Hw5

April 16, 2018

```
In [206]: import warnings
          warnings.filterwarnings('ignore')
          %matplotlib inline
          import numpy as np
          import matplotlib.pyplot as plt
          import pandas as pd
          from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer, TfidfTran
          from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
          from sklearn.pipeline import make_pipeline, Pipeline
          from sklearn.preprocessing import Normalizer

          from sklearn.metrics import f1_score ,average_precision_score ,roc_auc_score
          from sklearn.model_selection import train_test_split
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

0.1 Task 1 Title and Body (30Pts)

Use CountVectorizer with the default settings and train a linear classifier. Visualize the 20 most important features in the linear model. Tune the regularization parameter of the classifier, and visualize the 20 most important features after regularization. Do this for all 4 settings. Which one works best? For the simplicity, for the remaining tasks, we will work with option 3), concatenating the texts.

```
In [212]: train_main = pd.read_csv('hw5_data_train.csv')
          test_main = pd.read_csv('hw5_data_test.csv')
```

```
In [213]: train, test, y_train,y_test = train_test_split(train_main,train_main["Recommended"] ,
```

```
In [ ]:
```

```
In [ ]:
```

```
In [216]: print('train = ',train.shape,'\ntest = ',test.shape)
```

```
train =  (13210, 3)
test =  (4404, 3)
```

In [217]:
```
train.head()
```

Out[217]:
```
                                Title  \
14278  Beautiful sweater, but pills
7912                       Looooove!!
5783               A lovely basic top
14539         Pretty dress, poor fit
2704                       Beautiful!

                                                  Review  Recommended
14278  I love the look of this sweater and was so hap...            0
7912   I normally don't write reviews for clothing bu...            1
5783   I'm between sizes in deletta (m to l, 5'4", 34...            1
14539  I really wanted to love this dress, but it jus...            0
2704   I wore this dress to a july wedding in miami b...            1
```

In [218]:
```
test.head()
```

Out[218]:
```
                      Title  \
8234         Fits perfectly
7811       Comfy and stylish
13470                  Nope!
2745   Wish i could keep it!
2603              Super cute

                                                  Review  Recommended
8234   I'm small--5'1" and 90 lbs--and the petite xxs...            1
7811   I love this vest and could not leave without i...            1
13470  Weird fit and the tie just add to the problems...            0
2745   So this dress is absolutely gorgeous! my probl...            1
2603   Super cute and flow. mine had a bit of stitchi...            1
```

In [220]:
```
print(train['Recommended'].unique())
print(test['Recommended'].unique())
```

```
[0 1]
[1 0]
```

1) Use the title only

In [282]:
```
train1_clean = train[['Title','Recommended']].dropna(thresh=2).reset_index(drop=True)
test1_clean = test[['Title','Recommended']].dropna(thresh=2).reset_index(drop=True)

test_main_clean = test_main.dropna(thresh=3).reset_index(drop=True)

print(train1_clean.shape,'\n',test1_clean.shape)
```

```
(11088, 2)
 (3674, 2)
```

In [ ]:

In [222]:
```
train1_clean.head()
```

Out[222]:
```
                          Title  Recommended
0  Beautiful sweater, but pills            0
1                   Looooove!!             1
2             A lovely basic top           1
3         Pretty dress, poor fit           0
4                   Beautiful!             1
```

In [223]:
```
#default params
vect = CountVectorizer()
X_train1 = vect.fit_transform(train1_clean['Title'])
y_train1 = train1_clean['Recommended']
X_test1 = vect.transform(test1_clean['Title'])
y_test1 = test1_clean['Recommended']
feature_names1 = vect.get_feature_names()
logreg1 = LogisticRegression()
logreg1.fit(X_train1,y_train1)
logreg1.score(X_test1,y_test1)
```

Out[223]:
```
0.8889493739793141
```

In [224]:
```
print("Test Avg Precision score: ", average_precision_score(logreg1.predict(X_test1),y
print("Test F1 score: ", f1_score(logreg1.predict(X_test1),y_test1))
print("Test ROC AUC score: ", roc_auc_score(logreg1.predict(X_test1),y_test1))
```

```
Test Avg Precision score:  0.9564875099151535
Test F1 score:  0.9342571704801805
Test ROC AUC score:  0.8421410925134545
```

In [225]:
```
logreg1.coef_[0]
```

Out[225]:
```
array([0.27596701, 0.16014591, 0.01771065, ..., 0.21856947, 0.27167001,
       0.20306462])
```

In [228]:
```
colour = []
for i in range(40):
    if i < 20:
        colour.append("red")
    else:
        colour.append("blue")
def plot(coef,feature_names,i): #plots top 20 features
```

```
    top20_index_pos = coef.argsort()[-20:] #top 20 coefficient index values from logre
    top20_pos = coef[top20_index_pos]
    print (top20_pos)
    top20_names_pos = [feature_names[j] for j in top20_index_pos] #gets top20 words fr
    print(top20_names_pos)
    top20_index_neg = coef.argsort()[:20] #top 20 coefficient index values from logreg
    top20_neg = coef[top20_index_neg]
    print (top20_neg)
    top20_names_neg = [feature_names[j] for j in top20_index_neg] #gets top20 words fr
    print(top20_names_neg)
    top_coef = np.hstack([top20_neg,top20_pos])
    print(top_coef)
    top_names = np.hstack([top20_names_neg,top20_names_pos])
    print(top_names)

    plt.figure(figsize=(20, 5))
    plt.bar(range(1,41),top_coef,color=colour)
    plt.title('20 most important features '+str(i))
    plt.xticks(range(1,41),top_names,rotation=45)
    plt.show()
```

In [ ]:

In [240]:
```
colour = []
for i in range(40):
    if i < 20:
        colour.append("red")
    else:
        colour.append("blue")

def plot(coef,feature_names,i): #plots top 20 features
    top20_index_pos = coef.argsort()[-20:] #top 20 coefficient index values from logre
    top20_pos = coef[top20_index_pos]
    print (top20_pos)
    top20_names_pos = [feature_names[j] for j in top20_index_pos] #gets top20 words fr
    print(top20_names_pos)
    top20_index_neg = coef.argsort()[:20] #top 20 coefficient index values from logreg
    top20_neg = coef[top20_index_neg]
    print (top20_neg)
    top20_names_neg = [feature_names[j] for j in top20_index_neg] #gets top20 words fr
    print(top20_names_neg)
    top_coef = np.hstack([top20_neg,top20_pos])
    print(top_coef)
    top_names = np.hstack([top20_names_neg,top20_names_pos])
    print(top_names)

    plt.figure(figsize=(20, 5))
    plt.bar(range(1,41),top_coef,color=colour)
```

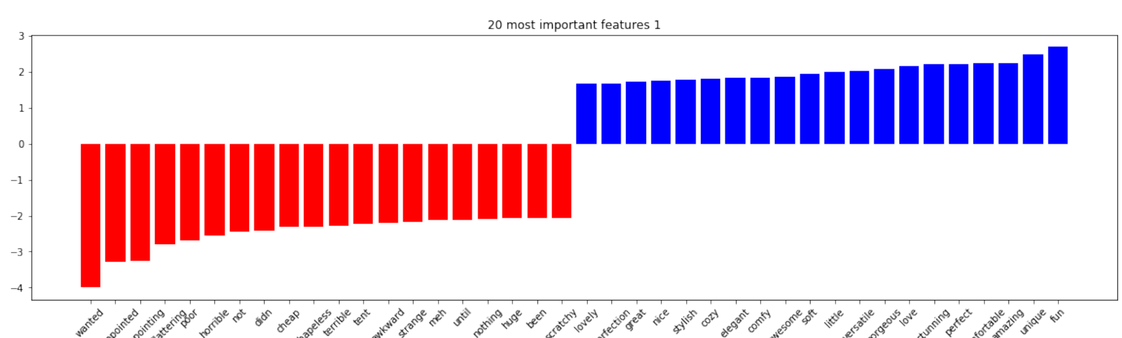
```
            plt.title('20 most important features '+str(i))
            plt.xticks(range(1,41),top_names,rotation=45)
            plt.show()
```

In [ ]:

In [ ]:

In [229]:
```
plot(logreg1.coef_[0],feature_names1,1)
```

```
[1.66463256 1.67440413 1.71419128 1.75805566 1.77722984 1.81044084
 1.83124505 1.83219752 1.86497013 1.93376101 2.00758922 2.02198025
 2.06683447 2.17071126 2.20057276 2.21417034 2.22679172 2.24397146
 2.47225981 2.69236594]
['lovely', 'perfection', 'great', 'nice', 'stylish', 'cozy', 'elegant', 'comfy', 'awesome', 'sof
[-3.9989159  -3.2961309  -3.25396528 -2.8097402  -2.67558442 -2.54559009
 -2.44165471 -2.40508654 -2.32055246 -2.31859306 -2.29101087 -2.22834341
 -2.19625909 -2.17344314 -2.12000815 -2.1179435  -2.08526583 -2.07812922
 -2.07290846 -2.07105653]
['wanted', 'disappointed', 'disappointing', 'unflattering', 'poor', 'horrible', 'not', 'didn', '
[-3.9989159  -3.2961309  -3.25396528 -2.8097402  -2.67558442 -2.54559009
 -2.44165471 -2.40508654 -2.32055246 -2.31859306 -2.29101087 -2.22834341
 -2.19625909 -2.17344314 -2.12000815 -2.1179435  -2.08526583 -2.07812922
 -2.07290846 -2.07105653  1.66463256  1.67440413  1.71419128  1.75805566
  1.77722984  1.81044084  1.83124505  1.83219752  1.86497013  1.93376101
  2.00758922  2.02198025  2.06683447  2.17071126  2.20057276  2.21417034
  2.22679172  2.24397146  2.47225981  2.69236594]
['wanted' 'disappointed' 'disappointing' 'unflattering' 'poor' 'horrible'
 'not' 'didn' 'cheap' 'shapeless' 'terrible' 'tent' 'awkward' 'strange'
 'meh' 'until' 'nothing' 'huge' 'been' 'scratchy' 'lovely' 'perfection'
 'great' 'nice' 'stylish' 'cozy' 'elegant' 'comfy' 'awesome' 'soft'
 'little' 'versatile' 'gorgeous' 'love' 'stunning' 'perfect' 'comfortable'
 'amazing' 'unique' 'fun']
```



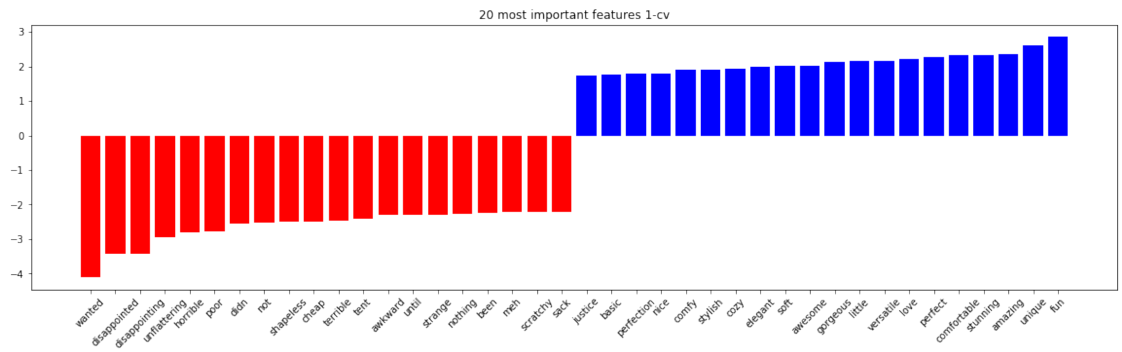
```
In [230]: #with CV
          vect = CountVectorizer()
          X_train1_cv = vect.fit_transform(train1_clean['Title'])
          y_train1_cv = train1_clean['Recommended']
          X_test1_cv = vect.transform(test1_clean['Title'])
          y_test1_cv = test1_clean['Recommended']
          feature_names1_cv = vect.get_feature_names()
          logreg1_cv = LogisticRegressionCV(Cs = [1,1.1,1.15,1.2,1.23,1.4],scoring = 'average_pr
          logreg1_cv.fit(X_train1_cv,y_train1_cv)
          print('test score:',logreg1_cv.score(X_test1_cv,y_test1_cv))

          print('C:',logreg1_cv.C_)

test score: 0.8889493739793141
C: [1.23]
```

```
In [231]: print("Test Avg Precision score: ", average_precision_score(logreg1_cv.predict(X_test1
          print("Test F1 score: ", f1_score(logreg1_cv.predict(X_test1_cv),y_test1_cv))
          print("Test ROC AUC score: ", roc_auc_score(logreg1_cv.predict(X_test1_cv),y_test1_cv)

Test Avg Precision score:  0.9551710272803298
Test F1 score:  0.9341723136495644
Test ROC AUC score:  0.840316778891862
```

```
In [232]: logreg1_cv.coef_[0]

Out[232]: array([0.34462075, 0.17922977, 0.01750281, ..., 0.26596794, 0.32177213,
                 0.23960715])
```

```
In [233]: plot(logreg1_cv.coef_[0],feature_names1_cv,'1-cv')

[1.74352062 1.77249804 1.79521947 1.80256515 1.8893198  1.91567424
 1.93261461 1.98483418 2.00304402 2.01059131 2.12877915 2.14619127
 2.14764808 2.21881955 2.27181736 2.31771738 2.32495123 2.35729232
 2.61895288 2.85236125]
['justice', 'basic', 'perfection', 'nice', 'comfy', 'stylish', 'cozy', 'elegant', 'soft', 'aweso
[-4.11040095 -3.4213091  -3.41771885 -2.93319747 -2.7947699  -2.77534732
 -2.54866309 -2.52184706 -2.50337964 -2.48922629 -2.46038333 -2.41933928
 -2.30812801 -2.28674925 -2.28441501 -2.27353251 -2.25200404 -2.22403701
 -2.22144981 -2.1987102 ]
['wanted', 'disappointed', 'disappointing', 'unflattering', 'horrible', 'poor', 'didn', 'not', '
[-4.11040095 -3.4213091  -3.41771885 -2.93319747 -2.7947699  -2.77534732
 -2.54866309 -2.52184706 -2.50337964 -2.48922629 -2.46038333 -2.41933928
 -2.30812801 -2.28674925 -2.28441501 -2.27353251 -2.25200404 -2.22403701
 -2.22144981 -2.1987102   1.74352062  1.77249804  1.79521947  1.80256515
  1.8893198   1.91567424  1.93261461  1.98483418  2.00304402  2.01059131
  2.12877915  2.14619127  2.14764808  2.21881955  2.27181736  2.31771738
```

```
 2.32495123  2.35729232  2.61895288  2.85236125]
['wanted' 'disappointed' 'disappointing' 'unflattering' 'horrible' 'poor'
 'didn' 'not' 'shapeless' 'cheap' 'terrible' 'tent' 'awkward' 'until'
 'strange' 'nothing' 'been' 'meh' 'scratchy' 'sack' 'justice' 'basic'
 'perfection' 'nice' 'comfy' 'stylish' 'cozy' 'elegant' 'soft' 'awesome'
 'gorgeous' 'little' 'versatile' 'love' 'perfect' 'comfortable' 'stunning'
 'amazing' 'unique' 'fun']
```



In [ ]:

In [ ]:

In [ ]:

2) Use the review body only

In [234]:
```
train2_clean = train[['Review','Recommended']].dropna(thresh=2).reset_index(drop=True)
test2_clean = test[['Review','Recommended']].dropna(thresh=2).reset_index(drop=True)
print(train2_clean.shape,'\n',test2_clean.shape)
```

```
(12734, 2)
 (4251, 2)
```

In [235]:
```
train2_clean.head()
```

Out[235]:

	Review	Recommended
0	I love the look of this sweater and was so hap...	0
1	I normally don't write reviews for clothing bu...	1
2	I'm between sizes in deletta (m to l, 5'4", 34...	1
3	I really wanted to love this dress, but it jus...	0
4	I wore this dress to a july wedding in miami b...	1

```
In [236]: #default params
          vect = CountVectorizer()
          X_train2 = vect.fit_transform(train2_clean['Review'])
          y_train2 = train2_clean['Recommended']
          X_test2 = vect.transform(test2_clean['Review'])
          y_test2 = test2_clean['Recommended']
          feature_names2 = vect.get_feature_names()
          logreg2 = LogisticRegression()
          logreg2.fit(X_train2,y_train2)
          print('test score:',logreg2.score(X_test2,y_test2))
```

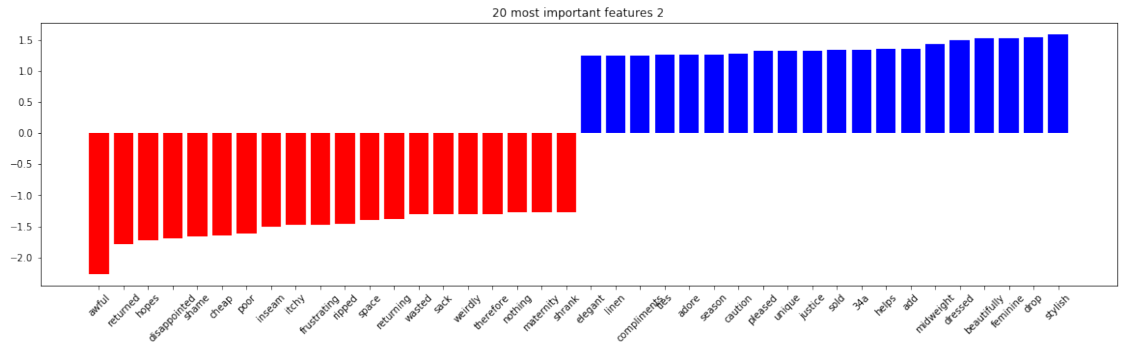
```
test score: 0.8882615855092919
```

```
In [237]: print("Test Avg Precision score: ", average_precision_score(logreg2.predict(X_test2),y
          print("Test F1 score: ", f1_score(logreg2.predict(X_test2),y_test2))
          print("Test ROC AUC score: ", roc_auc_score(logreg2.predict(X_test2),y_test2))
```

```
Test Avg Precision score:  0.9341045965080008
Test F1 score:  0.9324996447349723
Test ROC AUC score:  0.8170661768271983
```

```
In [238]: plot(logreg2.coef_[0],feature_names2,2)
```

```
[1.2393148  1.24229168 1.25202254 1.25734585 1.26059808 1.2671636
 1.27185852 1.31514321 1.3161773  1.3186311  1.33631546 1.34278663
 1.35642357 1.35660737 1.42777173 1.48908307 1.5209158  1.5258742
 1.54369813 1.57857957]
['elegant', 'linen', 'compliments', 'ties', 'adore', 'season', 'caution', 'pleased', 'unique', '
[-2.25986287 -1.78987905 -1.72961233 -1.68914445 -1.66549184 -1.64404591
 -1.61637488 -1.50511167 -1.4814577  -1.470046   -1.45186148 -1.39687668
 -1.38801472 -1.31011519 -1.30157348 -1.30095187 -1.2968461  -1.27071003
 -1.26682786 -1.26530928]
['awful', 'returned', 'hopes', 'disappointed', 'shame', 'cheap', 'poor', 'inseam', 'itchy', 'fru
[-2.25986287 -1.78987905 -1.72961233 -1.68914445 -1.66549184 -1.64404591
 -1.61637488 -1.50511167 -1.4814577  -1.470046   -1.45186148 -1.39687668
 -1.38801472 -1.31011519 -1.30157348 -1.30095187 -1.2968461  -1.27071003
 -1.26682786 -1.26530928  1.2393148   1.24229168  1.25202254  1.25734585
  1.26059808  1.2671636   1.27185852  1.31514321  1.3161773   1.3186311
  1.33631546  1.34278663  1.35642357  1.35660737  1.42777173  1.48908307
  1.5209158   1.5258742   1.54369813  1.57857957]
['awful' 'returned' 'hopes' 'disappointed' 'shame' 'cheap' 'poor' 'inseam'
 'itchy' 'frustrating' 'ripped' 'space' 'returning' 'wasted' 'sack'
 'weirdly' 'therefore' 'nothing' 'maternity' 'shrank' 'elegant' 'linen'
 'compliments' 'ties' 'adore' 'season' 'caution' 'pleased' 'unique'
 'justice' 'sold' '34a' 'helps' 'add' 'midweight' 'dressed' 'beautifully'
 'feminine' 'drop' 'stylish']
```




```
In [239]: #with CV
          vect = CountVectorizer()
          X_train2_cv = vect.fit_transform(train2_clean['Review'])
          y_train2_cv = train2_clean['Recommended']
          X_test2_cv = vect.transform(test2_clean['Review'])
          y_test2_cv = test2_clean['Recommended']
          feature_names2_cv = vect.get_feature_names()
          logreg2_cv = LogisticRegressionCV(scoring = 'average_precision')
          logreg2_cv.fit(X_train2_cv,y_train2_cv)
          print('test score:',logreg2_cv.score(X_test2_cv,y_test2_cv))
          print('C:',logreg2_cv.C_)
```

```
test score: 0.8852034815337567
C: [0.04641589]
```

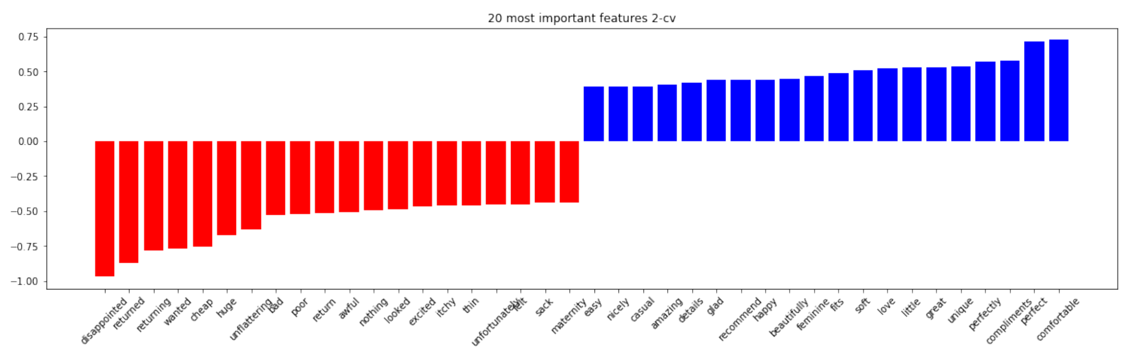
```
In [241]: print("Test Avg Precision score: ", average_precision_score(logreg2_cv.predict(X_test2
          print("Test F1 score: ", f1_score(logreg2_cv.predict(X_test2_cv),y_test2_cv))
          print("Test ROC AUC score: ", roc_auc_score(logreg2_cv.predict(X_test2_cv),y_test2_cv)
```

```
Test Avg Precision score:  0.9528468646552953
Test F1 score:  0.9320712694877505
Test ROC AUC score:  0.8301479336232371
```

```
In [242]: plot(logreg2_cv.coef_[0],feature_names2_cv,'2-cv')
```

```
[0.39227269 0.39231976 0.39317555 0.4022602  0.42184413 0.43743482
 0.44004033 0.44208536 0.44730537 0.46357951 0.48977441 0.50757709
 0.52239993 0.52681328 0.53163335 0.53510764 0.57205353 0.57338671
 0.7165083  0.7263525 ]
['easy', 'nicely', 'casual', 'amazing', 'details', 'glad', 'recommend', 'happy', 'beautifully',
[-0.97049427 -0.8736004  -0.7823568  -0.77052624 -0.75534608 -0.67518825
 -0.6343559  -0.52871938 -0.52363371 -0.51539049 -0.51135659 -0.49137049
 -0.48634917 -0.46613203 -0.45853969 -0.45810765 -0.45193783 -0.45010544
```

