:

```
coef_list=np.array(lr.coef_)
coef_list=-np.sort(-lr.coef_)

objects=[]
performance=[]

for i in range(20):
    position = np.where(coef_list[0][i]==np.array(lr.coef_).flatten())
    feat=vect.get_feature_names()[position[0][0]]
    objects.append(feat)
    performance.append(coef_list[0][i])
    #print('feature {}: {}'.format(i+1,feat))


y_pos = np.arange(len(objects))

plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Coefficient')
plt.title('The 20 most important features')

plt.show()
```



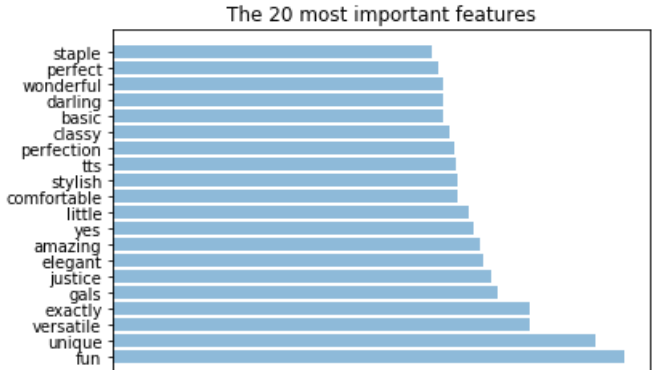


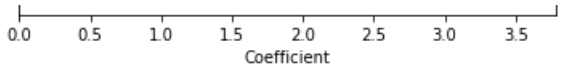
Let's try to tune our model. In this case, we are going to add the parameter stop_words.

In [21]:

```
vect=CountVectorizer(stop_words='english')
title_vect=vect.fit_transform(title)

lr=LogisticRegressionCV(cv=5).fit(title_vect,recom)

vect_pipe = make_pipeline(CountVectorizer(stop_words='english'),LogisticRegressionCV(),memory=memo
ry)
vect_pipe.fit(title,recom)
predictions = vect_pipe.predict(title_test)


coef_list=np.array(lr.coef_)
coef_list=-np.sort(-lr.coef_)

objects=[]
performance=[]

for i in range(20):
    position = np.where(coef_list[0][i]==np.array(lr.coef_).flatten())
    feat=vect.get_feature_names()[position[0][0]]
    objects.append(feat)
    performance.append(coef_list[0][i])
    #print('feature {}: {}'.format(i+1,feat))


y_pos = np.arange(len(objects))

plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Coefficient')
plt.title('The 20 most important features')

plt.show()


print ('Accuracy score is:',vect_pipe.score(title_test,recom_test))
print ('ROC AUC score is: ', roc_auc_score(recom_test, vect_pipe.predict(title_test)))
```

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words='english',
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Beautiful unique dress', ..., 'Beautiful, flowy,'], dtype=object), array([1, ..., 1], dtyp
e=int64), None)
_____________________________________________fit_transform_one - 0.1s, 0.0min
```





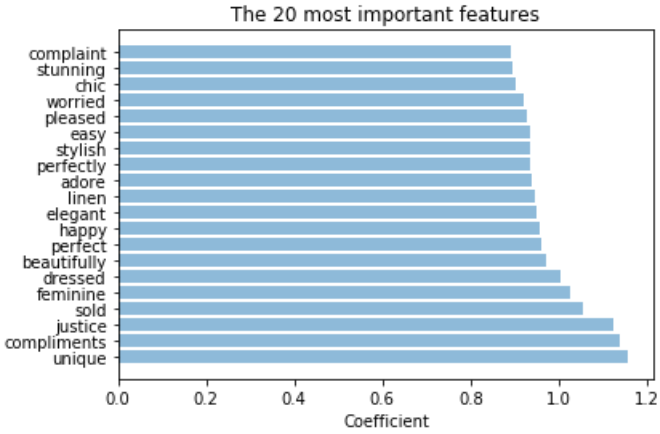