```
 -0.44118251 -0.4409696 ]
['disappointed', 'returned', 'returning', 'wanted', 'cheap', 'huge', 'unflattering', 'bad', 'poo
[-0.97049427 -0.8736004  -0.7823568  -0.77052624 -0.75534608 -0.67518825
 -0.6343559  -0.52871938 -0.52363371 -0.51539049 -0.51135659 -0.49137049
 -0.48634917 -0.46613203 -0.45853969 -0.45810765 -0.45193783 -0.45010544
 -0.44118251 -0.4409696   0.39227269  0.39231976  0.39317555  0.4022602
  0.42184413  0.43743482  0.44004033  0.44208536  0.44730537  0.46357951
  0.48977441  0.50757709  0.52239993  0.52681328  0.53163335  0.53510764
  0.57205353  0.57338671  0.7165083   0.7263525 ]
['disappointed' 'returned' 'returning' 'wanted' 'cheap' 'huge'
 'unflattering' 'bad' 'poor' 'return' 'awful' 'nothing' 'looked' 'excited'
 'itchy' 'thin' 'unfortunately' 'felt' 'sack' 'maternity' 'easy' 'nicely'
 'casual' 'amazing' 'details' 'glad' 'recommend' 'happy' 'beautifully'
 'feminine' 'fits' 'soft' 'love' 'little' 'great' 'unique' 'perfectly'
 'compliments' 'perfect' 'comfortable']
```



3) Concatenate the title and review to a single text and analyze that (discarding the information which words were in the title and which in the body)

In [243]:
```
train3 = pd.DataFrame()
train3['text'] = train['Title']+' '+train["Review"]
train3['Recommended'] = train['Recommended']
train3_clean = train3.dropna(thresh=2).reset_index(drop=True)
train3_clean.head()
```

Out[243]:

	text	Recommended
0	Beautiful sweater, but pills I love the look o...	0
1	Looooove!! I normally don't write reviews for ...	1
2	A lovely basic top I'm between sizes in delett...	1
3	Pretty dress, poor fit I really wanted to love...	0
4	Beautiful! I wore this dress to a july wedding...	1

In [244]:
```
test3 = pd.DataFrame()
test3['text'] = test['Title']+' '+test["Review"]
```

```
test3['Recommended'] = test['Recommended']
test3_clean = test3.dropna(thresh=2).reset_index(drop=True)
test3_clean.head()
```

Out[244]:

	text	Recommended
0	Fits perfectly I'm small--5'1" and 90 lbs--and...	1
1	Comfy and stylish I love this vest and could n...	1
2	Nope! Weird fit and the tie just add to the pr...	0
3	Wish i could keep it! So this dress is absolut...	1
4	Super cute Super cute and flow. mine had a bit...	1

In [245]:
```
print(train3_clean.shape)
print(test3_clean.shape)
```

```
(11088, 2)
(3674, 2)
```

In [246]:
```
#default params
vect = CountVectorizer()
X_train3 = vect.fit_transform(train3_clean['text'])
y_train3 = train3_clean['Recommended']
X_test3 = vect.transform(test3_clean['text'])
y_test3 = test3_clean['Recommended']
feature_names3 = vect.get_feature_names()
logreg3 = LogisticRegression()
logreg3.fit(X_train3,y_train3)
print('test score:',logreg3.score(X_test3,y_test3))
```

```
test score: 0.8987479586281981
```

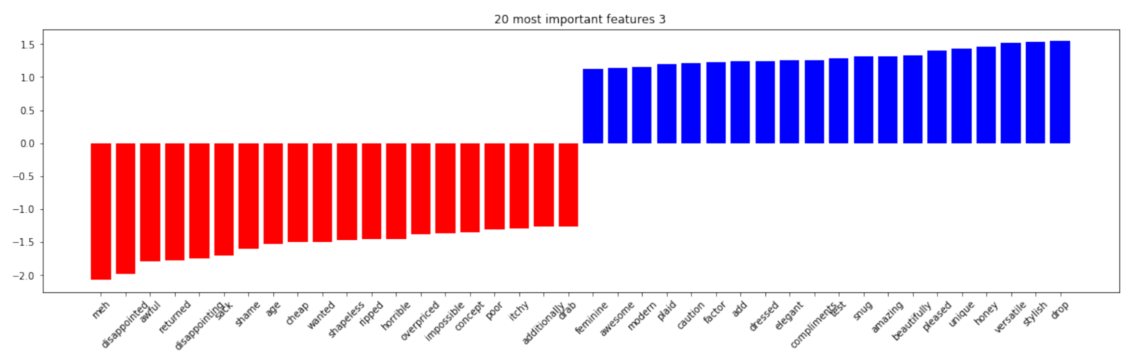
In [247]:
```
print("Test Avg Precision score: ", average_precision_score(logreg3.predict(X_test3),y
print("Test F1 score: ", f1_score(logreg3.predict(X_test3),y_test3))
print("Test ROC AUC score: ", roc_auc_score(logreg3.predict(X_test3),y_test3))
```

```
Test Avg Precision score:  0.9371516922946084
Test F1 score:  0.9384920634920635
Test ROC AUC score:  0.8348492798797058
```

In [248]:
```
plot(logreg3.coef_[0],feature_names3,3)
```

```
[1.12256214 1.13961049 1.1466177  1.19613308 1.21601869 1.22846537
 1.23764544 1.23821301 1.25500414 1.2570197  1.2841668  1.311923
 1.31934487 1.32858872 1.40812575 1.43277772 1.45557793 1.52251214
 1.53183634 1.54330323]
['feminine', 'awesome', 'modern', 'plaid', 'caution', 'factor', 'add', 'dressed', 'elegant', 'co
[-2.07321059 -1.98627844 -1.79370344 -1.77119123 -1.74060929 -1.70896474
```

```
 -1.60067504 -1.52309089 -1.49444797 -1.49102174 -1.47037495 -1.45994296
 -1.44883983 -1.38282882 -1.36940433 -1.34367213 -1.30009994 -1.29115809
 -1.26234493 -1.25688021]
['meh', 'disappointed', 'awful', 'returned', 'disappointing', 'sack', 'shame', 'age', 'cheap', '
[-2.07321059 -1.98627844 -1.79370344 -1.77119123 -1.74060929 -1.70896474
 -1.60067504 -1.52309089 -1.49444797 -1.49102174 -1.47037495 -1.45994296
 -1.44883983 -1.38282882 -1.36940433 -1.34367213 -1.30009994 -1.29115809
 -1.26234493 -1.25688021  1.12256214  1.13961049  1.1466177   1.19613308
  1.21601869  1.22846537  1.23764544  1.23821301  1.25500414  1.2570197
  1.2841668   1.311923    1.31934487  1.32858872  1.40812575  1.43277772
  1.45557793  1.52251214  1.53183634  1.54330323]
['meh' 'disappointed' 'awful' 'returned' 'disappointing' 'sack' 'shame'
 'age' 'cheap' 'wanted' 'shapeless' 'ripped' 'horrible' 'overpriced'
 'impossible' 'concept' 'poor' 'itchy' 'additionally' 'drab' 'feminine'
 'awesome' 'modern' 'plaid' 'caution' 'factor' 'add' 'dressed' 'elegant'
 'compliments' 'test' 'snug' 'amazing' 'beautifully' 'pleased' 'unique'
 'honey' 'versatile' 'stylish' 'drop']
```



In [249]:
```python
#with CV
vect = CountVectorizer()
X_train3_cv = vect.fit_transform(train3_clean['text'])
y_train3_cv = train3_clean['Recommended']
X_test3_cv = vect.transform(test3_clean['text'])
y_test3_cv = test3_clean['Recommended']
feature_names3_cv = vect.get_feature_names()
logreg3_cv = LogisticRegressionCV(scoring = 'average_precision')
logreg3_cv.fit(X_train3_cv,y_train3_cv)
print('test score:',logreg3_cv.score(X_test3_cv,y_test3_cv))
print('C:',logreg3_cv.C_)
```

```
test score: 0.9009254218835057
C: [0.04641589]
```

```
In [276]: X_train3_cv
```

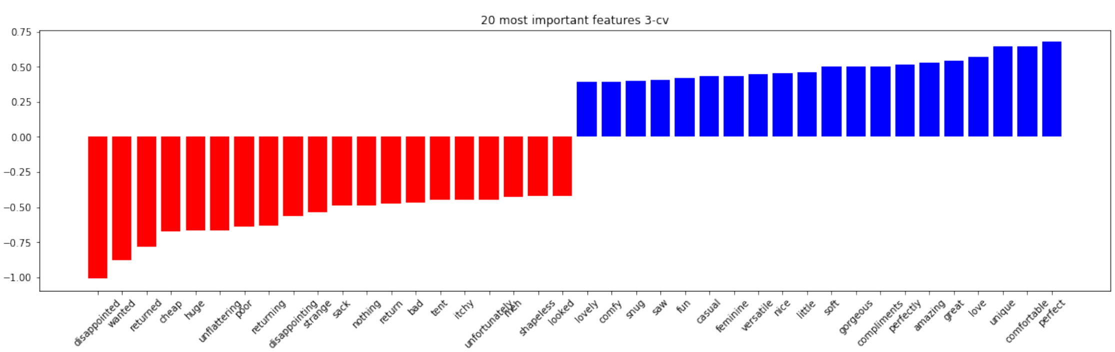
```
Out[276]: <11088x11095 sparse matrix of type '<class 'numpy.int64'>'
                  with 503030 stored elements in Compressed Sparse Row format>
```

```
In [250]: print("Test Avg Precision score: ", average_precision_score(logreg3_cv.predict(X_test3
          print("Test F1 score: ", f1_score(logreg3_cv.predict(X_test3_cv),y_test3_cv))
          print("Test ROC AUC score: ", roc_auc_score(logreg3_cv.predict(X_test3_cv),y_test3_cv)
```

```
Test Avg Precision score:  0.9541062096841407
Test F1 score:  0.9407744874715261
Test ROC AUC score:  0.854449519543325
```

```
In [251]: plot(logreg3_cv.coef_[0],feature_names3_cv,'3-cv')
```

```
[0.38882906 0.39418194 0.4015464  0.40508181 0.41908435 0.43452645
 0.43527615 0.44807764 0.4553095  0.46329898 0.49910003 0.50196928
 0.50245886 0.51296674 0.53099663 0.54481952 0.57144502 0.64233278
 0.64236304 0.67721892]
['lovely', 'comfy', 'snug', 'saw', 'fun', 'casual', 'feminine', 'versatile', 'nice', 'little', '
[-1.00966378 -0.87745027 -0.78369513 -0.67058447 -0.66835424 -0.66366222
 -0.64172107 -0.63130027 -0.56432058 -0.53874603 -0.49015367 -0.48644914
 -0.47441307 -0.46812074 -0.44773847 -0.44509114 -0.44480412 -0.42951192
 -0.42333524 -0.42331551]
['disappointed', 'wanted', 'returned', 'cheap', 'huge', 'unflattering', 'poor', 'returning', 'di
[-1.00966378 -0.87745027 -0.78369513 -0.67058447 -0.66835424 -0.66366222
 -0.64172107 -0.63130027 -0.56432058 -0.53874603 -0.49015367 -0.48644914
 -0.47441307 -0.46812074 -0.44773847 -0.44509114 -0.44480412 -0.42951192
 -0.42333524 -0.42331551  0.38882906  0.39418194  0.4015464   0.40508181
  0.41908435  0.43452645  0.43527615  0.44807764  0.4553095   0.46329898
  0.49910003  0.50196928  0.50245886  0.51296674  0.53099663  0.54481952
  0.57144502  0.64233278  0.64236304  0.67721892]
['disappointed' 'wanted' 'returned' 'cheap' 'huge' 'unflattering' 'poor'
 'returning' 'disappointing' 'strange' 'sack' 'nothing' 'return' 'bad'
 'tent' 'itchy' 'unfortunately' 'meh' 'shapeless' 'looked' 'lovely'
 'comfy' 'snug' 'saw' 'fun' 'casual' 'feminine' 'versatile' 'nice'
 'little' 'soft' 'gorgeous' 'compliments' 'perfectly' 'amazing' 'great'
 'love' 'unique' 'comfortable' 'perfect']
```



4) Vectorizing title and review individually and concatenating the vector representations.

In [252]:
```
train4_clean = train[['Title',"Review",'Recommended']].dropna(thresh=3).reset_index(dr
test4_clean = test[['Title',"Review",'Recommended']].dropna(thresh=3).reset_index(drop
print(train4_clean.shape,'\n',test4_clean.shape)


vect4_title = CountVectorizer()
vect4_rec = CountVectorizer()

X_train4_title = vect4_title.fit_transform(train4_clean["Title"])
X_train4_rev = vect4_rec.fit_transform(train4_clean["Review"])


y_train4 = train4_clean['Recommended']
```

```
(11088, 3)
 (3674, 3)
```

In [253]:
```
X_test4_title = vect4_title.transform(test4_clean["Title"])
X_test4_rev = vect4_rec.transform(test4_clean["Review"])
y_test4 = test4_clean["Recommended"]
```

In [254]:
```
X_train4_rev
```

Out[254]:
```
<11088x10691 sparse matrix of type '<class 'numpy.int64'>'
	with 484229 stored elements in Compressed Sparse Row format>
```

In [255]:
```
from scipy.sparse import hstack
X_train4 = hstack([X_train4_title,X_train4_rev])
X_test4 = hstack([X_test4_title,X_test4_rev])
```

In [256]:
```
X_train4
```

Out[256]:
```
<11088x13509 sparse matrix of type '<class 'numpy.int64'>'
	with 519943 stored elements in COOrdinate format>
```

In [257]:
```
X_test4
```

Out[257]:
```
<3674x13509 sparse matrix of type '<class 'numpy.int64'>'
	with 172413 stored elements in COOrdinate format>
```

In [258]:
```
feature_names4_rec = vect4_rec.get_feature_names()
feature_names4_title = vect4_title.get_feature_names()
logreg4 = LogisticRegression()
logreg4.fit(X_train4,y_train4)
logreg4.score(X_test4,y_test4)
```

```
Out[258]: 0.902286336418073
```

```
In [259]: print("Test Avg Precision score: ", average_precision_score(logreg4.predict(X_test4),y
          print("Test F1 score: ", f1_score(logreg4.predict(X_test4),y_test4))
          print("Test ROC AUC score: ", roc_auc_score(logreg4.predict(X_test4),y_test4))
```

```
Test Avg Precision score:  0.9411466437440666
Test F1 score:  0.9407492985641195
Test ROC AUC score:  0.8425094037513344
```

```
In [260]: logreg4_cv = LogisticRegressionCV(cv =5, scoring = 'average_precision')
          logreg4_cv.fit(X_train4,y_train4)
          print('test score:',logreg4_cv.score(X_test4,y_test4))
          print('C:',logreg4_cv.C_)
```

```
test score: 0.9044637996733805
C: [0.35938137]
```

```
In [261]: print("Test Avg Precision score: ", average_precision_score(logreg4_cv.predict(X_test4
          print("Test F1 score: ", f1_score(logreg4_cv.predict(X_test4),y_test4))
          print("Test ROC AUC score: ", roc_auc_score(logreg4_cv.predict(X_test4),y_test4))
```

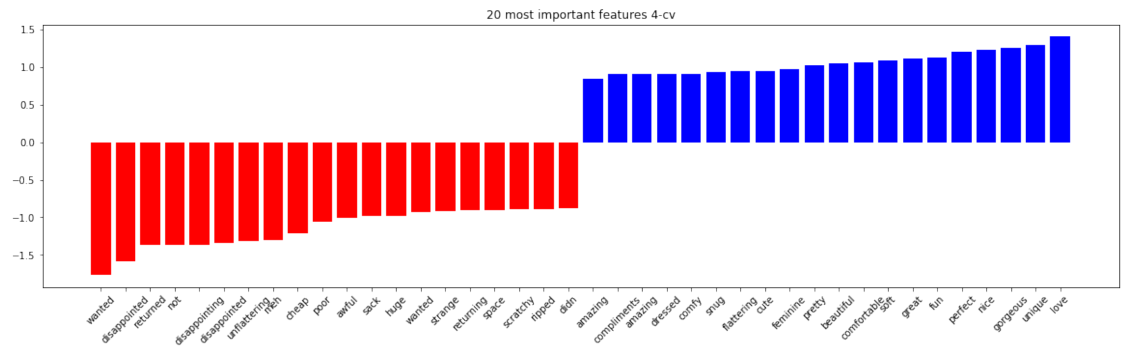
```
Test Avg Precision score:  0.9472617964318942
Test F1 score:  0.9423550665133849
Test ROC AUC score:  0.8508956401744107
```

```
In [262]: feature_names4 = feature_names4_title + feature_names4_rec
```

```
In [263]: plot(logreg4_cv.coef_[0],feature_names4,'4-cv')
```

```
[0.84589754 0.9034575  0.90392548 0.91047887 0.91061117 0.93253227
 0.9519536  0.95338408 0.97296504 1.02365254 1.05601754 1.0682746
 1.08442476 1.11198334 1.12889363 1.20993541 1.23032413 1.26158283
 1.2905034  1.40680532]
['amazing', 'compliments', 'amazing', 'dressed', 'comfy', 'snug', 'flattering', 'cute', 'feminin
[-1.77107997 -1.58985726 -1.36710404 -1.36568999 -1.36421356 -1.34393638
 -1.32129964 -1.30165293 -1.21575726 -1.05696592 -1.0022613  -0.9808555
 -0.97849909 -0.93018708 -0.91623128 -0.90892549 -0.89946732 -0.88713166
 -0.88637835 -0.88356472]
['wanted', 'disappointed', 'returned', 'not', 'disappointing', 'disappointed', 'unflattering', '
[-1.77107997 -1.58985726 -1.36710404 -1.36568999 -1.36421356 -1.34393638
 -1.32129964 -1.30165293 -1.21575726 -1.05696592 -1.0022613  -0.9808555
 -0.97849909 -0.93018708 -0.91623128 -0.90892549 -0.89946732 -0.88713166
 -0.88637835 -0.88356472  0.84589754  0.9034575   0.90392548  0.91047887
  0.91061117  0.93253227  0.9519536   0.95338408  0.97296504  1.02365254
  1.05601754  1.0682746   1.08442476  1.11198334  1.12889363  1.20993541
```

```
 1.23032413  1.26158283  1.2905034   1.40680532]
['wanted' 'disappointed' 'returned' 'not' 'disappointing' 'disappointed'
 'unflattering' 'meh' 'cheap' 'poor' 'awful' 'sack' 'huge' 'wanted'
 'strange' 'returning' 'space' 'scratchy' 'ripped' 'didn' 'amazing'
 'compliments' 'amazing' 'dressed' 'comfy' 'snug' 'flattering' 'cute'
 'feminine' 'pretty' 'beautiful' 'comfortable' 'soft' 'great' 'fun'
 'perfect' 'nice' 'gorgeous' 'unique' 'love']
```



In [264]: `np.mean(y_train4)`

Out[264]: 0.8193542568542569

We notice that the 3rd case where title and text and combined gives the best performance in terms of average precision score (95%) and roc auc(0.85). We will now evaluate this model on the main test set

In [285]:
```
test_main_clean["text"] = test_main_clean['Title']+' '+test_main_clean["Review"]
```

In [ ]:

In [287]:
```
X_test_4 = vect.transform(test_main_clean["text"])
```

In [288]:
```
X_test_4
```

Out[288]:
```
<4913x11095 sparse matrix of type '<class 'numpy.int64'>'
	with 222993 stored elements in Compressed Sparse Row format>
```

In [273]:
```
test_main[["Title","Review"]].shape
```

Out[273]: (5872, 2)

In [ ]:

Task 1 best model performance:

```
In [291]: print("Test Avg Precision score: ", average_precision_score(logreg3_cv.predict(X_test_
          print("Test F1 score: ", f1_score(logreg3_cv.predict(X_test_4),test_main_clean["Recomm
          print("Test ROC AUC score: ", roc_auc_score(logreg3_cv.predict(X_test_4),test_main_cle
```

```
Test Avg Precision score:  0.9466903289063802
Test F1 score:  0.9355626067854528
Test ROC AUC score:  0.8358385328039326
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

0.2 Task 2 Feature Tuning (30Pts)

2.1 Try using TfidfVectorizer instead of CountVectorizer. Does it change the score? Does it change the important coefficients?

```
In [292]: #default params
          tfidf = TfidfVectorizer()
          X_train_tfidf = tfidf.fit_transform(train3_clean['text'])
          y_train_tfidf = train3_clean['Recommended']
          X_test_tfidf = tfidf.transform(test3_clean['text'])
          y_test_tfidf = test3_clean['Recommended']
          feature_names_tfidf = tfidf.get_feature_names()
          logreg_tfidf = LogisticRegression()
          logreg_tfidf.fit(X_train_tfidf,y_train_tfidf)
          print('test score:',logreg_tfidf.score(X_test_tfidf,y_test_tfidf))
```

```
test score: 0.8938486663037561
```

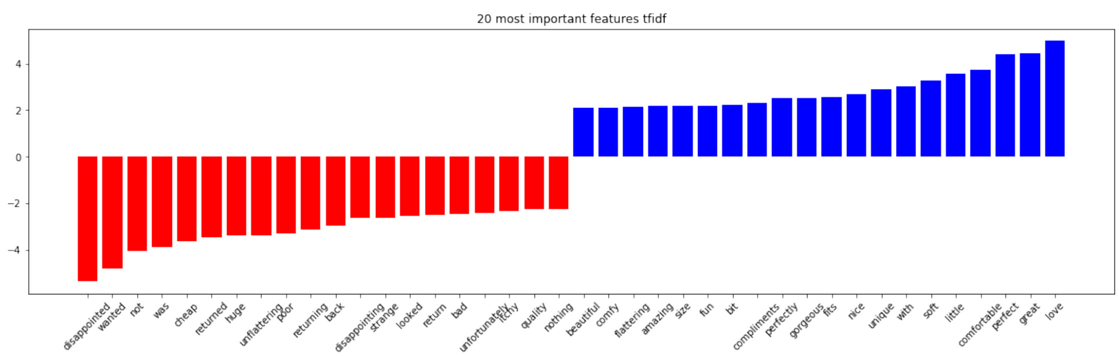
```
In [293]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf.predict(X_tes
          print("Test F1 score: ", f1_score(logreg_tfidf.predict(X_test_tfidf),y_test_tfidf))
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf.predict(X_test_tfidf),y_test_
```

```
Test Avg Precision score:  0.9627569989591834
Test F1 score:  0.9373594603276582
Test ROC AUC score:  0.8584488016769711
```

```
In [294]: plot(logreg_tfidf.coef_[0],feature_names_tfidf,'tfidf')
```

```
[2.09152769 2.10717629 2.16282323 2.18138223 2.1866902  2.19299446
 2.23788403 2.29356988 2.49877849 2.53011144 2.56983854 2.67352853
 2.91574297 3.03577341 3.26645044 3.57812594 3.72868027 4.38837031
```

```
 4.44994965 4.97742583]
['beautiful', 'comfy', 'flattering', 'amazing', 'size', 'fun', 'bit', 'compliments', 'perfectly'
[-5.35462342 -4.78297705 -4.03122464 -3.86028034 -3.61485673 -3.44714257
 -3.38903103 -3.38131006 -3.31073098 -3.11817191 -2.96757603 -2.62881655
 -2.62461881 -2.55234671 -2.48326092 -2.43687796 -2.42412503 -2.34249973
 -2.25186984 -2.24767947]
['disappointed', 'wanted', 'not', 'was', 'cheap', 'returned', 'huge', 'unflattering', 'poor', 'r
[-5.35462342 -4.78297705 -4.03122464 -3.86028034 -3.61485673 -3.44714257
 -3.38903103 -3.38131006 -3.31073098 -3.11817191 -2.96757603 -2.62881655
 -2.62461881 -2.55234671 -2.48326092 -2.43687796 -2.42412503 -2.34249973
 -2.25186984 -2.24767947  2.09152769  2.10717629  2.16282323  2.18138223
  2.1866902   2.19299446  2.23788403  2.29356988  2.49877849  2.53011144
  2.56983854  2.67352853  2.91574297  3.03577341  3.26645044  3.57812594
  3.72868027  4.38837031  4.44994965  4.97742583]
['disappointed' 'wanted' 'not' 'was' 'cheap' 'returned' 'huge'
 'unflattering' 'poor' 'returning' 'back' 'disappointing' 'strange'
 'looked' 'return' 'bad' 'unfortunately' 'itchy' 'quality' 'nothing'
 'beautiful' 'comfy' 'flattering' 'amazing' 'size' 'fun' 'bit'
 'compliments' 'perfectly' 'gorgeous' 'fits' 'nice' 'unique' 'with' 'soft'
 'little' 'comfortable' 'perfect' 'great' 'love']
```



In [295]:
```python
#with CV
tfidf = TfidfVectorizer()
X_train_tfidf_cv = tfidf.fit_transform(train3_clean['text'])
y_train_tfidf_cv = train3_clean['Recommended']
X_test_tfidf_cv = tfidf.transform(test3_clean['text'])
y_test_tfidf_cv = test3_clean['Recommended']
feature_names_tfidf_cv = tfidf.get_feature_names()
logreg_tfidf_cv = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf_cv.fit(X_train_tfidf_cv,y_train_tfidf_cv)
print('test score:',logreg_tfidf_cv.score(X_test_tfidf_cv,y_test_tfidf_cv))
print('C:',logreg_tfidf_cv.C_)
```

```
test score: 0.9066412629286881
```

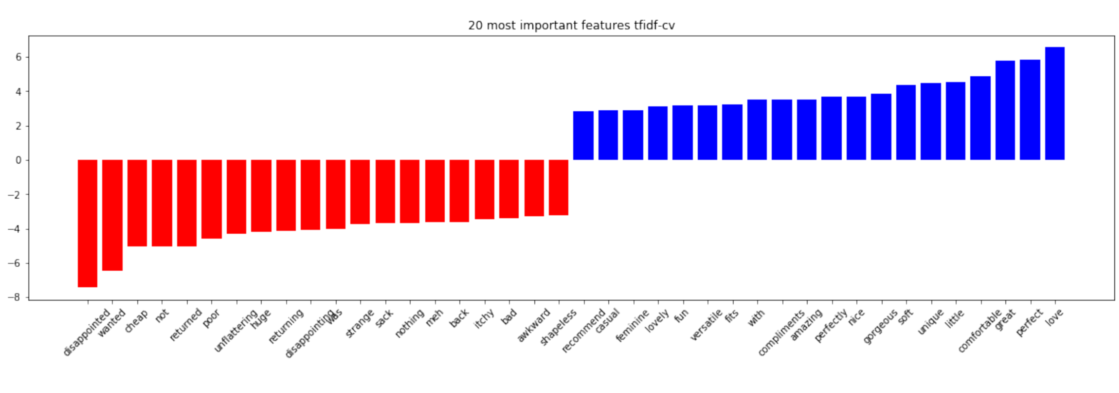
```
C: [2.7825594]
```

```
In [296]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf_cv.predict(X_
          print("Test F1 score: ", f1_score(logreg_tfidf_cv.predict(X_test_tfidf_cv),y_test_tfid
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf_cv.predict(X_test_tfidf_cv),y
```

```
Test Avg Precision score:  0.9572133245805011
Test F1 score:  0.9441640892072277
Test ROC AUC score:  0.8653414919249027
```

```
In [297]: plot(logreg_tfidf_cv.coef_[0],feature_names_tfidf_cv,'tfidf-cv')
```

```
[2.8381242  2.86257931 2.90621465 3.09881628 3.15788326 3.1664422
 3.23634753 3.4999815  3.51882053 3.52843952 3.67633199 3.70215825
 3.86628888 4.36136156 4.5000131  4.55031686 4.85982095 5.76411508
 5.81387123 6.55973747]
['recommend', 'casual', 'feminine', 'lovely', 'fun', 'versatile', 'fits', 'with', 'compliments',
[-7.45171713 -6.46594383 -5.06196461 -5.03119636 -5.02742239 -4.57719896
 -4.29440509 -4.20535607 -4.12687403 -4.09306881 -4.04566223 -3.75300874
 -3.67194077 -3.66169901 -3.65471698 -3.64504234 -3.46504659 -3.40386488
 -3.2887141  -3.24527281]
['disappointed', 'wanted', 'cheap', 'not', 'returned', 'poor', 'unflattering', 'huge', 'returnin
[-7.45171713 -6.46594383 -5.06196461 -5.03119636 -5.02742239 -4.57719896
 -4.29440509 -4.20535607 -4.12687403 -4.09306881 -4.04566223 -3.75300874
 -3.67194077 -3.66169901 -3.65471698 -3.64504234 -3.46504659 -3.40386488
 -3.2887141  -3.24527281  2.8381242   2.86257931  2.90621465  3.09881628
  3.15788326  3.1664422   3.23634753  3.4999815   3.51882053  3.52843952
  3.67633199  3.70215825  3.86628888  4.36136156  4.5000131   4.55031686
  4.85982095  5.76411508  5.81387123  6.55973747]
['disappointed' 'wanted' 'cheap' 'not' 'returned' 'poor' 'unflattering'
 'huge' 'returning' 'disappointing' 'was' 'strange' 'sack' 'nothing' 'meh'
 'back' 'itchy' 'bad' 'awkward' 'shapeless' 'recommend' 'casual'
 'feminine' 'lovely' 'fun' 'versatile' 'fits' 'with' 'compliments'
 'amazing' 'perfectly' 'nice' 'gorgeous' 'soft' 'unique' 'little'
 'comfortable' 'great' 'perfect' 'love']
```



```
In [298]: print ('With CountVectorizer:')
          print('Test Score with Linear Classifier: ',logreg3.score(X_test3,y_test3))
          print('Test Score with Linear Classifier and CV: ',logreg3_cv.score(X_test3_cv,y_test3
          print('With TfidfVectorizer:')
          print('Test Score with Linear Classifier: ',logreg_tfidf.score(X_test_tfidf,y_test_tfi
          print('Test Score with Linear Classifier and CV: ',logreg_tfidf_cv.score(X_test_tfidf_
```

```
With CountVectorizer:
Test Score with Linear Classifier:  0.8987479586281981
Test Score with Linear Classifier and CV:  0.9009254218835057
With TfidfVectorizer:
Test Score with Linear Classifier:  0.8938486663037561
Test Score with Linear Classifier and CV:  0.9066412629286881
```

Tfidf Vectorizer leads to a sizeable improvement in avg precision, roc auc and f1 scores. Also, it has imrpoved test set scores slightly

```
In [81]: coef3 = coefs(logreg3.coef_[0],feature_names3)
         coef3_cv = coefs(logreg3_cv.coef_[0],feature_names3_cv)
         coef_tfidf = coefs(logreg_tfidf.coef_[0],feature_names_tfidf)
         coef_tfidf_cv = coefs(logreg_tfidf_cv.coef_[0],feature_names_tfidf_cv)
         coef_changed = list(set(coef_tfidf)-set(coef3))
         coef_changed_cv = list(set(coef_tfidf_cv)-set(coef3_cv))
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-81-644846c3602c> in <module>()
    ----> 1 coef3 = coefs(logreg3.coef_[0],feature_names3)
          2 coef3_cv = coefs(logreg3_cv.coef_[0],feature_names3_cv)
          3 coef_tfidf = coefs(logreg_tfidf.coef_[0],feature_names_tfidf)
          4 coef_tfidf_cv = coefs(logreg_tfidf_cv.coef_[0],feature_names_tfidf_cv)
          5 coef_changed = list(set(coef_tfidf)-set(coef3))


        NameError: name 'coefs' is not defined
```

```
In [82]: print('By using TfidfVectorizer instead of CountVectorizer')
         print("{} Important Coefficients are changed using Linear Classifier. They are :\n{}".
               format(len(coef_changed), coef_changed))
         print("{} Important Coefficients are changed using Linear Classifier CV. They are :\n{}
               format(len(coef_changed_cv), coef_changed_cv))
```

```
By using TfidfVectorizer instead of CountVectorizer
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-82-90eb054cadab> in <module>()
      1 print('By using TfidfVectorizer instead of CountVectorizer')
      2 print("{} Important Coefficients are changed using Linear Classifier. They are :\n{}
----> 3       format(len(coef_changed), coef_changed))
      4 print("{} Important Coefficients are changed using Linear Classifier CV. They are :\
      5       format(len(coef_changed_cv), coef_changed_cv))


NameError: name 'coef_changed' is not defined
```

2.2 Remember that TfidfVectorizer uses normalization by default. Does using a Normalizer with CountVectorizer change the outcome?

```
In [299]: #default params
          pipe = Pipeline([("vect", CountVectorizer()),("norm", Normalizer())])
          X_train_vect_norm = pipe.fit_transform(train3_clean['text'])
          y_train_vect_norm = train3_clean['Recommended']
          X_test_vect_norm = pipe.transform(test3_clean['text'])
          y_test_vect_norm = test3_clean['Recommended']
          feature_names_vect_norm = vect.get_feature_names()
          logreg_vect_norm = LogisticRegression()
          logreg_vect_norm.fit(X_train_vect_norm,y_train_vect_norm)
          print('test score:',logreg_vect_norm.score(X_test_vect_norm,y_test_vect_norm))
```

```
test score: 0.8854109961894393
```

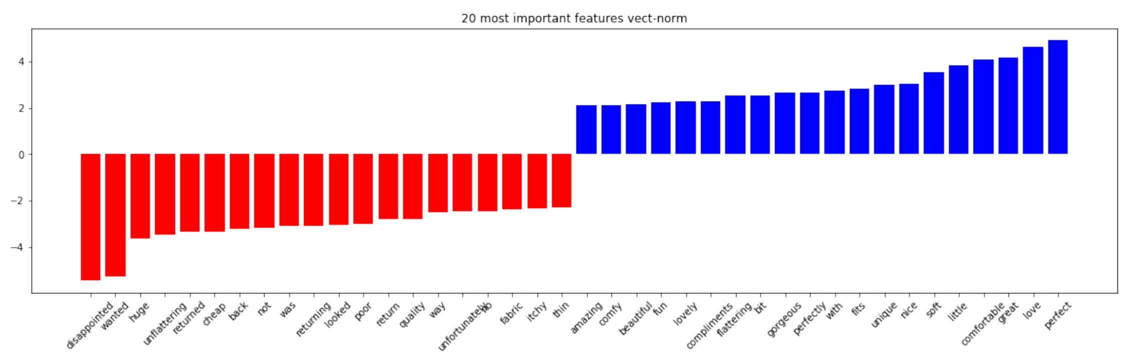
```
In [300]: print("Test Avg Precision score: ", average_precision_score(logreg_vect_norm.predict(X
          print("Test F1 score: ", f1_score(logreg_vect_norm.predict(X_test_vect_norm),y_test_ve
          print("Test ROC AUC score: ", roc_auc_score(logreg_vect_norm.predict(X_test_vect_norm)
```

```
Test Avg Precision score:  0.9588167355654054
Test F1 score:  0.9324562810845499
Test ROC AUC score:  0.8406109670041559
```

```
In [301]: plot(logreg_vect_norm.coef_[0],feature_names_vect_norm,'vect-norm')
```

```
[2.11546514 2.11812685 2.15143136 2.22731376 2.25634018 2.29715048
 2.50858488 2.54229484 2.63528104 2.66174843 2.73191794 2.81655202
```

```
 2.98819196 3.02867748 3.55402791 3.83304181 4.09773229 4.17622217
 4.61750103 4.90917801]
['amazing', 'comfy', 'beautiful', 'fun', 'lovely', 'compliments', 'flattering', 'bit', 'gorgeous
[-5.45024073 -5.25181622 -3.64262797 -3.48839304 -3.35485859 -3.35375814
 -3.2054242  -3.18398355 -3.10949235 -3.07154949 -3.05855551 -3.02667404
 -2.80793175 -2.78651428 -2.5125226  -2.4600435  -2.45273509 -2.36851057
 -2.33169082 -2.29468023]
['disappointed', 'wanted', 'huge', 'unflattering', 'returned', 'cheap', 'back', 'not', 'was', 'r
[-5.45024073 -5.25181622 -3.64262797 -3.48839304 -3.35485859 -3.35375814
 -3.2054242  -3.18398355 -3.10949235 -3.07154949 -3.05855551 -3.02667404
 -2.80793175 -2.78651428 -2.5125226  -2.4600435  -2.45273509 -2.36851057
 -2.33169082 -2.29468023  2.11546514  2.11812685  2.15143136  2.22731376
  2.25634018  2.29715048  2.50858488  2.54229484  2.63528104  2.66174843
  2.73191794  2.81655202  2.98819196  3.02867748  3.55402791  3.83304181
  4.09773229  4.17622217  4.61750103  4.90917801]
['disappointed' 'wanted' 'huge' 'unflattering' 'returned' 'cheap' 'back'
 'not' 'was' 'returning' 'looked' 'poor' 'return' 'quality' 'way'
 'unfortunately' 'no' 'fabric' 'itchy' 'thin' 'amazing' 'comfy'
 'beautiful' 'fun' 'lovely' 'compliments' 'flattering' 'bit' 'gorgeous'
 'perfectly' 'with' 'fits' 'unique' 'nice' 'soft' 'little' 'comfortable'
 'great' 'love' 'perfect']
```



In [302]:
```
#with CV
pipe = Pipeline([("vect", CountVectorizer()),("norm", Normalizer())])
X_train_vect_norm_cv = pipe.fit_transform(train3_clean['text'])
y_train_vect_norm_cv = train3_clean['Recommended']
X_test_vect_norm_cv = pipe.transform(test3_clean['text'])
y_test_vect_norm_cv = test3_clean['Recommended']
feature_names_vect_norm_cv = vect.get_feature_names()
logreg_vect_norm_cv = LogisticRegressionCV(scoring = 'average_precision')
logreg_vect_norm_cv.fit(X_train_vect_norm_cv,y_train_vect_norm_cv)
print('test score:',logreg_vect_norm_cv.score(X_test_vect_norm_cv,y_test_vect_norm_cv)
print('C:',logreg_vect_norm_cv.C_)
```

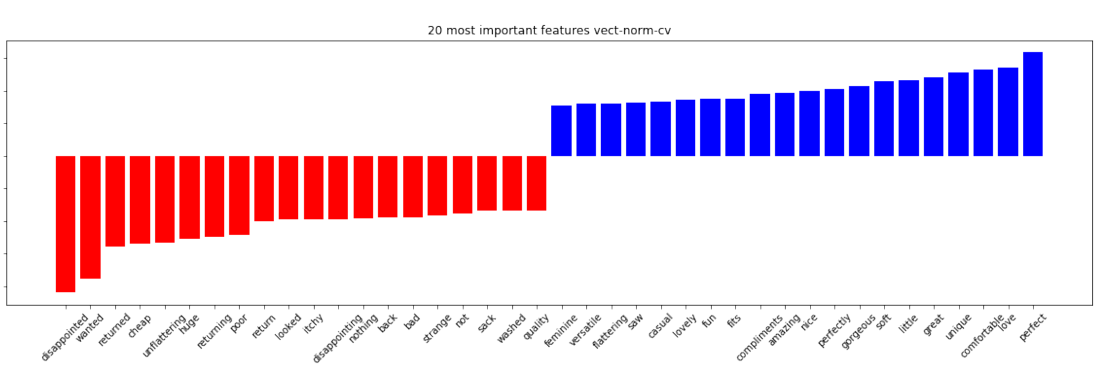
```
test score: 0.8962983124659771
C: [2.7825594]
```

```
In [303]: print("Test Avg Precision score: ", average_precision_score(logreg_vect_norm_cv.predic
          print("Test F1 score: ", f1_score(logreg_vect_norm_cv.predict(X_test_vect_norm_cv),y_t
          print("Test ROC AUC score: ", roc_auc_score(logreg_vect_norm_cv.predict(X_test_vect_no
```

```
Test Avg Precision score:  0.9533216550076367
Test F1 score:  0.9381393083292743
Test ROC AUC score:  0.8476610097332531
```

```
In [304]: plot(logreg_vect_norm_cv.coef_[0],feature_names_vect_norm_cv,'vect-norm-cv')
```

```
[3.10909631 3.19564714 3.20049755 3.25115085 3.35014979 3.43662333
 3.50021701 3.50936122 3.83427005 3.8826816  4.01011244 4.12739305
 4.2869086  4.58140123 4.66494523 4.83756943 5.10276402 5.31663753
 5.4209769  6.35477368]
['feminine', 'versatile', 'flattering', 'saw', 'casual', 'lovely', 'fun', 'fits', 'compliments',
[-8.38014997 -7.50818291 -5.54894674 -5.39225574 -5.32377821 -5.07377691
 -4.93899527 -4.8663299  -4.03298519 -3.87550547 -3.87299922 -3.87262506
 -3.81480378 -3.74055922 -3.73986894 -3.66948387 -3.49747193 -3.36723469
 -3.32297536 -3.31866858]
['disappointed', 'wanted', 'returned', 'cheap', 'unflattering', 'huge', 'returning', 'poor', 're
[-8.38014997 -7.50818291 -5.54894674 -5.39225574 -5.32377821 -5.07377691
 -4.93899527 -4.8663299  -4.03298519 -3.87550547 -3.87299922 -3.87262506
 -3.81480378 -3.74055922 -3.73986894 -3.66948387 -3.49747193 -3.36723469
 -3.32297536 -3.31866858  3.10909631  3.19564714  3.20049755  3.25115085
  3.35014979  3.43662333  3.50021701  3.50936122  3.83427005  3.8826816
  4.01011244  4.12739305  4.2869086   4.58140123  4.66494523  4.83756943
  5.10276402  5.31663753  5.4209769   6.35477368]
['disappointed' 'wanted' 'returned' 'cheap' 'unflattering' 'huge'
 'returning' 'poor' 'return' 'looked' 'itchy' 'disappointing' 'nothing'
 'back' 'bad' 'strange' 'not' 'sack' 'washed' 'quality' 'feminine'
 'versatile' 'flattering' 'saw' 'casual' 'lovely' 'fun' 'fits'
 'compliments' 'amazing' 'nice' 'perfectly' 'gorgeous' 'soft' 'little'
 'great' 'unique' 'comfortable' 'love' 'perfect']
```



```
In [305]: print ('With CountVectorizer:')
          print('Test Score with Linear Classifier: ',logreg3.score(X_test3,y_test3))
          print('Test Score with Linear Classifier and CV: ',logreg3_cv.score(X_test3_cv,y_test3
          print('With TfidfVectorizer:')
          print('Test Score with Linear Classifier: ',logreg_tfidf.score(X_test_tfidf,y_test_tfi
          print('Test Score with Linear Classifier and CV: ',logreg_tfidf_cv.score(X_test_tfidf_
          print('With CountVectorizer and Normalizer:')
          print('Test Score with Linear Classifier: ',logreg_vect_norm.score(X_test_vect_norm,y_
          print('Test Score with Linear Classifier and CV: ',logreg_vect_norm_cv.score(X_test_ve
```

```
With CountVectorizer:
Test Score with Linear Classifier:  0.8987479586281981
Test Score with Linear Classifier and CV:  0.9009254218835057
With TfidfVectorizer:
Test Score with Linear Classifier:  0.8938486663037561
Test Score with Linear Classifier and CV:  0.9066412629286881
With CountVectorizer and Normalizer:
Test Score with Linear Classifier:  0.8854109961894393
Test Score with Linear Classifier and CV:  0.8962983124659771
```

Using normalizer with countvectorizer has marginally improved performance

2.3 Try using stop-word. Do the standard English stop-words help? Why / why not?

```
In [306]: #default params
          vect = CountVectorizer(stop_words='english')
          X_train_vect_sw = vect.fit_transform(train3_clean['text'])
          y_train_vect_sw = train3_clean['Recommended']
          X_test_vect_sw = vect.transform(test3_clean['text'])
          y_test_vect_sw = test3_clean['Recommended']
          feature_names_vect_sw = vect.get_feature_names()
          logreg_vect_sw = LogisticRegression()
          logreg_vect_sw.fit(X_train_vect_sw,y_train_vect_sw)
          print('test score:',logreg_vect_sw.score(X_test_vect_sw,y_test_vect_sw))
```

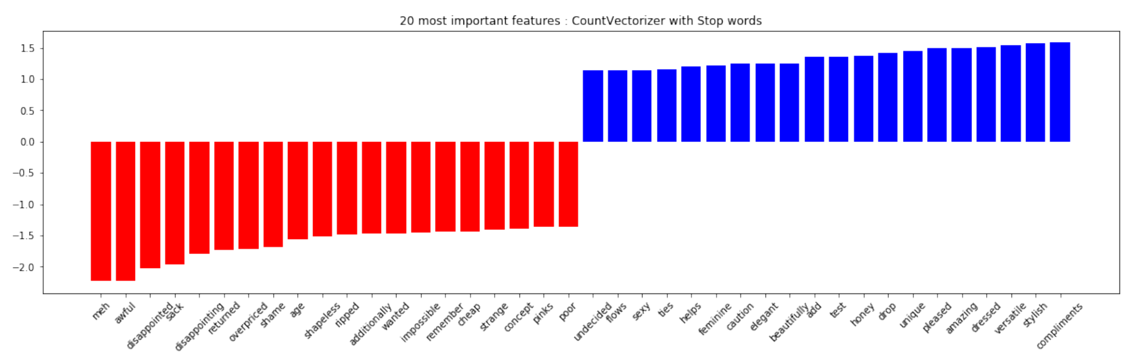
```
test score: 0.896842678279804
```

```
In [307]: print("Test Avg Precision score: ", average_precision_score(logreg_vect_sw.predict(X_t
          print("Test F1 score: ", f1_score(logreg_vect_sw.predict(X_test_vect_sw),y_test_vect_s
          print("Test ROC AUC score: ", roc_auc_score(logreg_vect_sw.predict(X_test_vect_sw),y_t
```

```
Test Avg Precision score:  0.9427435212548922
Test F1 score:  0.9377770481037596
Test ROC AUC score:  0.8370086120002121
```

In [308]: plot(logreg_vect_sw.coef_[0],feature_names_vect_sw,': CountVectorizer with Stop words'

```
[1.13122817 1.13658114 1.13938982 1.14638635 1.201373   1.2105783
 1.24508005 1.24512604 1.25067255 1.34808707 1.35607133 1.37295456
 1.41088651 1.43883432 1.4904509  1.49450224 1.50496597 1.53579253
 1.56336435 1.58051794]
['undecided', 'flows', 'sexy', 'ties', 'helps', 'feminine', 'caution', 'elegant', 'beautifully',
[-2.23187472 -2.21878404 -2.0312487  -1.97051128 -1.79263101 -1.73418806
 -1.72229867 -1.69288398 -1.55811543 -1.51648496 -1.48449446 -1.47365371
 -1.46474721 -1.46176939 -1.43934002 -1.43897215 -1.40267521 -1.38728859
 -1.36865472 -1.36698248]
['meh', 'awful', 'disappointed', 'sack', 'disappointing', 'returned', 'overpriced', 'shame', 'ag
[-2.23187472 -2.21878404 -2.0312487  -1.97051128 -1.79263101 -1.73418806
 -1.72229867 -1.69288398 -1.55811543 -1.51648496 -1.48449446 -1.47365371
 -1.46474721 -1.46176939 -1.43934002 -1.43897215 -1.40267521 -1.38728859
 -1.36865472 -1.36698248  1.13122817  1.13658114  1.13938982  1.14638635
  1.201373    1.2105783   1.24508005  1.24512604  1.25067255  1.34808707
  1.35607133  1.37295456  1.41088651  1.43883432  1.4904509   1.49450224
  1.50496597  1.53579253  1.56336435  1.58051794]
['meh' 'awful' 'disappointed' 'sack' 'disappointing' 'returned'
 'overpriced' 'shame' 'age' 'shapeless' 'ripped' 'additionally' 'wanted'
 'impossible' 'remember' 'cheap' 'strange' 'concept' 'pinks' 'poor'
 'undecided' 'flows' 'sexy' 'ties' 'helps' 'feminine' 'caution' 'elegant'
 'beautifully' 'add' 'test' 'honey' 'drop' 'unique' 'pleased' 'amazing'
 'dressed' 'versatile' 'stylish' 'compliments']
```



In [309]:
```
#with CV and stopwords
vect = CountVectorizer(stop_words='english')
X_train_vect_sw_cv = vect.fit_transform(train3_clean['text'])
y_train_vect_sw_cv = train3_clean['Recommended']
X_test_vect_sw_cv = vect.transform(test3_clean['text'])
y_test_vect_sw_cv = test3_clean['Recommended']
feature_names_vect_sw_cv = vect.get_feature_names()
logreg_vect_sw_cv = LogisticRegressionCV(scoring = 'average_precision')
```

```
logreg_vect_sw_cv.fit(X_train_vect_sw_cv,y_train_vect_sw_cv)
print('test score:',logreg_vect_sw_cv.score(X_test_vect_sw_cv,y_test_vect_sw_cv))
print('C:',logreg_vect_sw_cv.C_)
```