```
Accuracy score is: 0.8702316076294278
ROC AUC score is:  0.6970978907197587
```

We are using Accuracy and Roc Auc scores to see how good are our models. Have in mind we need roc auc score because we are working with a imbalanced dataset. In the table below, we can see the difference between the scores are not so different.

Using review body

In [22]:

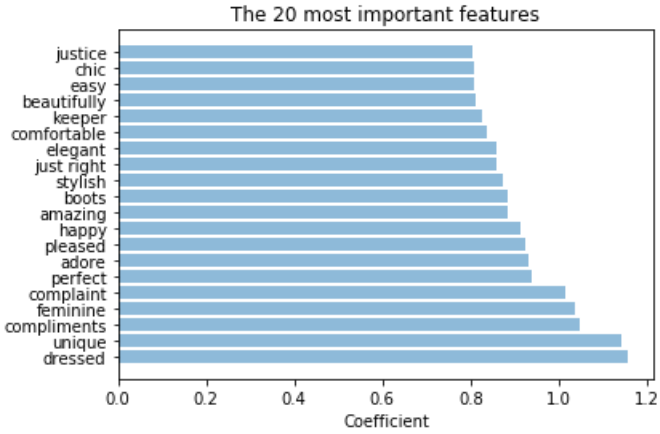
```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Wore this to my sons wedding. found it last minute when i had changed my mind about
another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'I fell in love with this shirt in the store and it has become one of my favorite items i
n my closet! the bright pink contrasts perfectly with the white peplum bottom and the fit is wo
nderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________________fit_transform_one - 0.9s, 0.0min
```




```
Accuracy score is: 0.8877724795640327
ROC AUC score is:  0.7784215371853536
```

In [23]:

```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=range(1, 3), preprocessor=None, stop_words='english',
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Wore this to my sons wedding. found it last minute when i had changed my mind about
another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'I fell in love with this shirt in the store and it has become one of my favorite items i
n my closet! the bright pink contrasts perfectly with the white peplum bottom and the fit is wo
nderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________________fit_transform_one - 2.4s, 0.0min
```




```
Accuracy score is: 0.8823228882833788
ROC AUC score is:  0.7574434936575862
```

There's no much difference between the models' scores

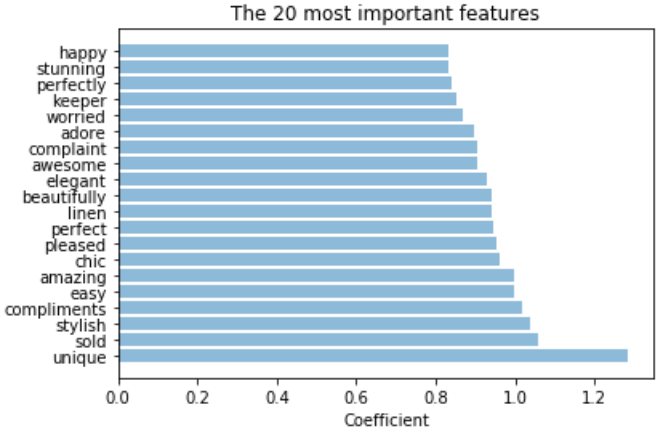
	Accuracy	Roc auc
Title model	0.88	0.73
Tunning Title model	0.87	0.70

Using title and review

In [24]:

In [25]:

```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________________fit_transform_one - 0.9s, 0.0min
```



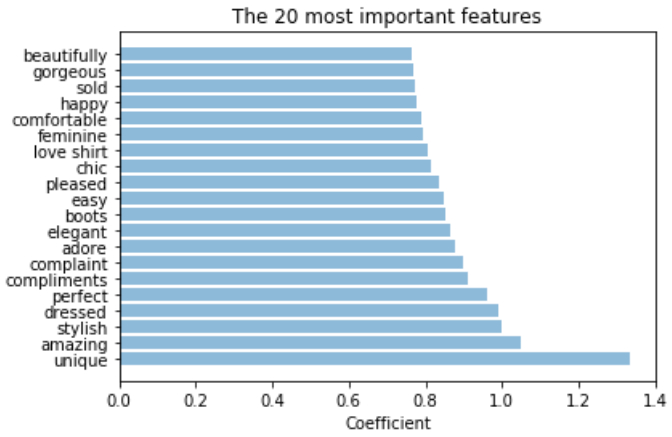