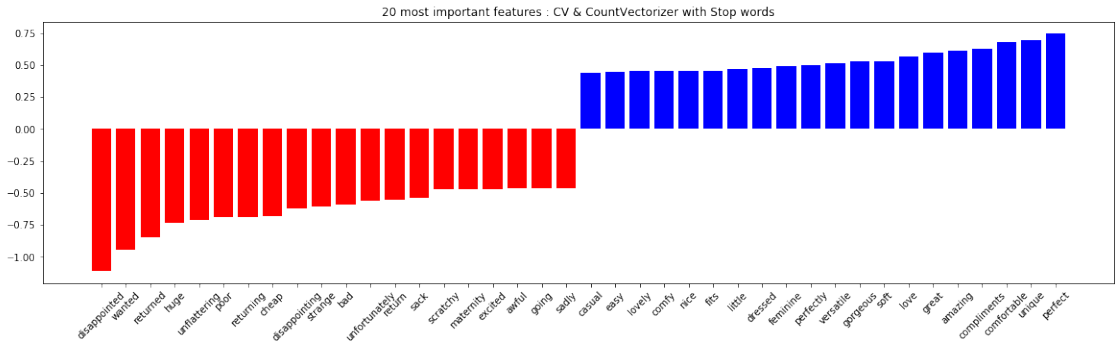
```
test score: 0.8908546543277083
C: [0.04641589]
```

```
In [310]: print("Test Avg Precision score: ", average_precision_score(logreg_vect_sw_cv.predict(
          print("Test F1 score: ", f1_score(logreg_vect_sw_cv.predict(X_test_vect_sw_cv),y_test_
          print("Test ROC AUC score: ", roc_auc_score(logreg_vect_sw_cv.predict(X_test_vect_sw_c
```

```
Test Avg Precision score:  0.9597860202527733
Test F1 score:  0.9355201800932627
Test ROC AUC score:  0.8496498131612679
```

```
In [311]: plot(logreg_vect_sw_cv.coef_[0],feature_names_vect_sw_cv,': CV & CountVectorizer with
```

```
[0.44232217 0.44372799 0.4503959  0.4573169  0.45736872 0.45745416
 0.46552602 0.47324456 0.4921474  0.49978993 0.5136648  0.52687561
 0.5308046  0.56465259 0.59639531 0.61255577 0.62354827 0.68252145
 0.69587213 0.74585348]
['casual', 'easy', 'lovely', 'comfy', 'nice', 'fits', 'little', 'dressed', 'feminine', 'perfectl
[-1.1130487  -0.94737919 -0.84963344 -0.73283457 -0.71214146 -0.69075782
 -0.69019049 -0.68558836 -0.62414574 -0.60340188 -0.59150961 -0.56125621
 -0.55226193 -0.53654228 -0.4743584  -0.47013634 -0.47005558 -0.46708256
 -0.46581472 -0.46356465]
['disappointed', 'wanted', 'returned', 'huge', 'unflattering', 'poor', 'returning', 'cheap', 'di
[-1.1130487  -0.94737919 -0.84963344 -0.73283457 -0.71214146 -0.69075782
 -0.69019049 -0.68558836 -0.62414574 -0.60340188 -0.59150961 -0.56125621
 -0.55226193 -0.53654228 -0.4743584  -0.47013634 -0.47005558 -0.46708256
 -0.46581472 -0.46356465  0.44232217  0.44372799  0.4503959   0.4573169
  0.45736872  0.45745416  0.46552602  0.47324456  0.4921474   0.49978993
  0.5136648   0.52687561  0.5308046   0.56465259  0.59639531  0.61255577
  0.62354827  0.68252145  0.69587213  0.74585348]
['disappointed' 'wanted' 'returned' 'huge' 'unflattering' 'poor'
 'returning' 'cheap' 'disappointing' 'strange' 'bad' 'unfortunately'
 'return' 'sack' 'scratchy' 'maternity' 'excited' 'awful' 'going' 'sadly'
 'casual' 'easy' 'lovely' 'comfy' 'nice' 'fits' 'little' 'dressed'
 'feminine' 'perfectly' 'versatile' 'gorgeous' 'soft' 'love' 'great'
 'amazing' 'compliments' 'comfortable' 'unique' 'perfect']
```




```
In [312]: #default params and tfidf with stop words

          tfidf = TfidfVectorizer(stop_words='english')
          X_train_tfidf_sw = tfidf.fit_transform(train3_clean['text'])
          y_train_tfidf_sw = train3_clean['Recommended']
          X_test_tfidf_sw = tfidf.transform(test3_clean['text'])
          y_test_tfidf_sw = test3_clean['Recommended']
          feature_names_tfidf_sw = tfidf.get_feature_names()
          logreg_tfidf_sw = LogisticRegression()
          logreg_tfidf_sw.fit(X_train_tfidf_sw,y_train_tfidf_sw)
          print('test score:',logreg_tfidf_sw.score(X_test_tfidf_sw,y_test_tfidf_sw))
```

```
test score: 0.8938486663037561
```

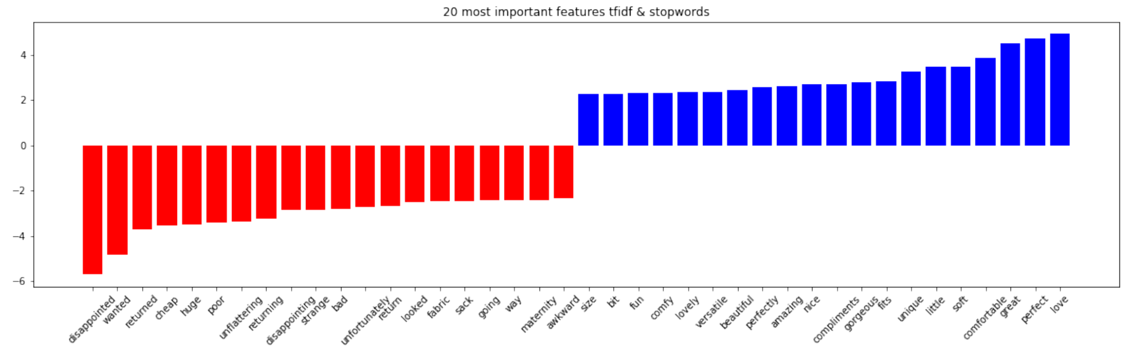
```
In [313]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf_sw.predict(X_
          print("Test F1 score: ", f1_score(logreg_tfidf_sw.predict(X_test_tfidf_sw),y_test_tfid
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf_sw.predict(X_test_tfidf_sw),y
```

```
Test Avg Precision score:  0.9683735664405579
Test F1 score:  0.937699680511182
Test ROC AUC score:  0.86903323518649
```

```
In [314]: plot(logreg_tfidf_sw.coef_[0],feature_names_tfidf_sw,'tfidf & stopwords')
```

```
[2.24309334 2.27317205 2.29249083 2.29565165 2.35160429 2.36360053
 2.42158491 2.55875488 2.61334287 2.67602534 2.71030247 2.75837011
 2.83883188 3.23322345 3.48019051 3.48050415 3.8497441  4.49913654
 4.71209902 4.9195551 ]
['size', 'bit', 'fun', 'comfy', 'lovely', 'versatile', 'beautiful', 'perfectly', 'amazing', 'nic
[-5.70946075 -4.85057813 -3.73639444 -3.53734756 -3.48839878 -3.43451937
 -3.38481685 -3.23307691 -2.87287812 -2.84674384 -2.82581412 -2.72887465
 -2.7047743  -2.52016324 -2.4716081  -2.45137907 -2.44178174 -2.43649776
 -2.42454743 -2.35527015]
```

```
['disappointed', 'wanted', 'returned', 'cheap', 'huge', 'poor', 'unflattering', 'returning', 'di
[-5.70946075 -4.85057813 -3.73639444 -3.53734756 -3.48839878 -3.43451937
 -3.38481685 -3.23307691 -2.87287812 -2.84674384 -2.82581412 -2.72887465
 -2.7047743  -2.52016324 -2.4716081  -2.45137907 -2.44178174 -2.43649776
 -2.42454743 -2.35527015  2.24309334  2.27317205  2.29249083  2.29565165
  2.35160429  2.36360053  2.42158491  2.55875488  2.61334287  2.67602534
  2.71030247  2.75837011  2.83883188  3.23322345  3.48019051  3.48050415
  3.8497441   4.49913654  4.71209902  4.9195551 ]
['disappointed' 'wanted' 'returned' 'cheap' 'huge' 'poor' 'unflattering'
 'returning' 'disappointing' 'strange' 'bad' 'unfortunately' 'return'
 'looked' 'fabric' 'sack' 'going' 'way' 'maternity' 'awkward' 'size' 'bit'
 'fun' 'comfy' 'lovely' 'versatile' 'beautiful' 'perfectly' 'amazing'
 'nice' 'compliments' 'gorgeous' 'fits' 'unique' 'little' 'soft'
 'comfortable' 'great' 'perfect' 'love']
```



In [315]:
```python
#with CV tfidf with stop words

tfidf = TfidfVectorizer(stop_words='english')
X_train_tfidf_sw_cv = tfidf.fit_transform(train3_clean['text'])
y_train_tfidf_sw_cv = train3_clean['Recommended']
X_test_tfidf_sw_cv = tfidf.transform(test3_clean['text'])
y_test_tfidf_sw_cv = test3_clean['Recommended']
feature_names_tfidf_sw_cv = tfidf.get_feature_names()
logreg_tfidf_sw_cv = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf_sw_cv.fit(X_train_tfidf_sw_cv,y_train_tfidf_sw_cv)
print('test score:',logreg_tfidf_sw_cv.score(X_test_tfidf_sw_cv,y_test_tfidf_sw_cv))
print('C:',logreg_tfidf_sw_cv.C_)
```

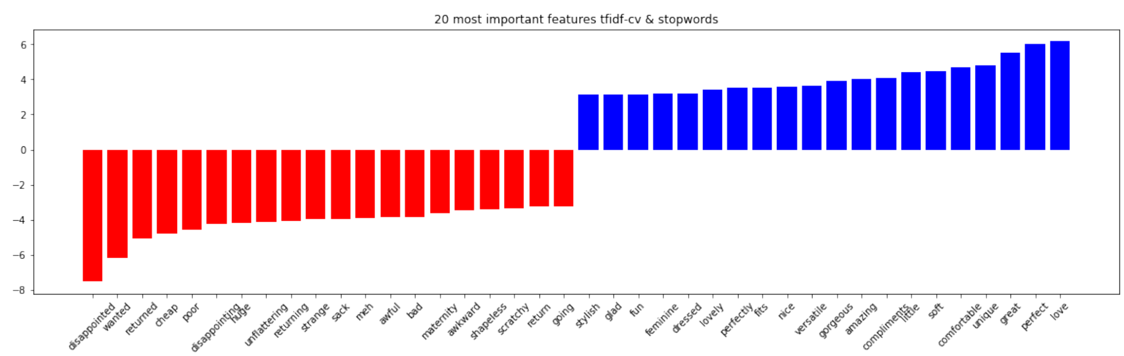
```
test score: 0.902286336418073
C: [2.7825594]
```

In [316]:
```python
print("Test Avg Precision score: ", average_precision_score(logreg_tfidf_sw_cv.predict
print("Test F1 score: ", f1_score(logreg_tfidf_sw_cv.predict(X_test_tfidf_sw_cv),y_tes
print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf_sw_cv.predict(X_test_tfidf_sw
```

```
Test Avg Precision score:  0.959625936859364
Test F1 score:  0.9418623481781376
Test ROC AUC score:  0.8637643624744401
```

In [317]:
```python
plot(logreg_tfidf_sw_cv.coef_[0],feature_names_tfidf_sw_cv,'tfidf-cv & stopwords')
```

```
[3.12208732 3.13204727 3.13389526 3.17731055 3.19010598 3.39522046
 3.50938111 3.52701653 3.57693134 3.63081836 3.91266679 4.04640978
 4.08363353 4.38968143 4.47229675 4.68826727 4.77137418 5.5457754
 6.04501549 6.16864579]
['stylish', 'glad', 'fun', 'feminine', 'dressed', 'lovely', 'perfectly', 'fits', 'nice', 'versat
[-7.53388133 -6.17636272 -5.08383178 -4.78998201 -4.58673271 -4.23868543
 -4.19697817 -4.12784765 -4.06171244 -3.98678958 -3.95073265 -3.92344061
 -3.84406456 -3.83423518 -3.63657697 -3.45340063 -3.43096836 -3.36442451
 -3.22027304 -3.21963511]
['disappointed', 'wanted', 'returned', 'cheap', 'poor', 'disappointing', 'huge', 'unflattering',
[-7.53388133 -6.17636272 -5.08383178 -4.78998201 -4.58673271 -4.23868543
 -4.19697817 -4.12784765 -4.06171244 -3.98678958 -3.95073265 -3.92344061
 -3.84406456 -3.83423518 -3.63657697 -3.45340063 -3.43096836 -3.36442451
 -3.22027304 -3.21963511  3.12208732  3.13204727  3.13389526  3.17731055
  3.19010598  3.39522046  3.50938111  3.52701653  3.57693134  3.63081836
  3.91266679  4.04640978  4.08363353  4.38968143  4.47229675  4.68826727
  4.77137418  5.5457754   6.04501549  6.16864579]
['disappointed' 'wanted' 'returned' 'cheap' 'poor' 'disappointing' 'huge'
 'unflattering' 'returning' 'strange' 'sack' 'meh' 'awful' 'bad'
 'maternity' 'awkward' 'shapeless' 'scratchy' 'return' 'going' 'stylish'
 'glad' 'fun' 'feminine' 'dressed' 'lovely' 'perfectly' 'fits' 'nice'
 'versatile' 'gorgeous' 'amazing' 'compliments' 'little' 'soft'
 'comfortable' 'unique' 'great' 'perfect' 'love']
```



In [318]:
```python
print ('With CountVectorizer:')
print('Test Score with Linear Classifier: ',logreg3.score(X_test3,y_test3))
print('Test Score with Linear Classifier and CV: ',logreg3_cv.score(X_test3_cv,y_test3
```

```
print('With TfidfVectorizer:')
print('Test Score with Linear Classifier: ',logreg_tfidf.score(X_test_tfidf,y_test_tfi
print('Test Score with Linear Classifier and CV: ',logreg_tfidf_cv.score(X_test_tfidf_
```

```
With CountVectorizer:
Test Score with Linear Classifier:  0.8987479586281981
Test Score with Linear Classifier and CV:  0.9009254218835057
With TfidfVectorizer:
Test Score with Linear Classifier:  0.8938486663037561
Test Score with Linear Classifier and CV:  0.9066412629286881
```

In [319]:
```
print ('With CountVectorizer and Stop Words:')
print('Test Score with Linear Classifier: ',logreg_vect_sw.score(X_test_vect_sw,y_test
print('Test Score with Linear Classifier and CV: ',logreg_vect_sw_cv.score(X_test_vect
print('With TfidfVectorizer and Stop Words:')
print('Test Score with Linear Classifier: ',logreg_tfidf_sw.score(X_test_tfidf_sw,y_te
print('Test Score with Linear Classifier and CV: ',logreg_tfidf_sw_cv.score(X_test_tfi
```

```
With CountVectorizer and Stop Words:
Test Score with Linear Classifier:  0.896842678279804
Test Score with Linear Classifier and CV:  0.8908546543277083
With TfidfVectorizer and Stop Words:
Test Score with Linear Classifier:  0.8938486663037561
Test Score with Linear Classifier and CV:  0.902286336418073
```

Stop words removal also gives a marginal improvement in avg precision and roc auc scores. But not much in terms of accuracy. Thus removal of stop words improves performance of this imbalanced model. Also, a reduction in features is observed which saves computational cost

In [320]:
```
print('Lenght of feature_names before stop words removal:')
print('before')
bsw = [feature_names3, feature_names3_cv, feature_names_tfidf, feature_names_tfidf_cv]
for i in bsw:
    print(len(i))
asw = [feature_names_vect_sw, feature_names_vect_sw_cv, feature_names_tfidf_sw, featur
print('after')
for i in asw:
    print(len(i))
```

```
Lenght of feature_names before stop words removal:
before
11095
11095
11095
11095
after
10814
```

```
10814
10814
10814
```

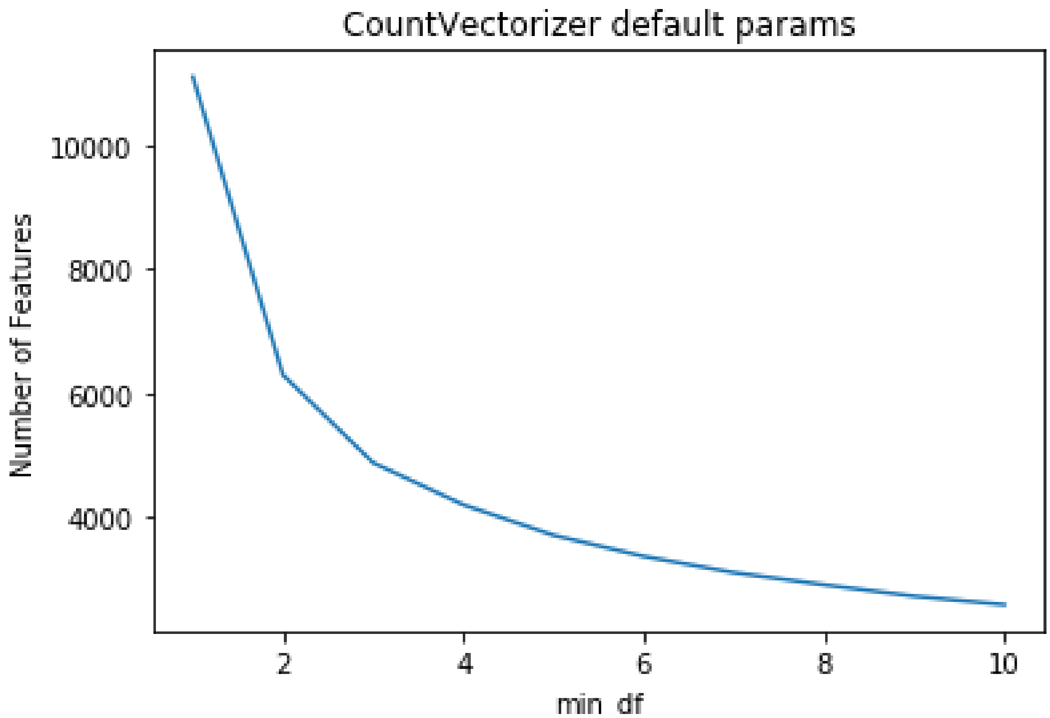
The features were reduced by merely 280 which is less than 2.25%. So, no much impact on stop words removal

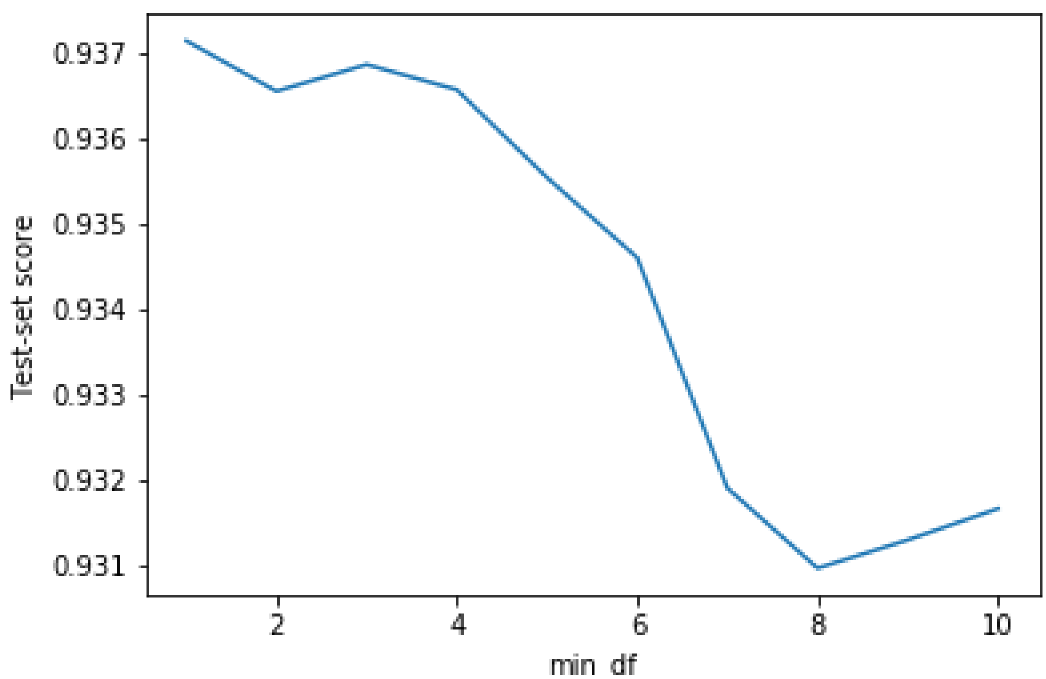
2.4 Limit the vocabulary using min_df or max_df. How to these impact the number of features, and how do they impact the scores?

```
In [321]: def min_df(feature_count,test_scores,title):
              plt.plot(range(1,11),feature_count)
              plt.title(str(title))
              plt.xlabel('min_df')
              plt.ylabel('Number of Features')
              plt.show()
              plt.plot(range(1,11),test_scores)
              plt.xlabel('min_df')
              plt.ylabel('Test-set score')
              plt.show()
```

```
In [322]: feature_count_vect = []
          test_scores_vect = []
          for i in range(1,11):
          #default params
              vect = CountVectorizer(min_df = i)
              X_train = vect.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = vect.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = vect.get_feature_names()
              feature_count_vect.append(len(feature_names))
              logreg = LogisticRegression()
              logreg.fit(X_train,y_train)
              test_scores_vect.append(average_precision_score(logreg.predict(X_test), y_test))
```

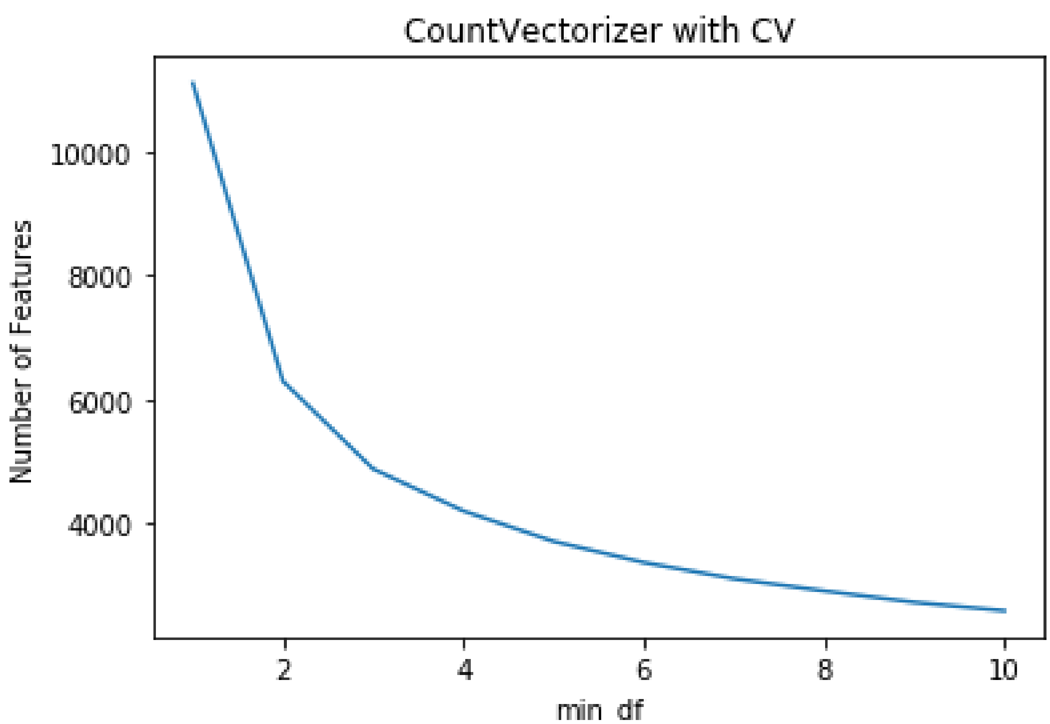
```
In [323]: min_df(feature_count_vect,test_scores_vect,'CountVectorizer default params')
```