```
Accuracy score is: 0.8969686648501363
ROC AUC score is:  0.8013016397615318
```

```
ROC AUC score is:  0.8015016597615316
```

In [ ]:

In [26]:

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=range(1, 3), preprocessor=None, stop_words='english',
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 2.6s, 0.0min
```




```
Accuracy score is: 0.8956062670299727
ROC AUC score is:  0.7876945714083199
```

We see there is a slight difference between the two models.

	Accuracy	Roc auc
Review body model	0.88	0.77
Tunning review body model	0.88	0.76

Now, we are going to compare all model's scores to see which approach is the ideal to manage this dataset.

	Accuracy	Roc auc
Title & Review body model	0.90	0.80
Tunning Title & review body model	0.90	0.79

We can conclude that the model using the title and review body is the best among all, so we decide to use this for the rest of the problem.

Using Tf-dif

Next, we will introduce another method to obtain the feature matrix. We introduce Tfidf algorithm and we will compare this with the CountVectorizer we used before.

In [27]:

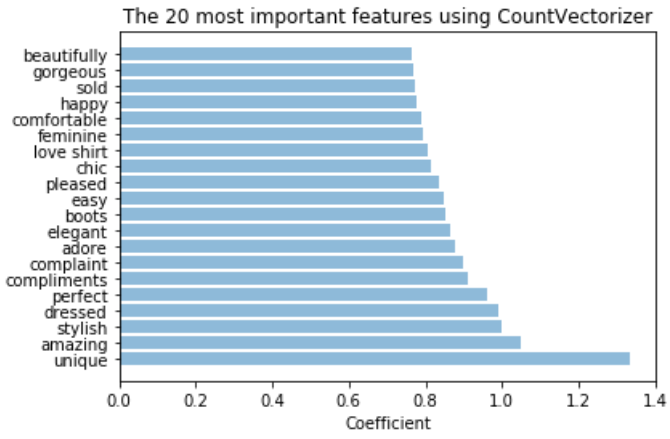
In [28]:

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm=None, preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 0.9s, 0.0min
Accuracy score is: 0.9000340599455041
ROC AUC score is:  0.8005344627090205
```

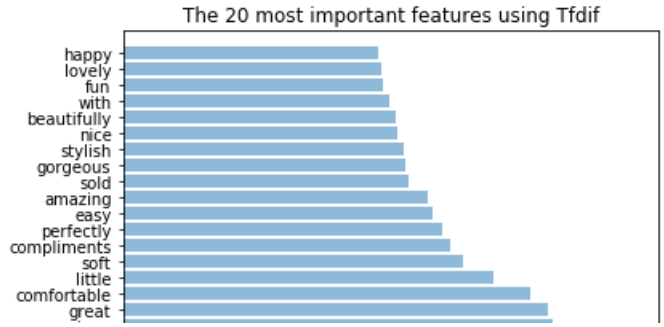
	Accuracy	Roc auc
Title model	0.88	0.73
Tunning Title model	0.87	0.70
Review body model	0.88	0.77
Tunning review body model	0.88	0.76
Title & Review body model	0.90	0.80
Tunning Title & review body model	0.90	0.79

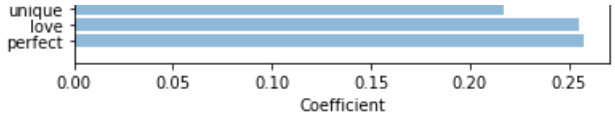
We see there's not much difference between both algorithms. Let's see now if the most important features of each model have changed.

In [29]:



In [30]:





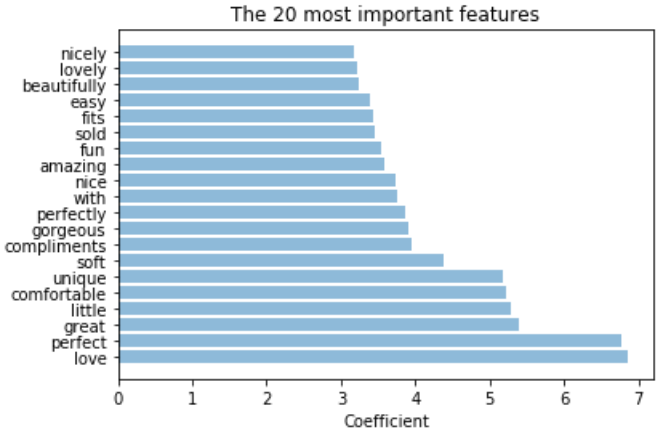
The most important features for each model have changed because they are using different approaches to obtain the feature matrix, so it's understandable these words are not equal.

Now, we are going to see how different is the model if we change some parameters. In this case, we are going to change the normalization parameter.

In [31]:

```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 1.0s, 0.0min
Accuracy score is: 0.9017370572207084
ROC AUC score is:  0.7955563492934883
```

In [32]:



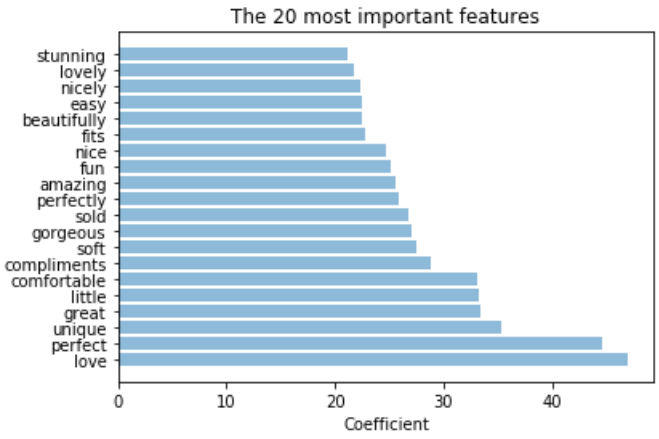
Here, we are changing the normalization parameter from l2 to l1

In [33]:

```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
```

```
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 1.0s, 0.0min
Accuracy score is: 0.9036103542234333
ROC AUC score is:  0.8015812900923968
```

In [34]:



	Accuracy	Roc auc
Title & review body model using CountVectorizer	0.90	0.80
Title & Review body model using tfidf	0.90	0.80

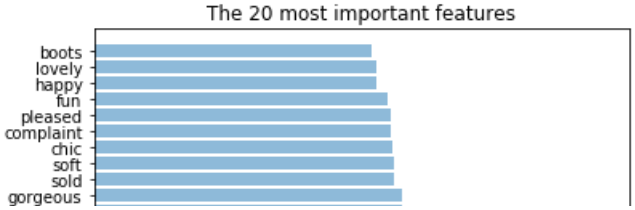
In the table above, we see that there's not big difference between the normalization parameters we used for each model. Also, if we compare the most frequent words of each model, we see that these words doesn't change at all ('perfect', 'love', 'unique', 'great' are included in every model). Now, we are going to try to change some parameters of tfdif function to see if they have an impact in the results of our models.

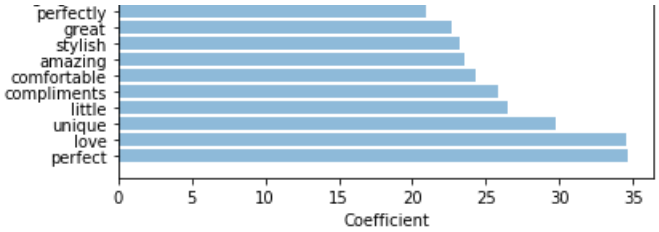
Here, we are going to use the parameters stop_words, which is going to exclude the most frequent words in our corpus. i.e. 'the', 'a','an', among others.

In [35]:

```
_____________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 0.8s, 0.0min
Accuracy score is: 0.8959468664850136
ROC AUC score is:  0.787901653619958
```

In [36]:





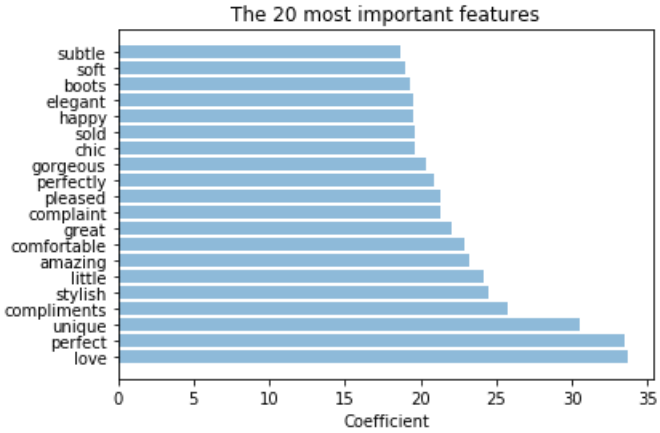
The graphic above with the most important features looks similar to the ones using normalization feature (it contains 'perfect', 'love', 'unique'). Also, the scores are similar to the other models. We don't see an impact using stop_words, but we need to try to use the stop_words parameters in our model as a good practice.

There's other parameters we need to know to obtain a better score in our model. These parameters are min_df and max_df.

In [37]:

```
_____________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=(1, 1), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 0.7s, 0.0min
Accuracy score is: 0.8956062670299727
ROC AUC score is:  0.7888221072471427
```

In [38]:



We see there's not much difference between the scores and the most important features of this model comparing with the previous models. However, we need to have in mind that in this case, our feature matrix was reduced of size. In conclusion, our model is running faster with this approach than the others.

In [39]:

```
Dimension of vocabulary: (17614, 13010)
Dimension of vocabulary using normalization type l1: (17614, 13010)
Dimension of vocabulary using normalization type l2: (17614, 13010)
Dimension of vocabulary using stop words: (17614, 12727)
Dimension of vocabulary using min_df=4: (17614, 4706)
```

Above we compared the sizes of each feature matrices. In this case, we reduced the size of the matrix 35%. This method can help us with the computation time if we use a bigger dataset than this one.

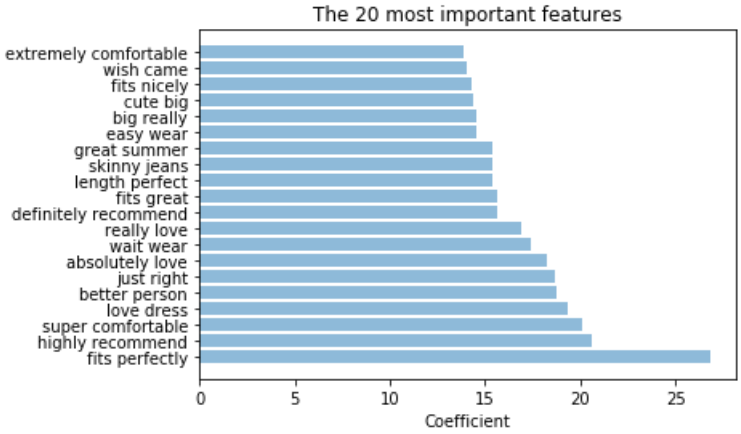
Importance of ngram

Now, we are going to introduce another parameter of tfdid function. This is ngram_range, which will allow us to use n-grams. N-grams is another way to detect language, or when syntax rules are not being followed, is using n-gram based text categorization.

In [40]:

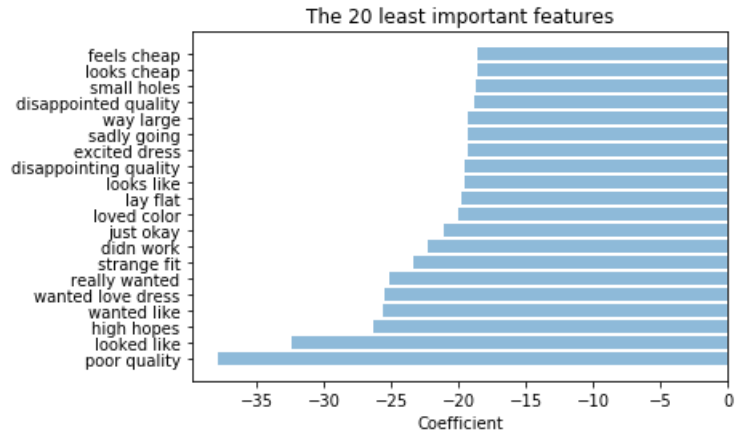
```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=(2, 4), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 8.0s, 0.1min
Accuracy score is: 0.8598433242506812
ROC AUC score is:  0.7073190755675354
```

In [41]:



This is the best model until now, with good scores of Accuracy (0.99) and ROC AUC score (0.97). Watching the most frequent words of this model, we realize that every word makes sense with the prediction. Let's take a look the worst words of this model.

In [42]:



All the words showing above in the graphic, are from the products which are not recommended for people. We can see it makes sense that people complain with the products and pointing out in the review to advice people don't buy these products

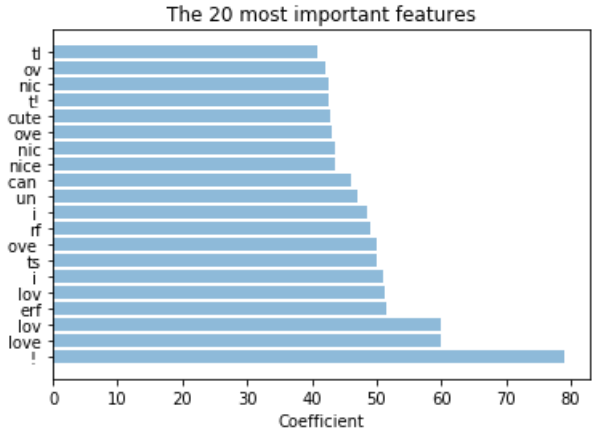
sense that people complain with the products and pointing out in the review to advice people don't buy these products.

Now, in the same way we take out certain amount of words to make the categorization, we can use certain number of characters belongs to a paragraph. This approach is named char n-grams, which will be using in the following lines.

In [43]:

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='char_wb', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=(2, 4), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 8.0s, 0.1min
Accuracy score is: 0.9019073569482289
ROC AUC score is:  0.7945323545604843
```