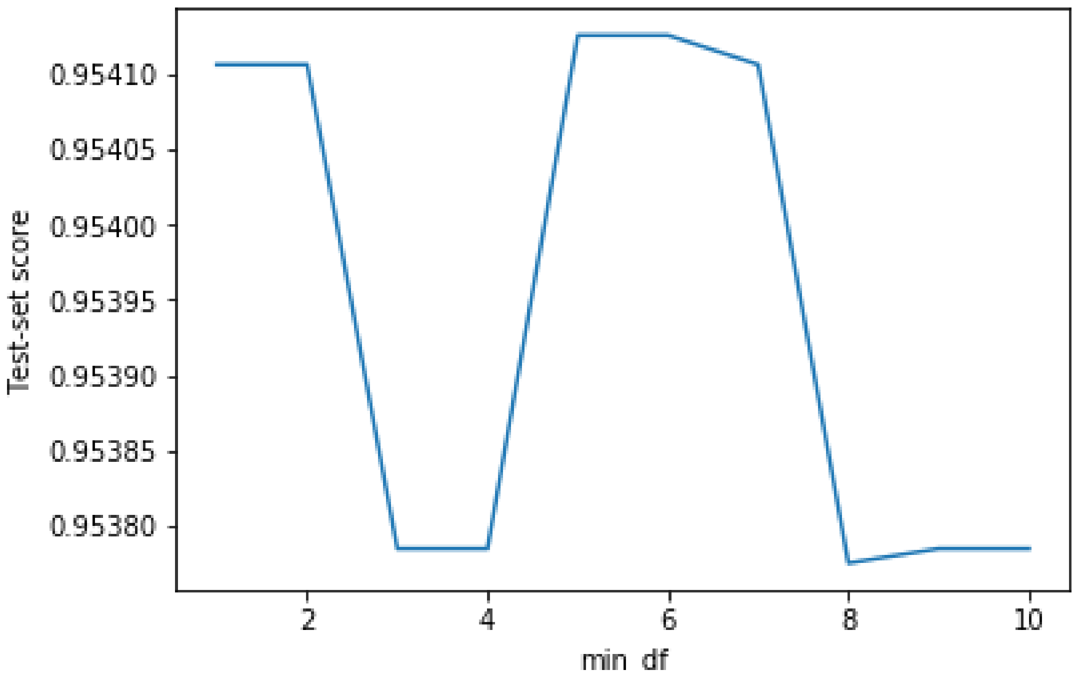

CountVectorizer default params
Number of Features
10000
8000
6000
4000
2
4
6
8
10
min_df


Test-set score
0.937
0.936
0.935
0.934
0.933
0.932
0.931
2
4
6
8
10
min_df

```
In [324]: feature_count_vect_cv = []
          test_scores_vect_cv = []
          for i in range(1,11):
          #default params and CV
              vect = CountVectorizer(min_df = i)
              X_train = vect.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = vect.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = vect.get_feature_names()
              feature_count_vect_cv.append(len(feature_names))
              logreg = LogisticRegressionCV(scoring = 'average_precision')
              logreg.fit(X_train,y_train)
              test_scores_vect_cv.append(average_precision_score(logreg.predict(X_test),y_test))
```

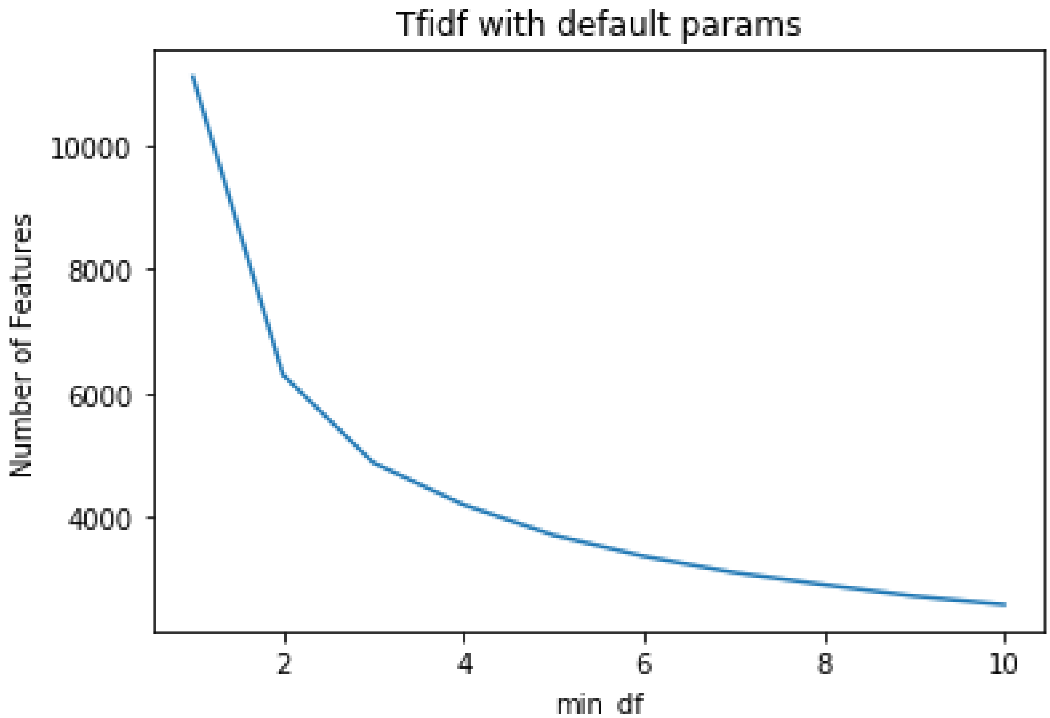
```
In [325]: min_df(feature_count_vect_cv,test_scores_vect_cv, 'CountVectorizer with CV')
```

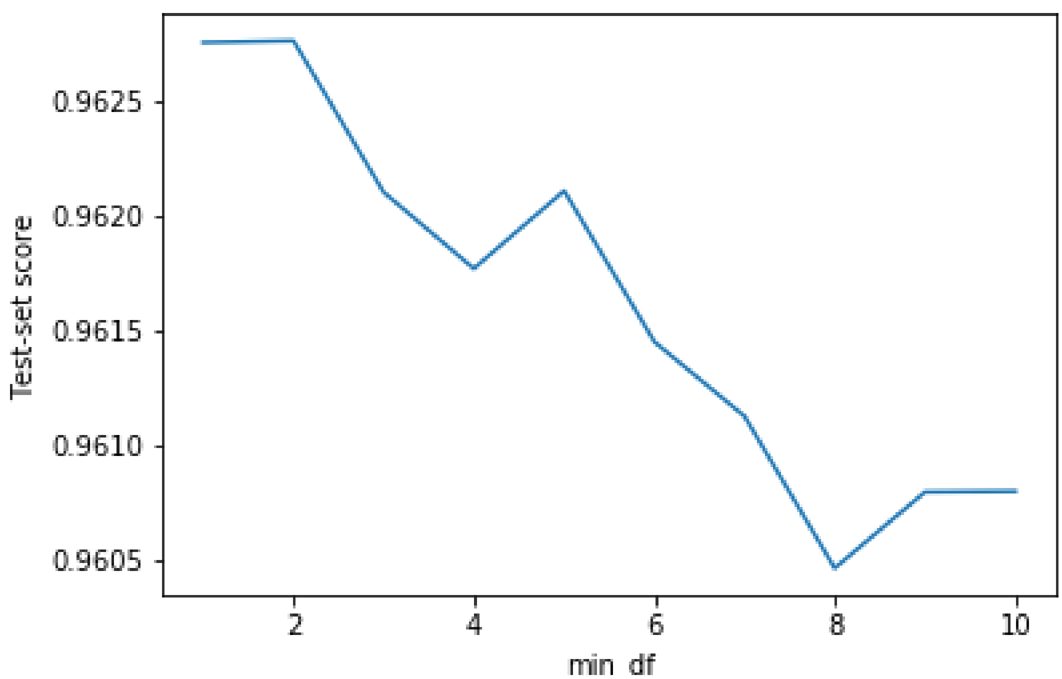




In [326]:
```
feature_count_tfidf = []
test_scores_tfidf = []
#default params and tfidf
for i in range(1,11):
    tfidf = TfidfVectorizer(min_df = i)
    X_train = tfidf.fit_transform(train3_clean['text'])
    y_train = train3_clean['Recommended']
    X_test = tfidf.transform(test3_clean['text'])
    y_test = test3_clean['Recommended']
    feature_names = tfidf.get_feature_names()
    feature_count_tfidf.append(len(feature_names))
    logreg = LogisticRegression()
    logreg.fit(X_train,y_train)
    test_scores_tfidf.append(average_precision_score(logreg.predict(X_test),y_test))
```

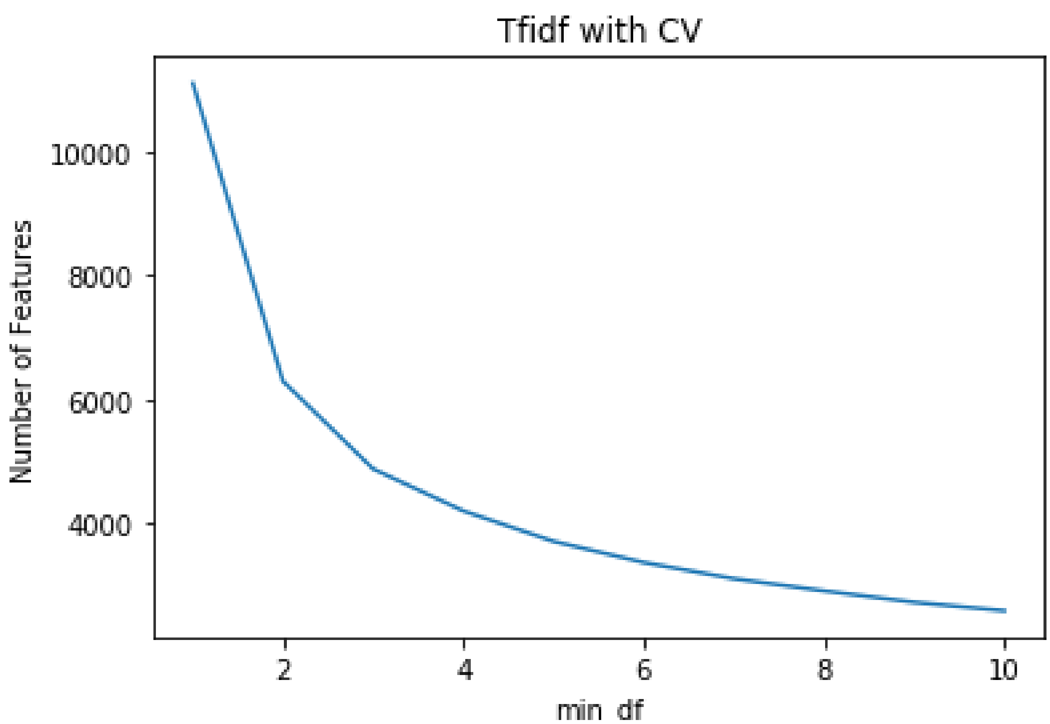
In [327]:
```
min_df(feature_count_tfidf,test_scores_tfidf,'Tfidf with default params')
```

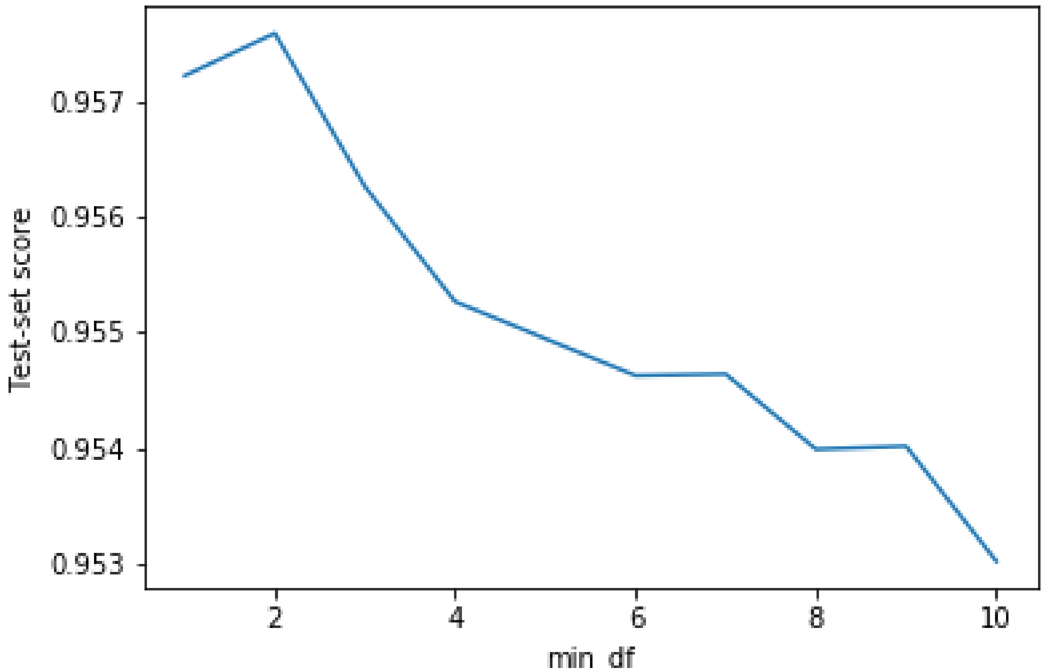

Tfidf with default params
Number of Features
10000
8000
6000
4000
2
4
6
8
10
min_df


Test-set score
0.9625
0.9620
0.9615
0.9610
0.9605
2
4
6
8
10
min_df

```
In [328]: feature_count_tfidf_cv = []
          test_scores_tfidf_cv = []
          #default params and tfidf
          for i in range(1,11):
              tfidf = TfidfVectorizer(min_df = i)
              X_train = tfidf.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = tfidf.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = tfidf.get_feature_names()
              feature_count_tfidf_cv.append(len(feature_names))
              logreg = LogisticRegressionCV(scoring = 'average_precision')
              logreg.fit(X_train,y_train)
              test_scores_tfidf_cv.append(average_precision_score(logreg.predict(X_test),y_test)
```

```
In [329]: min_df(feature_count_tfidf_cv,test_scores_tfidf_cv,'Tfidf with CV')
```





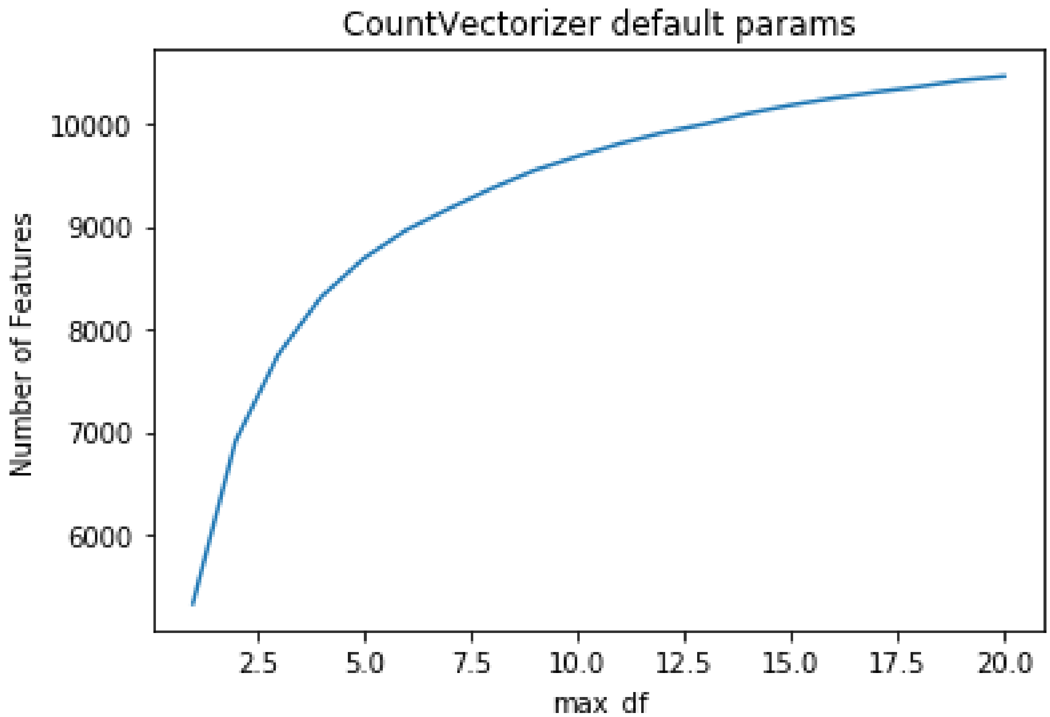
max_df

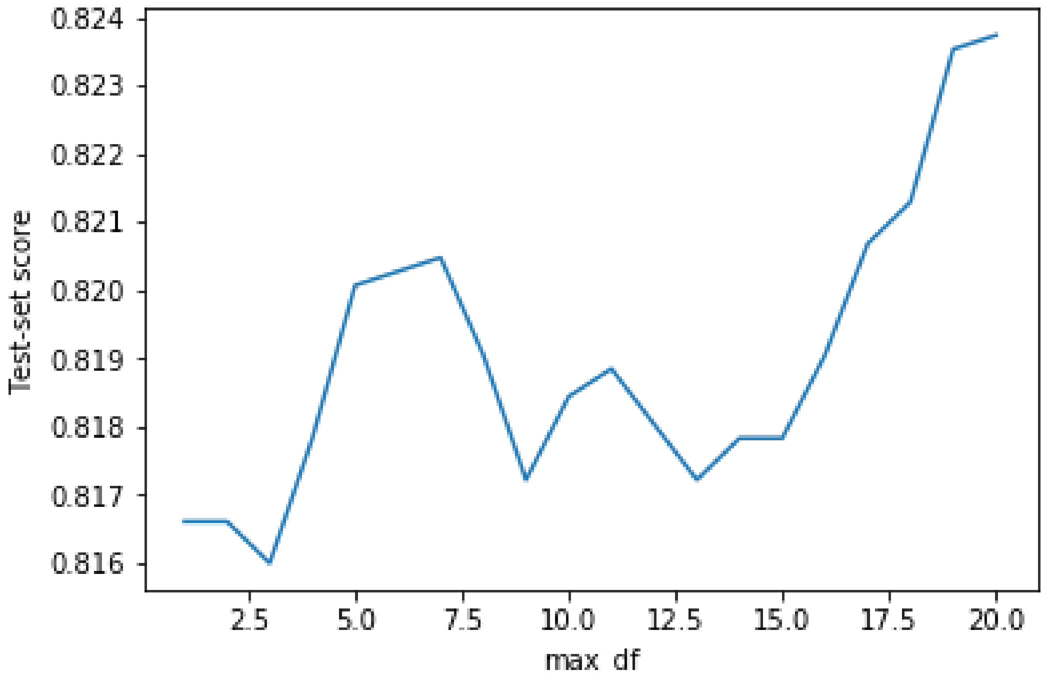
```
In [421]: def max_df(feature_count,test_scores,title):
              plt.plot(range(1,21),feature_count)
              plt.title(str(title))
              plt.xlabel('max_df')
              plt.ylabel('Number of Features')
              plt.show()
              plt.plot(range(1,21),test_scores)
              plt.xlabel('max_df')
              plt.ylabel('Test-set score')
              plt.show()
```

```
In [422]: feature_count_vect = []
          test_scores_vect = []
          for i in range(1,21):
          #default params
              vect = CountVectorizer(max_df = i)
              X_train = vect.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = vect.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = vect.get_feature_names()
              feature_count_vect.append(len(feature_names))
```

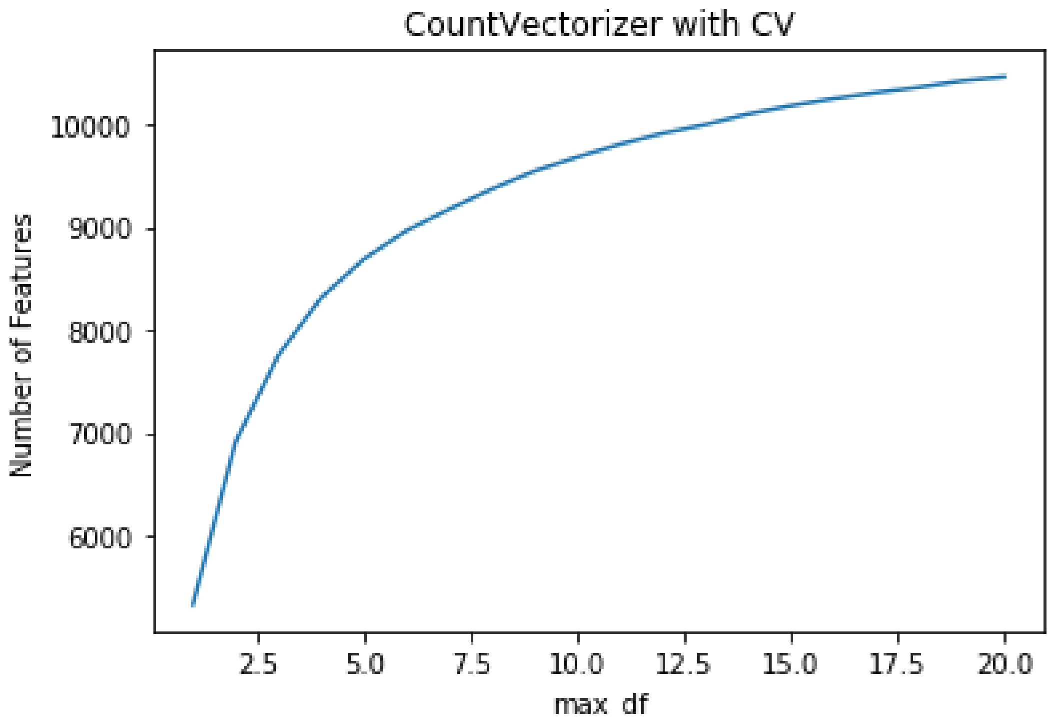
```
        logreg = LogisticRegression()
        logreg.fit(X_train,y_train)
        test_scores_vect.append(average_precision_score(logreg.predict(X_test),y_test))
```

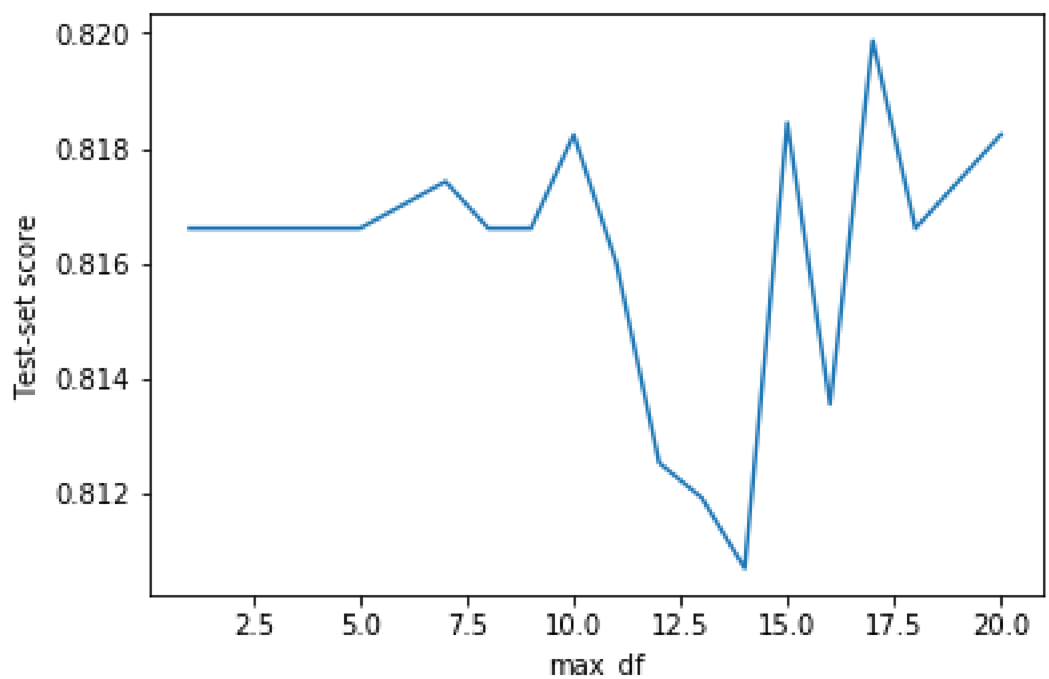
```
In [423]: max_df(feature_count_vect,test_scores_vect,'CountVectorizer default params')
```





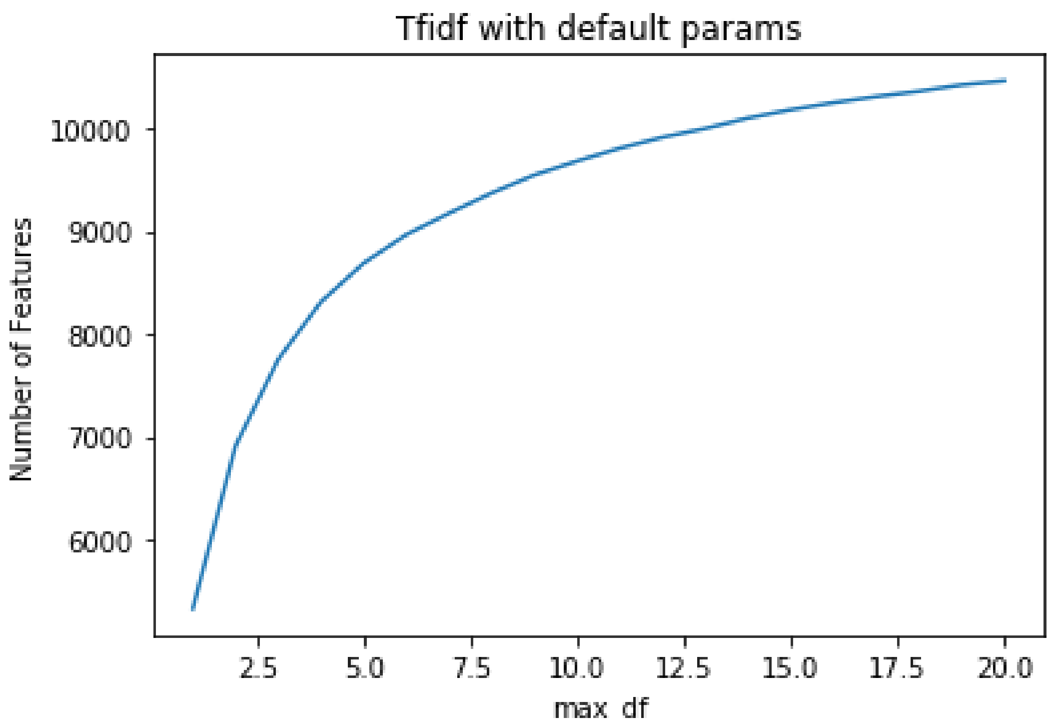

```
In [424]: feature_count_vect_cv = []
          test_scores_vect_cv = []
          for i in range(1,21):
          #default params and CV
              vect = CountVectorizer(max_df = i)
              X_train = vect.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = vect.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = vect.get_feature_names()
              feature_count_vect_cv.append(len(feature_names))
              logreg = LogisticRegressionCV(scoring = 'average_precision')
              logreg.fit(X_train,y_train)
              test_scores_vect_cv.append(average_precision_score(logreg.predict(X_test),y_test))

In [425]: max_df(feature_count_vect_cv,test_scores_vect_cv, 'CountVectorizer with CV')
```

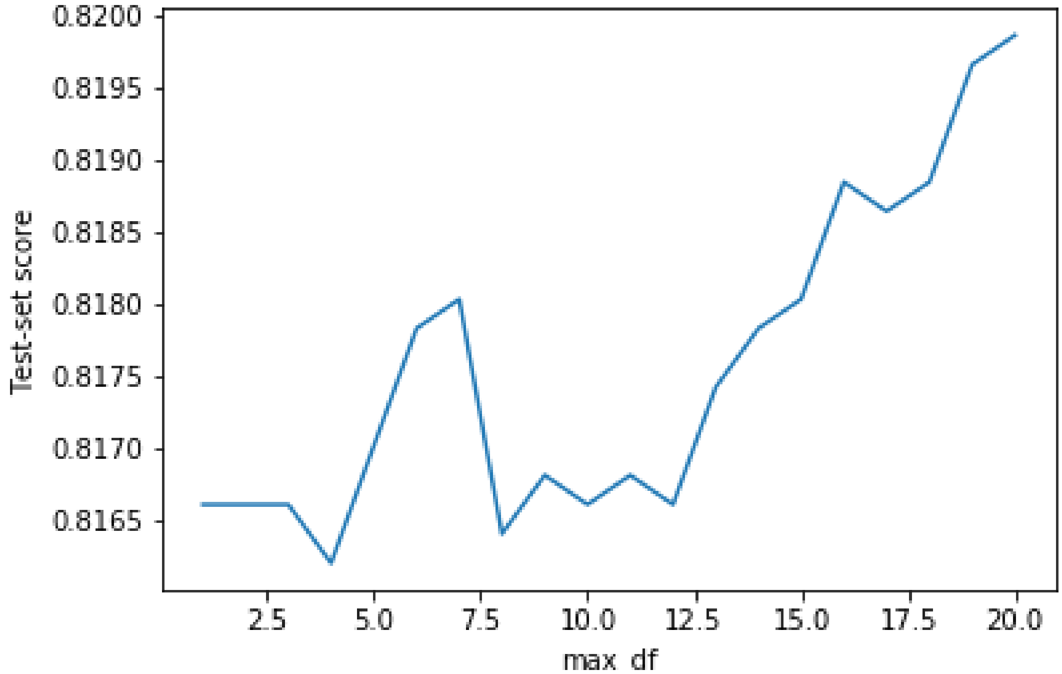

CountVectorizer with CV
Number of Features
10000
9000
8000
7000
6000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_df


Test-set score
0.820
0.818
0.816
0.814
0.812
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_df

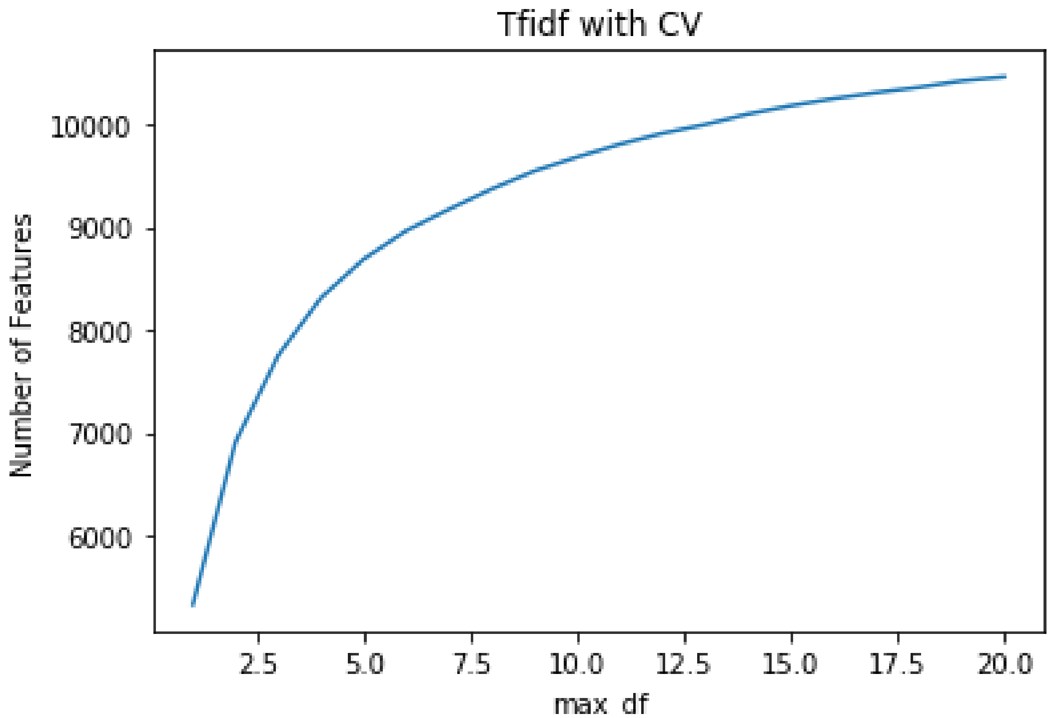
In [426]:
```
feature_count_tfidf = []
test_scores_tfidf = []
#default params and tfidf
for i in range(1,21):
    tfidf = TfidfVectorizer(max_df = i)
    X_train = tfidf.fit_transform(train3_clean['text'])
    y_train = train3_clean['Recommended']
    X_test = tfidf.transform(test3_clean['text'])
    y_test = test3_clean['Recommended']
    feature_names = tfidf.get_feature_names()
    feature_count_tfidf.append(len(feature_names))
    logreg = LogisticRegression()
    logreg.fit(X_train,y_train)
    test_scores_tfidf.append(average_precision_score(logreg.predict(X_test),y_test))
```

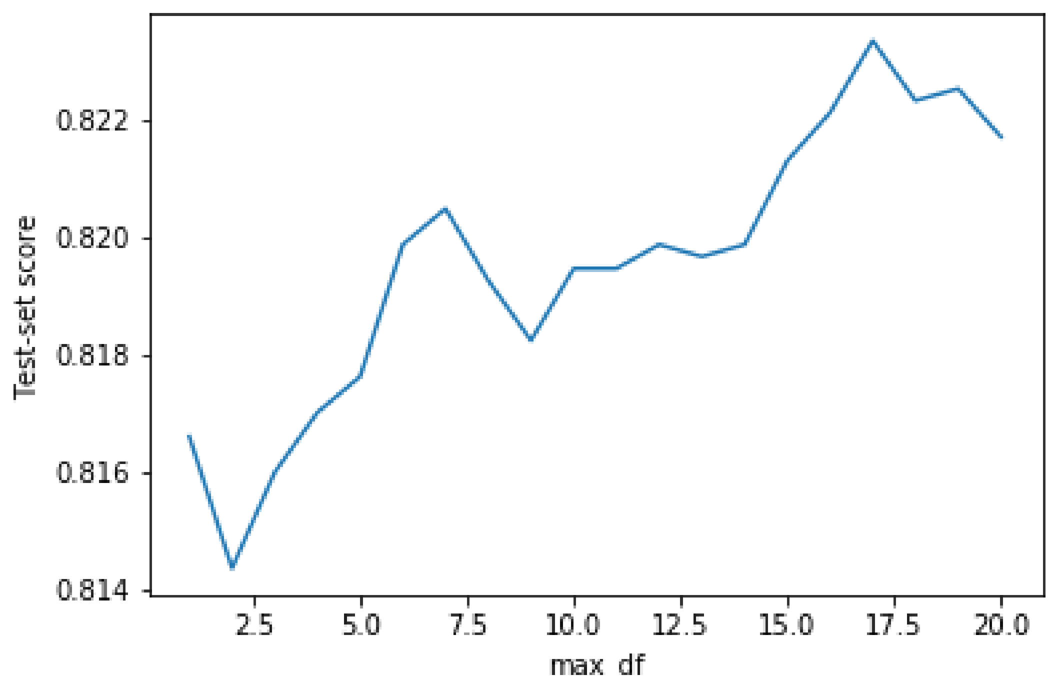
In [427]:
```
max_df(feature_count_tfidf,test_scores_tfidf,'Tfidf with default params')
```






```
In [428]: feature_count_tfidf_cv = []
          test_scores_tfidf_cv = []
          #default params and tfidf
          for i in range(1,21):
              tfidf = TfidfVectorizer(max_df = i)
              X_train = tfidf.fit_transform(train3_clean['text'])
              y_train = train3_clean['Recommended']
              X_test = tfidf.transform(test3_clean['text'])
              y_test = test3_clean['Recommended']
              feature_names = tfidf.get_feature_names()
              feature_count_tfidf_cv.append(len(feature_names))
              logreg = LogisticRegressionCV(scoring = 'average_precision')
              logreg.fit(X_train,y_train)
              test_scores_tfidf_cv.append(average_precision_score(logreg.predict(X_test),y_test)

In [429]: max_df(feature_count_tfidf_cv,test_scores_tfidf_cv,'Tfidf with CV')
```


Tfidf with CV
Number of Features
10000
9000
8000
7000
6000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_df


Test-set score
0.822
0.820
0.818
0.816
0.814
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_df

It is seen clearly that with increase in min_df number of features are reducing and with increase in max_df number of features are increasing

Also, variation in both min_df and max_df has changed the test scores very slightly only. Thus with almost negligible loss in average precision score we get a significant reduction in number of features

In [ ]:

Now we evaluate the best model of Task 2 with the main test set The tfidf with stop words has the best model performance on the basis of average precision score and roc auc

```
In [358]: tfidf = TfidfVectorizer(stop_words='english')
          X_train_tfidf_sw_cv = tfidf.fit_transform(train3_clean['text'])

          X_test_4 = tfidf.transform(test_main_clean["text"])

          #logreg_tfidf_sw_cv

          print("Test Avg Precision score: ", average_precision_score(logreg_tfidf_sw_cv.predict
          print("Test F1 score: ", f1_score(logreg_tfidf_sw_cv.predict(X_test_4),test_main_clean
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf_sw_cv.predict(X_test_4),test_
```

```
Test Avg Precision score:  0.9530968742631256
Test F1 score:  0.9383586626139818
Test ROC AUC score:  0.8487014682445492
```

```
In [333]: #logreg_tfidf_sw_cv

          print("Test Avg Precision score: ", average_precision_score(logreg_tfidf_sw_cv.predict
          print("Test F1 score: ", f1_score(logreg_tfidf_sw_cv.predict(X_test_4),test_main_clean
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf_sw_cv.predict(X_test_4),test_
```

```
Test Avg Precision score:  0.9530968742631256
Test F1 score:  0.9383586626139818
Test ROC AUC score:  0.8487014682445492
```

In [ ]:

0.3 Task 3 n-grams (30Pts)

3.1 Using your current best model, try changing from unigrams to n-grams of varying length. What provides the best performance? Visualize the coefficients. Try visualizing only the higher-order n-grams that are important.

In terms of average precision scores and roc auc the best model from all the previous tasks is tfidf vectorizer with stop words

In [330]:
```
tfidf22 = TfidfVectorizer(stop_words='english', ngram_range=(2,2))

X_train_tfidf22_sw = tfidf22.fit_transform(train3_clean['text'])
y_train_tfidf22_sw = train3_clean['Recommended']
X_test_tfidf22_sw = tfidf22.transform(test3_clean['text'])
y_test_tfidf22_sw = test3_clean['Recommended']
feature_names_tfidf22_sw = tfidf22.get_feature_names()
logreg_tfidf22_sw = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf22_sw.fit(X_train_tfidf22_sw,y_train_tfidf22_sw)
print('test score:',logreg_tfidf22_sw.score(X_test_tfidf22_sw,y_test_tfidf22_sw))
```

```
test score: 0.8489384866630375
```

In [345]:
```
X_train_tfidf22_sw
```

Out[345]:
```
<11088x156338 sparse matrix of type '<class 'numpy.float64'>'
	with 302647 stored elements in Compressed Sparse Row format>
```

In [331]:
```
print("Test Avg Precision score: ", average_precision_score(logreg_tfidf22_sw.predict(
print("Test F1 score: ", f1_score(logreg_tfidf22_sw.predict(X_test_tfidf22_sw),y_test_
print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf22_sw.predict(X_test_tfidf22_s
```

```
Test Avg Precision score:  0.9886119348421321
Test F1 score:  0.9149425287356322
Test ROC AUC score:  0.8645543799874472
```

In [ ]:

In [139]:
```
colour = []
for i in range(40):
    if i < 20:
        colour.append("red")
    else:
        colour.append("blue")

def plot2(coef,feature_names,titl): #plots top 20 features
    top300_index_pos = coef.argsort()[-300:]
    top300_names_pos = [feature_names[j] for j in top300_index_pos]
    top300_coef_pos = coef[top300_index_pos]
#     print (top300_coef_pos)
    names_pos = []
    coef_pos =[]
    total = 1
    for i in range(300):
```

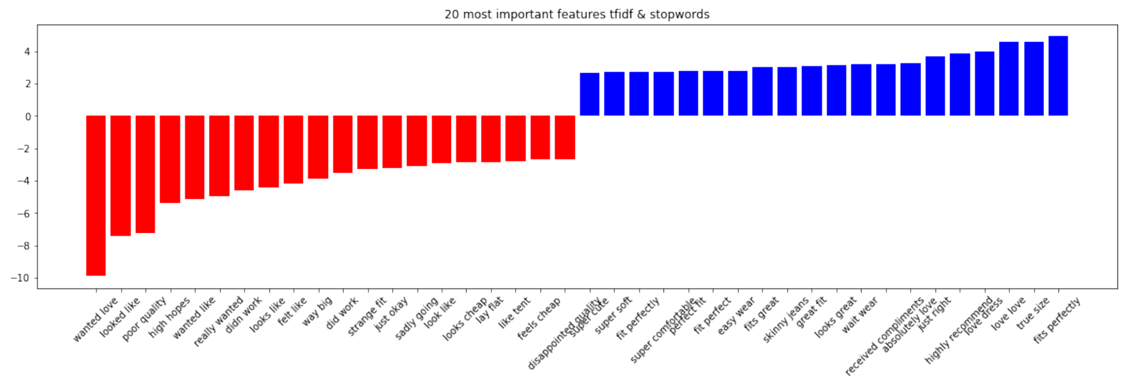
```
                if total <=20:
                    if " " in top300_names_pos[299-i]:
                        names_pos.append(top300_names_pos[299-i])
                        coef_pos.append(top300_coef_pos[299-i])
                        total += 1


            top300_index_neg = coef.argsort()[:300]
            top300_names_neg = [feature_names[j] for j in top300_index_neg]
            top300_coef_neg = coef[top300_index_neg]
            names_neg = []
            coef_neg =[]
            total = 1
            for i in range(300):
                if total <=20:
                    if " " in top300_names_neg[i]:
                        names_neg.append(top300_names_neg[i])
                        coef_neg.append(top300_coef_neg[i])
                        total += 1

            top_coef = np.hstack([coef_neg,coef_pos[::-1]])
            print(top_coef)
            top_names = np.hstack([names_neg,names_pos[::-1]])
            print(top_names)

            plt.figure(figsize=(20, 5))
            plt.bar(range(1,41),top_coef,color=colour)
            plt.title('20 most important features '+str(titl))
            plt.xticks(range(1,41),top_names,rotation=45)
            plt.show()
```

In [ ]:

In [140]:
```
plot2(logreg_tfidf22_sw.coef_[0],feature_names_tfidf22_sw,'tfidf & stopwords')
```

```
[-9.89153026 -7.40883159 -7.25793166 -5.37460345 -5.15416228 -4.97612901
 -4.59765652 -4.42730344 -4.1570644  -3.89734986 -3.51022232 -3.28167424
 -3.19498293 -3.12261321 -2.94847273 -2.88354451 -2.87001837 -2.81674339
 -2.70940987 -2.69536506  2.6638098   2.69098415  2.7169936   2.72186052
  2.77909289  2.79374033  2.80041215  2.99576014  3.01047788  3.08309091
  3.13010298  3.17795158  3.18684767  3.2630977   3.66334684  3.85816849
  4.00612924  4.55173626  4.59992499  4.93492602]
['wanted love' 'looked like' 'poor quality' 'high hopes' 'wanted like'
 'really wanted' 'didn work' 'looks like' 'felt like' 'way big' 'did work'
 'strange fit' 'just okay' 'sadly going' 'look like' 'looks cheap'
 'lay flat' 'like tent' 'feels cheap' 'disappointed quality' 'super cute'
 'super soft' 'fit perfectly' 'super comfortable' 'perfect fit'
 'fit perfect' 'easy wear' 'fits great' 'skinny jeans' 'great fit'
```

```
 'looks great' 'wait wear' 'received compliments' 'absolutely love'
 'just right' 'highly recommend' 'love dress' 'love love' 'true size'
 'fits perfectly']
```



In [ ]:

In [119]:
```
tfidf12 = TfidfVectorizer(stop_words='english', ngram_range=(1,2))

X_train_tfidf12_sw = tfidf12.fit_transform(train3_clean['text'])
y_train_tfidf12_sw = train3_clean['Recommended']
X_test_tfidf12_sw = tfidf12.transform(test3_clean['text'])
y_test_tfidf12_sw = test3_clean['Recommended']
feature_names_tfidf12_sw = tfidf12.get_feature_names()
logreg_tfidf12_sw = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf12_sw.fit(X_train_tfidf12_sw,y_train_tfidf12_sw)
print('test score:',logreg_tfidf12_sw.score(X_test_tfidf12_sw,y_test_tfidf12_sw))
```

```
test score: 0.9043354365967841
```

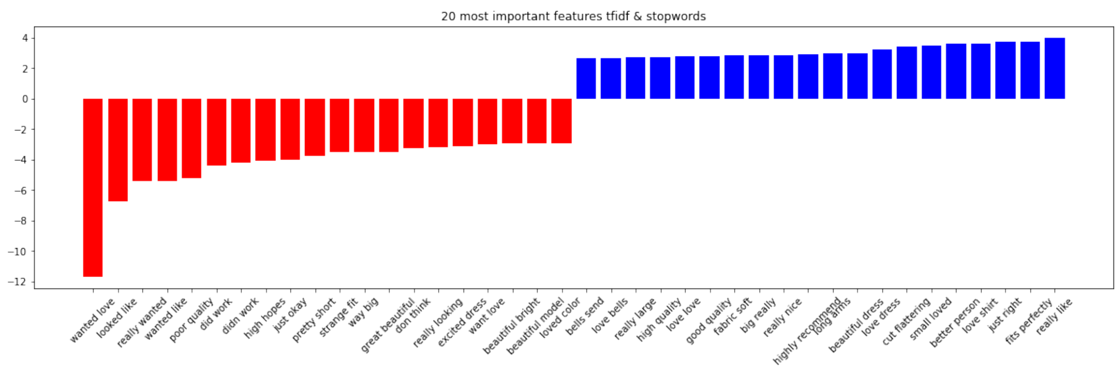
In [120]:
```
print("Test Avg Precision score: ", average_precision_score(logreg_tfidf12_sw.predict(
print("Test F1 score: ", f1_score(logreg_tfidf12_sw.predict(X_test_tfidf12_sw),y_test_
print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf12_sw.predict(X_test_tfidf12_s
```

```
Test Avg Precision score:  0.9562402857747404
Test F1 score:  0.9427666829030686
Test ROC AUC score:  0.8619553529686769
```

In [141]:
```
plot2(logreg_tfidf12_sw.coef_[0],feature_names_tfidf12_sw,'tfidf & stopwords')
```

```
[-11.69801927  -6.73712187  -5.41815033  -5.39648805  -5.20211385
  -4.36842375  -4.19307605  -4.06430097  -3.98936339  -3.77680496
  -3.51468419  -3.49765593  -3.47684654  -3.27040375  -3.18715634
```

```
 -3.15306869  -2.9793406   -2.95383939  -2.92573818  -2.91619746
  2.66479549   2.66479549   2.69770799   2.72157348   2.77150483
  2.79014634   2.80854011   2.80901199   2.86332113   2.88463928
  2.9377724    2.94774184   3.24449917   3.41012066   3.48870708
  3.60972567   3.61990861   3.74131716   3.74536562   3.98295195]
['wanted love' 'looked like' 'really wanted' 'wanted like' 'poor quality'
 'did work' 'didn work' 'high hopes' 'just okay' 'pretty short'
 'strange fit' 'way big' 'great beautiful' 'don think' 'really looking'
 'excited dress' 'want love' 'beautiful bright' 'beautiful model'
 'loved color' 'bells send' 'love bells' 'really large' 'high quality'
 'love love' 'good quality' 'fabric soft' 'big really' 'really nice'
 'highly recommend' 'long arms' 'beautiful dress' 'love dress'
 'cut flattering' 'small loved' 'better person' 'love shirt' 'just right'
 'fits perfectly' 'really like']
```