In [44]:



After using char n-grams, we can see this is a good apporach to predict our dataset. We can see the result of the most important features, we can see the char '!' expressing admiration or astonishment for certain products, that means this would be a recommendation to buy these products. Also, we can see the characters of love on it. This means people usually use this words to recommend the products people like.

Tunning the predictive models

Now, we are going to tune our models to obtain better accuracy and roc auc scores.

In [45]:

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
```

```
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 18.1s, 0.3min
Best score: 0.908992846599296
Best parameters set: {'tfidf__ngram_range': (1, 2), 'tfidf__norm': 'l1'}
```

In [46]:

```
_____________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=4,
        ngram_range=(1, 2), norm='l1', preprocessor=None, smooth_idf=True,
        stop_words='english', strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None),
array(['Beautiful unique dress Wore this to my sons wedding. found it last minute when i had ch
anged my mind about another byron lars dress. i felt awesome in it and got many compliments',
       ...,
       'Beautiful, flowy, I fell in love with this shirt in the store and it has become one of
my favorite items in my closet! the bright pink contrasts perfectly with the white peplum
bottom and the fit is wonderfully flattering!'],
      dtype=object),
array([1, ..., 1], dtype=int64), None)
_____________________________________________fit_transform_one - 7.3s, 0.1min
Accuracy score is: 0.8978201634877384
ROC AUC score is:  0.7965575113761199
```

There's no difference between this scores and the other we made previously. Remember that we are using an imbalanced dataset to fit our models, so it's too difficult to obtain good scores using this dataset. It would be better if we have equal amount of reviews of products that people like and dislike.

In [ ]:

	Accuracy	Roc auc
Tfdif approach without normalization	0.900	0.800
Tfdif approach with normalization l2	0.901	0.795
Tfdif approach with normalization l1	0.903	0.800