In [122]:
```
logreg_tfidf12_sw.coef_[0]
```

Out[122]:
```
array([0.92111827, 0.06139087, 0.19457047, ..., 0.07799795, 0.05367791,
       0.05367791])
```

In [123]:
```
#from sklearn.feature_extraction.text import TfidfVectorizer
from collections import defaultdict

#lectures = ["this is some food", "this is some drink"]
#vectorizer = TfidfVectorizer(ngram_range=(1,2))
#X = vectorizer.fit_transform(lectures)
features_by_gram = defaultdict(list)
for f, w in zip(feature_names_tfidf12_sw, tfidf12.idf_):
    features_by_gram[len(f.split(' '))].append((f, w))
top_n = 2
for gram, features in features_by_gram.items():
    top_features = sorted(features, key=lambda x: x[1], reverse=True)[:top_n]
    top_features = [f[0] for f in top_features]
    print('{}-gram top:'.format(gram), top_features)
```

```
1-gram top: ['03', '03dd']
2-gram top: ['00 115', '00 24']
```

```
In [363]: tfidf33 = TfidfVectorizer(stop_words='english', ngram_range=(3,3))

          X_train_tfidf33_sw = tfidf33.fit_transform(train3_clean['text'])
          y_train_tfidf33_sw = train3_clean['Recommended']
          X_test_tfidf33_sw = tfidf33.transform(test3_clean['text'])
          y_test_tfidf33_sw = test3_clean['Recommended']
          feature_names_tfidf33_sw = tfidf33.get_feature_names()
          logreg_tfidf33_sw = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf33_sw.fit(X_train_tfidf33_sw,y_train_tfidf33_sw)
          print('test score:',logreg_tfidf33_sw.score(X_test_tfidf33_sw,y_test_tfidf33_sw))
```

```
test score: 0.8173652694610778
```

```
In [ ]:
```

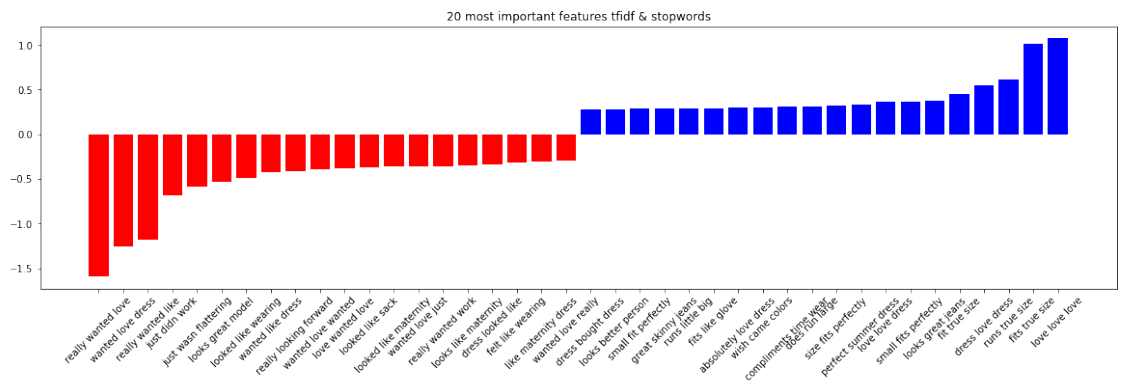
```
In [366]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_sw.predict(
          print("Test F1 score: ", f1_score(logreg_tfidf33_sw.predict(X_test_tfidf33_sw),y_test_
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_sw.predict(X_test_tfidf33_s
```

```
Test Avg Precision score:  0.9981326500764675 6
Test F1 score:  0.8995057660626029
Test ROC AUC score:  0.8663098002114658389
```

```
In [143]: plot(logreg_tfidf33_sw.coef_[0],feature_names_tfidf33_sw,'tfidf & stopwords')
```

```
[0.27769891 0.28144258 0.28323473 0.29069194 0.29079684 0.29129772
 0.29646217 0.30172146 0.31197023 0.31270705 0.31885576 0.33036784
 0.36047008 0.36620191 0.3714412  0.44548365 0.54874357 0.61409926
 1.00756163 1.07588418]
['dress bought dress', 'looks better person', 'small fit perfectly', 'great skinny jeans', 'runs
[-1.59321089 -1.26093855 -1.18147231 -0.68270773 -0.58649065 -0.53065975
 -0.48406289 -0.42886997 -0.41875689 -0.39347424 -0.38049487 -0.36596123
 -0.36422625 -0.35642819 -0.35546267 -0.34494998 -0.33289932 -0.3155874
 -0.30898285 -0.29817724]
['really wanted love', 'wanted love dress', 'really wanted like', 'just didn work', 'just wasn f
[-1.59321089 -1.26093855 -1.18147231 -0.68270773 -0.58649065 -0.53065975
 -0.48406289 -0.42886997 -0.41875689 -0.39347424 -0.38049487 -0.36596123
 -0.36422625 -0.35642819 -0.35546267 -0.34494998 -0.33289932 -0.3155874
 -0.30898285 -0.29817724  0.27769891  0.28144258  0.28323473  0.29069194
  0.29079684  0.29129772  0.29646217  0.30172146  0.31197023  0.31270705
  0.31885576  0.33036784  0.36047008  0.36620191  0.3714412   0.44548365
  0.54874357  0.61409926  1.00756163  1.07588418]
['really wanted love' 'wanted love dress' 'really wanted like'
```

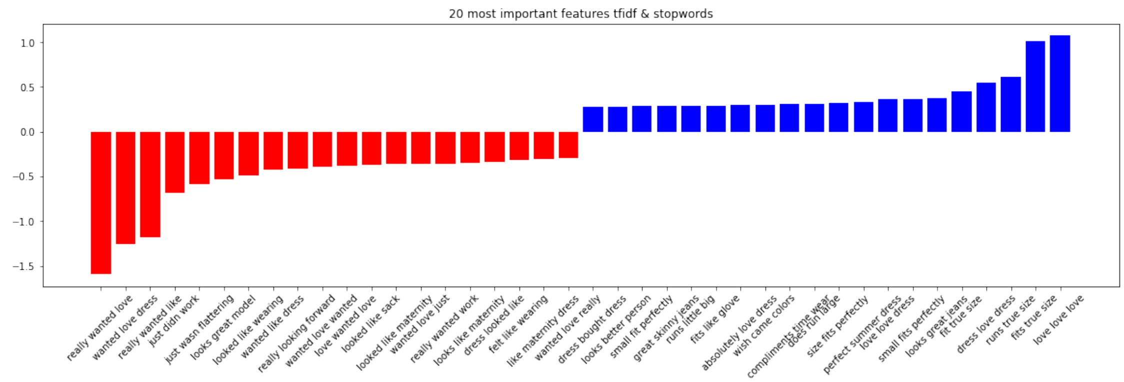
```
 'just didn work' 'just wasn flattering' 'looks great model'
 'looked like wearing' 'wanted like dress' 'really looking forward'
 'wanted love wanted' 'love wanted love' 'looked like sack'
 'looked like maternity' 'wanted love just' 'really wanted work'
 'looks like maternity' 'dress looked like' 'felt like wearing'
 'like maternity dress' 'wanted love really' 'dress bought dress'
 'looks better person' 'small fit perfectly' 'great skinny jeans'
 'runs little big' 'fits like glove' 'absolutely love dress'
 'wish came colors' 'compliments time wear' 'does run large'
 'size fits perfectly' 'perfect summer dress' 'love love dress'
 'small fits perfectly' 'looks great jeans' 'fit true size'
 'dress love dress' 'runs true size' 'fits true size' 'love love love']
```




```
In [144]: plot2(logreg_tfidf33_sw.coef_[0],feature_names_tfidf33_sw,'tfidf & stopwords')
```

```
[-1.59321089 -1.26093855 -1.18147231 -0.68270773 -0.58649065 -0.53065975
 -0.48406289 -0.42886997 -0.41875689 -0.39347424 -0.38049487 -0.36596123
 -0.36422625 -0.35642819 -0.35546267 -0.34494998 -0.33289932 -0.3155874
 -0.30898285 -0.29817724  0.27769891  0.28144258  0.28323473  0.29069194
  0.29079684  0.29129772  0.29646217  0.30172146  0.31197023  0.31270705
  0.31885576  0.33036784  0.36047008  0.36620191  0.3714412   0.44548365
  0.54874357  0.61409926  1.00756163  1.07588418]
['really wanted love' 'wanted love dress' 'really wanted like'
 'just didn work' 'just wasn flattering' 'looks great model'
 'looked like wearing' 'wanted like dress' 'really looking forward'
 'wanted love wanted' 'love wanted love' 'looked like sack'
 'looked like maternity' 'wanted love just' 'really wanted work'
 'looks like maternity' 'dress looked like' 'felt like wearing'
 'like maternity dress' 'wanted love really' 'dress bought dress'
 'looks better person' 'small fit perfectly' 'great skinny jeans'
 'runs little big' 'fits like glove' 'absolutely love dress'
 'wish came colors' 'compliments time wear' 'does run large'
 'size fits perfectly' 'perfect summer dress' 'love love dress'
```

```
 'small fits perfectly' 'looks great jeans' 'fit true size'
 'dress love dress' 'runs true size' 'fits true size' 'love love love']
```




```
In [334]: tfidf33 = TfidfVectorizer(ngram_range=(3,3))

          X_train_tfidf33 = tfidf33.fit_transform(train3_clean['text'])
          y_train_tfidf33 = train3_clean['Recommended']
          X_test_tfidf33 = tfidf33.transform(test3_clean['text'])
          y_test_tfidf33 = test3_clean['Recommended']
          feature_names_tfidf33 = tfidf33.get_feature_names()
          logreg_tfidf33 = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf33.fit(X_train_tfidf33,y_train_tfidf33)
          print('test score:',logreg_tfidf33.score(X_test_tfidf33,y_test_tfidf33))
```

```
test score: 0.8696243875884594
```

```
In [338]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33.predict(X_t
          print("Test F1 score: ", f1_score(logreg_tfidf33.predict(X_test_tfidf33),y_test_tfidf3
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33.predict(X_test_tfidf33),y_t
```

```
Test Avg Precision score:  0.9760725673059305
Test F1 score:  0.9250273908279857
Test ROC AUC score:  0.8530222484741091
```

```
In [339]: X_train_tfidf33_sw
```

```
Out[339]: <14762x353532 sparse matrix of type '<class 'numpy.float64'>'
                  with 392215 stored elements in Compressed Sparse Row format>
```

```
In [340]: X_train_tfidf33
```

```
Out[340]: <11088x388498 sparse matrix of type '<class 'numpy.float64'>'
                  with 637545 stored elements in Compressed Sparse Row format>
```

```
In [126]: tfidf23 = TfidfVectorizer(stop_words='english', ngram_range=(2,3))

          X_train_tfidf23_sw = tfidf23.fit_transform(train3_clean['text'])
          y_train_tfidf23_sw = train3_clean['Recommended']
          X_test_tfidf23_sw = tfidf23.transform(test3_clean['text'])
          y_test_tfidf23_sw = test3_clean['Recommended']
          feature_names_tfidf23_sw = tfidf23.get_feature_names()
          logreg_tfidf23_sw = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf23_sw.fit(X_train_tfidf23_sw,y_train_tfidf23_sw)
          print('test score:',logreg_tfidf23_sw.score(X_test_tfidf23_sw,y_test_tfidf23_sw))
```

```
test score: 0.8662731528597598
```

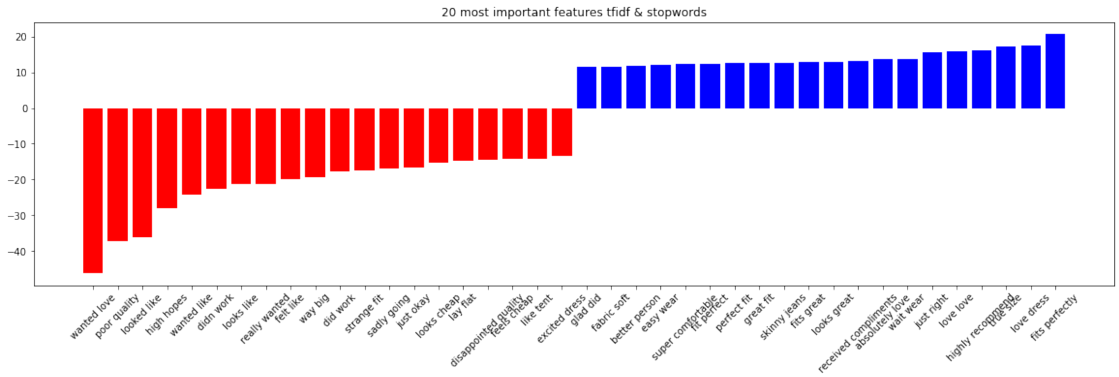
```
In [127]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf23_sw.predict(
          print("Test F1 score: ", f1_score(logreg_tfidf23_sw.predict(X_test_tfidf23_sw),y_test_
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf23_sw.predict(X_test_tfidf23_s
```

```
Test Avg Precision score:  0.9702760317758865
Test F1 score:  0.9227694839543905
Test ROC AUC score:  0.8325292323740892
```

```
In [129]: plot(logreg_tfidf23_sw.coef_[0],feature_names_tfidf23_sw,'tfidf & stopwords')
```

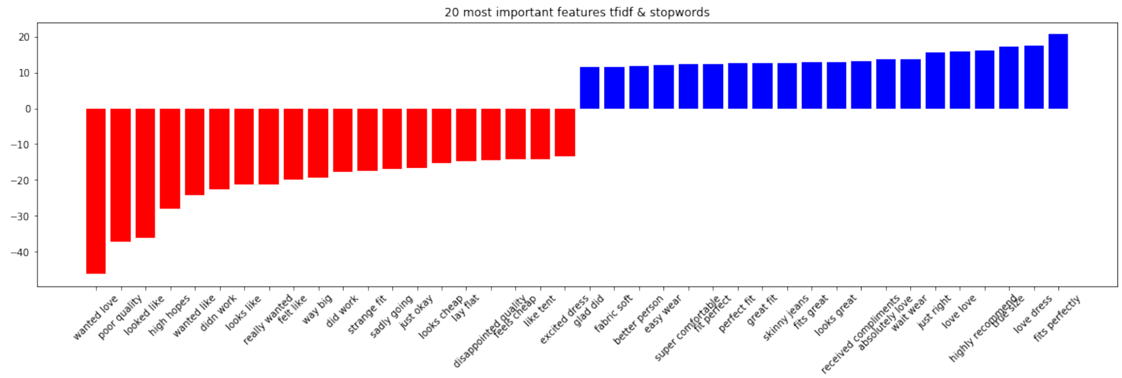
```
[11.48649225 11.50350149 11.8966814  11.94115261 12.21720419 12.41381576
 12.45775746 12.46985121 12.5009868  12.74842429 12.81347263 13.14136791
 13.75151701 13.7834796  15.61901639 15.9444128  16.00208149 17.20289754
 17.45909013 20.68657209]
['glad did', 'fabric soft', 'better person', 'easy wear', 'super comfortable', 'fit perfect', 'p
[-46.18113309 -37.2860512  -36.23934701 -27.90896706 -24.24185099
 -22.68180366 -21.25844586 -21.17137736 -19.83464956 -19.37035545
 -17.72350007 -17.57990174 -16.79285084 -16.71574845 -15.31566129
 -14.67284075 -14.40880132 -14.17441056 -14.15149482 -13.27736212]
['wanted love', 'poor quality', 'looked like', 'high hopes', 'wanted like', 'didn work', 'looks
[-46.18113309 -37.2860512  -36.23934701 -27.90896706 -24.24185099
 -22.68180366 -21.25844586 -21.17137736 -19.83464956 -19.37035545
 -17.72350007 -17.57990174 -16.79285084 -16.71574845 -15.31566129
 -14.67284075 -14.40880132 -14.17441056 -14.15149482 -13.27736212
  11.48649225  11.50350149  11.8966814   11.94115261  12.21720419
  12.41381576  12.45775746  12.46985121  12.5009868   12.74842429
  12.81347263  13.14136791  13.75151701  13.7834796   15.61901639
  15.9444128   16.00208149  17.20289754  17.45909013  20.68657209]
['wanted love' 'poor quality' 'looked like' 'high hopes' 'wanted like'
 'didn work' 'looks like' 'really wanted' 'felt like' 'way big' 'did work'
 'strange fit' 'sadly going' 'just okay' 'looks cheap' 'lay flat'
 'disappointed quality' 'feels cheap' 'like tent' 'excited dress'
 'glad did' 'fabric soft' 'better person' 'easy wear' 'super comfortable'
 'fit perfect' 'perfect fit' 'great fit' 'skinny jeans' 'fits great'
```

```
 'looks great' 'received compliments' 'absolutely love' 'wait wear'
 'just right' 'love love' 'highly recommend' 'true size' 'love dress'
 'fits perfectly']
```




```
In [145]: plot2(logreg_tfidf23_sw.coef_[0],feature_names_tfidf23_sw,'tfidf & stopwords')
```

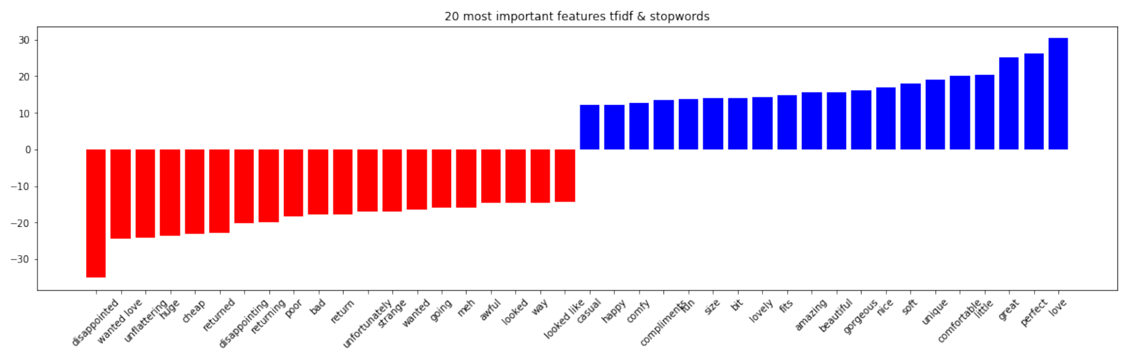
```
[-46.18113309 -37.2860512  -36.23934701 -27.90896706 -24.24185099
 -22.68180366 -21.25844586 -21.17137736 -19.83464956 -19.37035545
 -17.72350007 -17.57990174 -16.79285084 -16.71574845 -15.31566129
 -14.67284075 -14.40880132 -14.17441056 -14.15149482 -13.27736212
  11.48649225  11.50350149  11.8966814   11.94115261  12.21720419
  12.41381576  12.45775746  12.46985121  12.5009868   12.74842429
  12.81347263  13.14136791  13.75151701  13.7834796   15.61901639
  15.9444128   16.00208149  17.20289754  17.45909013  20.68657209]
['wanted love' 'poor quality' 'looked like' 'high hopes' 'wanted like'
 'didn work' 'looks like' 'really wanted' 'felt like' 'way big' 'did work'
 'strange fit' 'sadly going' 'just okay' 'looks cheap' 'lay flat'
 'disappointed quality' 'feels cheap' 'like tent' 'excited dress'
 'glad did' 'fabric soft' 'better person' 'easy wear' 'super comfortable'
 'fit perfect' 'perfect fit' 'great fit' 'skinny jeans' 'fits great'
 'looks great' 'received compliments' 'absolutely love' 'wait wear'
 'just right' 'love love' 'highly recommend' 'true size' 'love dress'
 'fits perfectly']
```




```
In [130]: tfidf13 = TfidfVectorizer(stop_words='english', ngram_range=(1,3))

          X_train_tfidf13_sw = tfidf13.fit_transform(train3_clean['text'])
          y_train_tfidf13_sw = train3_clean['Recommended']
          X_test_tfidf13_sw = tfidf13.transform(test3_clean['text'])
          y_test_tfidf13_sw = test3_clean['Recommended']
          feature_names_tfidf13_sw = tfidf13.get_feature_names()
          logreg_tfidf13_sw = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf13_sw.fit(X_train_tfidf13_sw,y_train_tfidf13_sw)
          print('test score:',logreg_tfidf13_sw.score(X_test_tfidf13_sw,y_test_tfidf13_sw))

          print("Test Avg Precision score: ", average_precision_score(logreg_tfidf13_sw.predict(
          print("Test F1 score: ", f1_score(logreg_tfidf13_sw.predict(X_test_tfidf13_sw),y_test_
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf13_sw.predict(X_test_tfidf13_s


          plot(logreg_tfidf13_sw.coef_[0],feature_names_tfidf13_sw,'tfidf & stopwords')
```

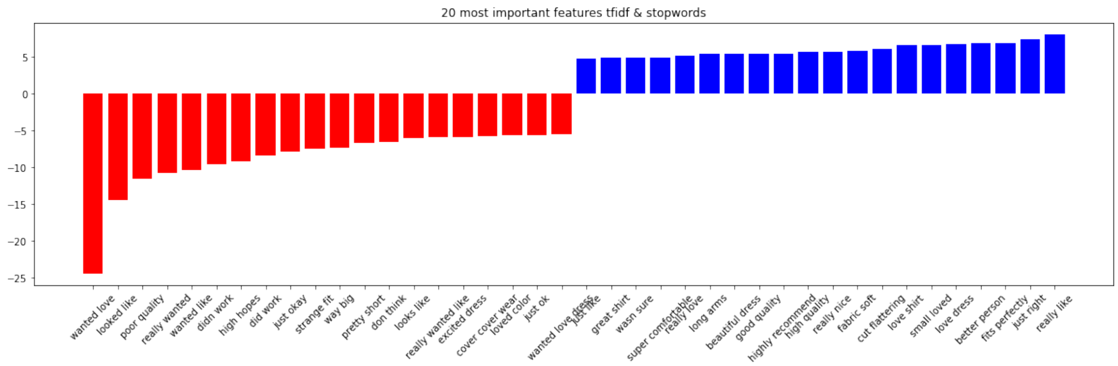
```
test score: 0.9035212700997354
Test Avg Precision score:  0.950951056407899
Test F1 score:  0.9419970631424376
Test ROC AUC score:  0.8543864605679844
[12.05848816 12.16095043 12.61447121 13.42765095 13.81425072 13.93160812
 14.03745156 14.41426652 14.79590002 15.56441722 15.57226103 16.22496061
 16.95067579 18.03222937 19.10086077 20.20236878 20.38469168 25.05160831
 26.09879116 30.3913551 ]
['casual', 'happy', 'comfy', 'compliments', 'fun', 'size', 'bit', 'lovely', 'fits', 'amazing', '
[-35.07239766 -24.43958457 -24.20977962 -23.57648968 -23.13928364
 -22.86799576 -20.05282738 -19.89092423 -18.2893465  -17.89409231
 -17.79872131 -17.01417623 -16.96278907 -16.46124946 -16.00636427
 -15.86844172 -14.66344062 -14.62695099 -14.51920369 -14.42157622]
['disappointed', 'wanted love', 'unflattering', 'huge', 'cheap', 'returned', 'disappointing', 'r
[-35.07239766 -24.43958457 -24.20977962 -23.57648968 -23.13928364
 -22.86799576 -20.05282738 -19.89092423 -18.2893465  -17.89409231
```

```
 -17.79872131 -17.01417623 -16.96278907 -16.46124946 -16.00636427
 -15.86844172 -14.66344062 -14.62695099 -14.51920369 -14.42157622
  12.05848816  12.16095043  12.61447121  13.42765095  13.81425072
  13.93160812  14.03745156  14.41426652  14.79590002  15.56441722
  15.57226103  16.22496061  16.95067579  18.03222937  19.10086077
  20.20236878  20.38469168  25.05160831  26.09879116  30.3913551 ]
['disappointed' 'wanted love' 'unflattering' 'huge' 'cheap' 'returned'
 'disappointing' 'returning' 'poor' 'bad' 'return' 'unfortunately'
 'strange' 'wanted' 'going' 'meh' 'awful' 'looked' 'way' 'looked like'
 'casual' 'happy' 'comfy' 'compliments' 'fun' 'size' 'bit' 'lovely' 'fits'
 'amazing' 'beautiful' 'gorgeous' 'nice' 'soft' 'unique' 'comfortable'
 'little' 'great' 'perfect' 'love']
```



In [146]: plot2(logreg_tfidf13_sw.coef_[0],feature_names_tfidf13_sw,'tfidf & stopwords')

```
[-24.43958457 -14.42157622 -11.49482339 -10.80945325 -10.38391307
  -9.5719569   -9.2460921   -8.45323035  -7.90508495  -7.53092583
  -7.31302445  -6.62910219  -6.52847423  -6.04965721  -5.94973235
  -5.87872434  -5.80629848  -5.66728998  -5.58763017  -5.56896563
   4.81270877   4.83722476   4.89190076   4.90371438   5.18269343
   5.34201402   5.38470411   5.40376242   5.4449024    5.61922513
   5.6628399    5.84687262   6.04975015   6.54965658   6.55472482
   6.78281017   6.80752266   6.83318307   7.32445314   8.03433286]
['wanted love' 'looked like' 'poor quality' 'really wanted' 'wanted like'
 'didn work' 'high hopes' 'did work' 'just okay' 'strange fit' 'way big'
 'pretty short' 'don think' 'looks like' 'really wanted like'
 'excited dress' 'cover cover wear' 'loved color' 'just ok'
 'wanted love dress' 'just like' 'great shirt' 'wasn sure'
 'super comfortable' 'really love' 'long arms' 'beautiful dress'
 'good quality' 'highly recommend' 'high quality' 'really nice'
 'fabric soft' 'cut flattering' 'love shirt' 'small loved' 'love dress'
 'better person' 'fits perfectly' 'just right' 'really like']
```



(3,3) gives best performance

3.2 Running character n-grams

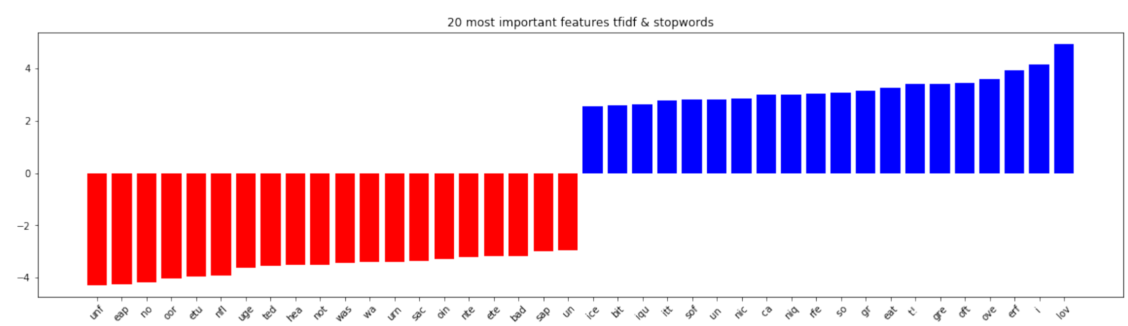
```
In [ ]: cv = CountVectorizer(ngram_range=(2, 3), analyzer="char_wb").fit(malory)
        print("Vocabulary size: {}".format(len(cv.vocabulary_)))
        print("Vocabulary:\n{}".format(cv.get_feature_names()))
```

```
In [ ]: cv = CountVectorizer(ngram_range=(2, 3), analyzer="char").fit(malory)
        print("Vocabulary size: {}".format(len(cv.vocabulary_)))
        print("Vocabulary:\n{}".format(cv.get_feature_names()))
```

```
In [133]: tfidf33_chr = TfidfVectorizer(stop_words='english', ngram_range=(3,3),analyzer="char_w

          X_train_tfidf33_chr_sw = tfidf33_chr.fit_transform(train3_clean['text'])
          y_train_tfidf33_chr_sw = train3_clean['Recommended']
          X_test_tfidf33_chr_sw = tfidf33_chr.transform(test3_clean['text'])
          y_test_tfidf33_chr_sw = test3_clean['Recommended']
          feature_names_tfidf33_chr_sw = tfidf33_chr.get_feature_names()
          logreg_tfidf33_chr_sw = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf33_chr_sw.fit(X_train_tfidf33_chr_sw,y_train_tfidf33_chr_sw)
          print('test score:',logreg_tfidf33_chr_sw.score(X_test_tfidf33_chr_sw,y_test_tfidf33_c

          print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_chr_sw.pred
          print("Test F1 score: ", f1_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf33_chr_sw)
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf


          plot(logreg_tfidf33_chr_sw.coef_[0],feature_names_tfidf33_chr_sw,'tfidf & stopwords')
```

```
test score: 0.8984327294931813
Test Avg Precision score:  0.9476195968337298
Test F1 score:  0.9389303634806021
Test ROC AUC score:  0.8444576748006783
[2.55712537 2.60525699 2.64089948 2.77183922 2.80021868 2.8158045
```

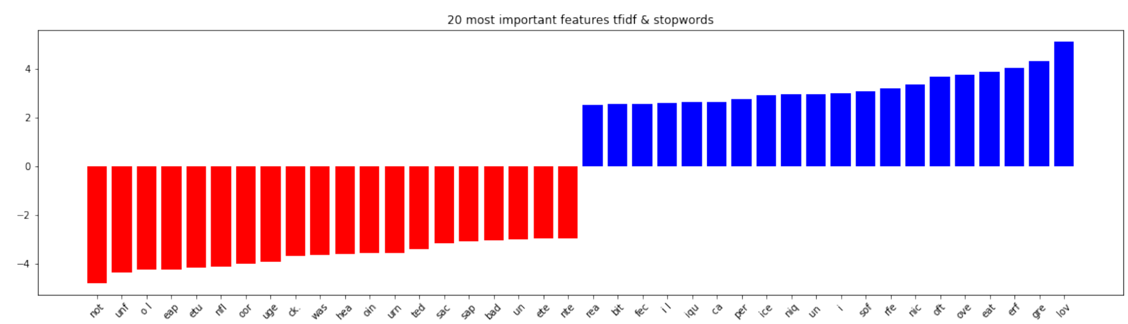
```
 2.85277669 2.99110441 2.9920341  3.03332472 3.06048308 3.15120968
 3.24477052 3.40246434 3.40674582 3.42982084 3.59062062 3.91414783
 4.15804706 4.91443257]
['ice', 'bit', 'iqu', 'itt', 'sof', 'un ', 'nic', ' ca', 'niq', 'rfe', ' so', ' gr', 'eat', 't!
[-4.27253072 -4.2604215  -4.18666862 -4.03839552 -3.94989989 -3.91756218
 -3.62350597 -3.56162864 -3.52295726 -3.4971716  -3.42789419 -3.41355136
 -3.41123275 -3.35153164 -3.277063   -3.21611191 -3.17979266 -3.16097599
 -2.9934303  -2.96598155]
['unf', 'eap', ' no', 'oor', 'etu', 'nfl', 'uge', 'ted', 'hea', 'not', 'was', ' wa', 'urn', 'sac
[-4.27253072 -4.2604215  -4.18666862 -4.03839552 -3.94989989 -3.91756218
 -3.62350597 -3.56162864 -3.52295726 -3.4971716  -3.42789419 -3.41355136
 -3.41123275 -3.35153164 -3.277063   -3.21611191 -3.17979266 -3.16097599
 -2.9934303  -2.96598155  2.55712537  2.60525699  2.64089948  2.77183922
  2.80021868  2.8158045   2.85277669  2.99110441  2.9920341   3.03332472
  3.06048308  3.15120968  3.24477052  3.40246434  3.40674582  3.42982084
  3.59062062  3.91414783  4.15804706  4.91443257]
['unf' 'eap' ' no' 'oor' 'etu' 'nfl' 'uge' 'ted' 'hea' 'not' 'was' ' wa'
 'urn' 'sac' 'oin' 'nte' 'ete' 'bad' 'sap' ' un' 'ice' 'bit' 'iqu' 'itt'
 'sof' 'un ' 'nic' ' ca' 'niq' 'rfe' ' so' ' gr' 'eat' 't! ' 'gre' 'oft'
 'ove' 'erf' ' i ' 'lov']
```



In [134]:
```
tfidf33_chr = TfidfVectorizer(stop_words='english', ngram_range=(3,3),analyzer="char")

X_train_tfidf33_chr_sw = tfidf33_chr.fit_transform(train3_clean['text'])
y_train_tfidf33_chr_sw = train3_clean['Recommended']
X_test_tfidf33_chr_sw = tfidf33_chr.transform(test3_clean['text'])
y_test_tfidf33_chr_sw = test3_clean['Recommended']
feature_names_tfidf33_chr_sw = tfidf33_chr.get_feature_names()
logreg_tfidf33_chr_sw = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf33_chr_sw.fit(X_train_tfidf33_chr_sw,y_train_tfidf33_chr_sw)
print('test score:',logreg_tfidf33_chr_sw.score(X_test_tfidf33_chr_sw,y_test_tfidf33_c

print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_chr_sw.pred
print("Test F1 score: ", f1_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf33_chr_sw)
```

```
print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf

plot(logreg_tfidf33_chr_sw.coef_[0],feature_names_tfidf33_chr_sw,'tfidf & stopwords')
```

```
test score: 0.8998575208630165
Test Avg Precision score:  0.9495930945416644
Test F1 score:  0.9398533007334964
Test ROC AUC score:  0.8483807597676127
[2.53179396 2.54387805 2.5633725  2.60092904 2.61727002 2.65238405
 2.77378936 2.90425295 2.94889685 2.96558084 3.00731959 3.05826113
 3.2130584  3.35480093 3.67987551 3.74581866 3.88473375 4.03231939
 4.31581204 5.09590356]
['rea', 'bit', 'fec', 'i l', 'iqu', ' ca', 'per', 'ice', 'niq', 'un ', ' i ', 'sof', 'rfe', 'nic
[-4.79294832 -4.38634511 -4.25171691 -4.24989003 -4.16154312 -4.12123615
 -4.01792948 -3.94057573 -3.67528792 -3.63555593 -3.61485619 -3.58595399
 -3.58160198 -3.40786851 -3.17743627 -3.10763308 -3.03351723 -3.02334675
 -2.97512045 -2.97318553]
['not', 'unf', 'o l', 'eap', 'etu', 'nfl', 'oor', 'uge', 'ck.', 'was', 'hea', 'oin', 'urn', 'ted
[-4.79294832 -4.38634511 -4.25171691 -4.24989003 -4.16154312 -4.12123615
 -4.01792948 -3.94057573 -3.67528792 -3.63555593 -3.61485619 -3.58595399
 -3.58160198 -3.40786851 -3.17743627 -3.10763308 -3.03351723 -3.02334675
 -2.97512045 -2.97318553  2.53179396  2.54387805  2.5633725   2.60092904
  2.61727002  2.65238405  2.77378936  2.90425295  2.94889685  2.96558084
  3.00731959  3.05826113  3.2130584   3.35480093  3.67987551  3.74581866
  3.88473375  4.03231939  4.31581204  5.09590356]
['not' 'unf' 'o l' 'eap' 'etu' 'nfl' 'oor' 'uge' 'ck.' 'was' 'hea' 'oin'
 'urn' 'ted' 'sac' 'sap' 'bad' ' un' 'ete' 'nte' 'rea' 'bit' 'fec' 'i l'
 'iqu' ' ca' 'per' 'ice' 'niq' 'un ' ' i ' 'sof' 'rfe' 'nic' 'oft' 'ove'
 'eat' 'erf' 'gre' 'lov']
```

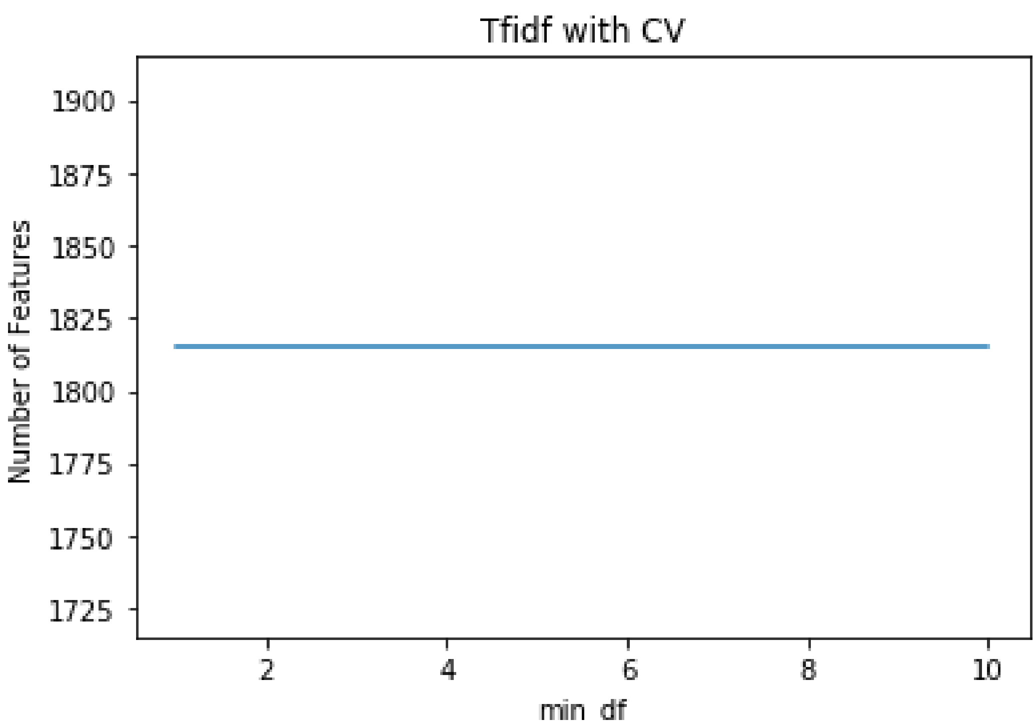


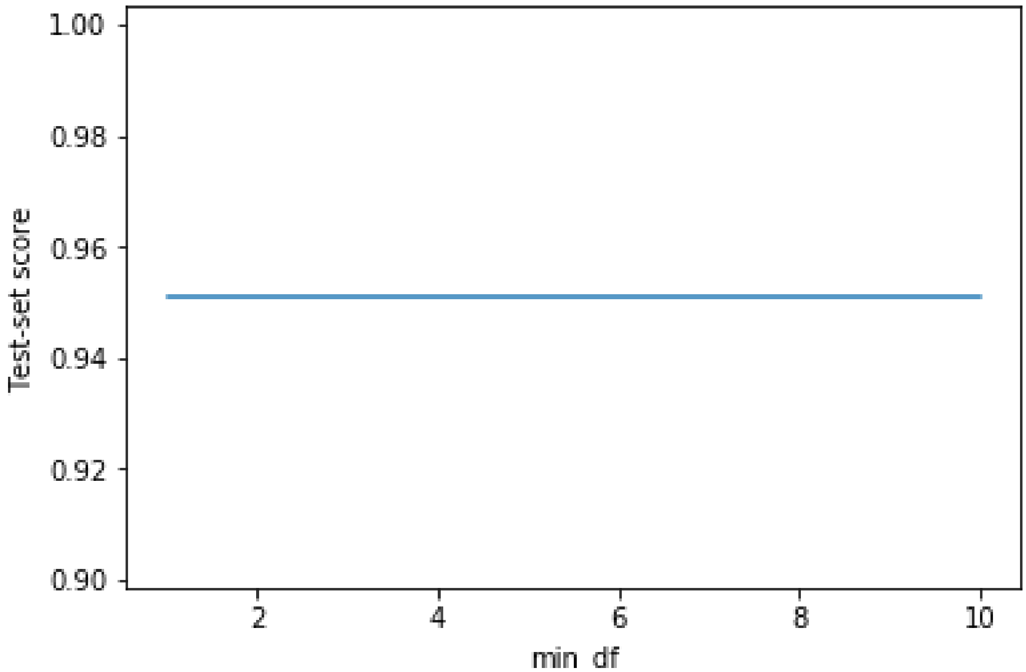
3.3 Use of min-df and stop words

In [ ]:

In [341]:
```
feature_count_tfidf_cv = []
test_scores_tfidf_cv = []
#default params and tfidf
for i in range(2,11):
    tfidf = TfidfVectorizer(stop_words='english', ngram_range=(2,2),analyzer="char")
    X_train = tfidf.fit_transform(train3_clean['text'])
    y_train = train3_clean['Recommended']
    X_test = tfidf.transform(test3_clean['text'])
    y_test = test3_clean['Recommended']
    feature_names = tfidf.get_feature_names()
    feature_count_tfidf_cv.append(len(feature_names))
    logreg = LogisticRegressionCV(scoring = 'average_precision')
    logreg.fit(X_train,y_train)
    test_scores_tfidf_cv.append(average_precision_score(logreg.predict(X_test),y_test)
```

In [342]:
```
min_df(feature_count_tfidf_cv,test_scores_tfidf_cv,'Tfidf with CV')
```






```
In [ ]:
```

From average precision score and roc auc we observe that tfidf with 3,3 n grams i.e using 3 gram features alone we ge the best results Also, for the best model case the number of features after using stop words reduces from ~388k to ~353k which is a 9% reduction with a very slight drop (2%) in average precision score.

For the min-df case we observe that combinations of n grams are not very likely to reappear. THus the number of features does not vary accross different values of min-df. However, the number of features drastically reduces from to 156k to 1.8k

Now evaluating the best performing model on the main test set. In this case the best performing model on the basis of average_precision and roc_auc is 3,3 n gram with tfidf

```
In [ ]:
```

```
In [372]: tfidf33 = TfidfVectorizer(ngram_range=(3,3))

          X_train_tfidf33_sw = tfidf33.fit_transform(train3_clean['text'])
          y_train_tfidf33_sw = train3_clean['Recommended']
          X_test_tfidf33_sw = tfidf33.transform(test3_clean['text'])
          y_test_tfidf33_sw = test3_clean['Recommended']
          feature_names_tfidf33_sw = tfidf33.get_feature_names()
          logreg_tfidf33_sw = LogisticRegressionCV(scoring = 'average_precision')
          logreg_tfidf33_sw.fit(X_train_tfidf33_sw,y_train_tfidf33_sw)
          print('test score:',logreg_tfidf33_sw.score(X_test_tfidf33_sw,y_test_tfidf33_sw))
```

```
test score: 0.8696243875884594
```

```
In [373]: X_test_4 = tfidf33.transform(test_main_clean["text"])

In [ ]:

In [374]: print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_sw.predict(
          print("Test F1 score: ", f1_score(logreg_tfidf33_sw.predict(X_test_4),y_test_tfidf33_c
          print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_sw.predict(X_test_4),y_test
```

```
Test Avg Precision score:  0.9752228323332524
Test F1 score:  0.9211540707861232
Test ROC AUC score:  0.8386117344954452
```

```
In [ ]:

In [ ]:

In [ ]:
```

1 Task 4

```
In [ ]:

In [ ]:

In [ ]:

In [ ]: tfidf33_chr = TfidfVectorizer(stop_words='english', ngram_range=(3,3),analyzer="char_wb"

        X_train_tfidf33_chr_sw = tfidf33_chr.fit_transform(train3_clean['text'])
        y_train_tfidf33_chr_sw = train3_clean['Recommended']
        X_test_tfidf33_chr_sw = tfidf33_chr.transform(test3_clean['text'])
        y_test_tfidf33_chr_sw = test3_clean['Recommended']
        feature_names_tfidf33_chr_sw = tfidf33_chr.get_feature_names()
        logreg_tfidf33_chr_sw = LogisticRegressionCV(scoring = 'average_precision')
        logreg_tfidf33_chr_sw.fit(X_train_tfidf33_chr_sw,y_train_tfidf33_chr_sw)
        print('test score:',logreg_tfidf33_chr_sw.score(X_test_tfidf33_chr_sw,y_test_tfidf33_chr

        print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_chr_sw.predic
        print("Test F1 score: ", f1_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf33_chr_sw),y
        print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf33


        plot(logreg_tfidf33_chr_sw.coef_[0],feature_names_tfidf33_chr_sw,'tfidf & stopwords')
```

In [197]:
```
top20_names = ['really wanted love' ,'wanted love dress' ,'really wanted like',
 'just didn work', 'just wasn flattering' ,'looks great model',
 'looked like wearing', 'wanted like dress' ,'really looking forward',
 'wanted love wanted' ,'love wanted love', 'looked like sack',
 'looked like maternity', 'wanted love just' ,'really wanted work',
 'looks like maternity' ,'dress looked like' ,'felt like wearing',
 'like maternity dress', 'wanted love really' ,'dress bought dress',
 'looks better person' ,'small fit perfectly', 'great skinny jeans',
 'runs little big', 'fits like glove', 'absolutely love dress',
 'wish came colors' ,'compliments time wear' ,'does run large',
 'size fits perfectly', 'perfect summer dress' ,'love love dress',
 'small fits perfectly', 'looks great jeans', 'fit true size',
 'dress love dress', 'runs true size', 'fits true size' ,'love love love']
```

In [198]:
```
top20_names
```

Out[198]:
```
['really wanted love',
 'wanted love dress',
 'really wanted like',
 'just didn work',
 'just wasn flattering',
 'looks great model',
 'looked like wearing',
 'wanted like dress',
 'really looking forward',
 'wanted love wanted',
 'love wanted love',
 'looked like sack',
 'looked like maternity',
 'wanted love just',
 'really wanted work',
 'looks like maternity',
 'dress looked like',
 'felt like wearing',
 'like maternity dress',
 'wanted love really',
 'dress bought dress',
 'looks better person',
 'small fit perfectly',
 'great skinny jeans',
 'runs little big',
 'fits like glove',
 'absolutely love dress',
 'wish came colors',
 'compliments time wear',
 'does run large',
 'size fits perfectly',
 'perfect summer dress',
```

```
'love love dress',
 'small fits perfectly',
 'looks great jeans',
 'fit true size',
 'dress love dress',
 'runs true size',
 'fits true size',
 'love love love']
```

In [199]:
```
voc = " ".join(top20_names)
```

In [200]:
```
voc
```

Out[200]:
```
'really wanted love wanted love dress really wanted like just didn work just wasn flat
```

In [ ]:

3,3 is the best performing model. THerefore, here we train the model on only on the top 20 features. Thus the training value will have

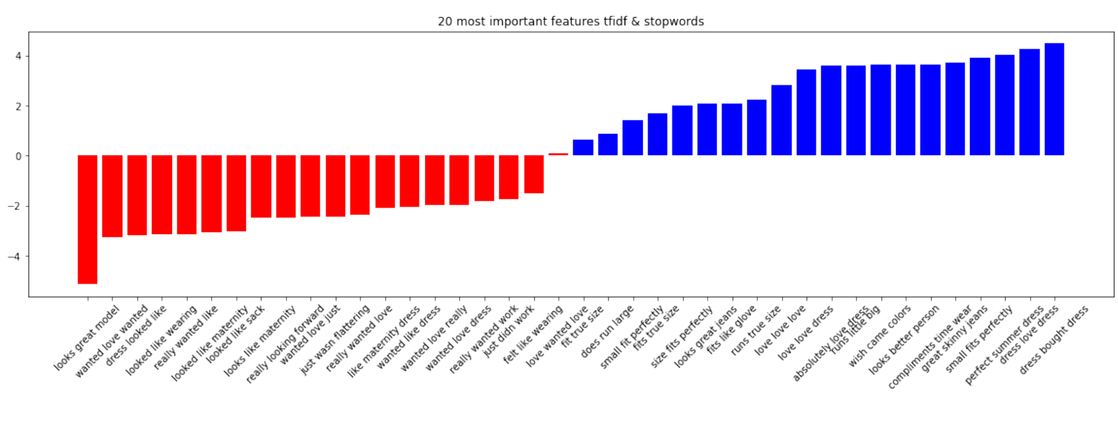
In [203]:
```
tfidf33_chr = TfidfVectorizer(stop_words='english', ngram_range=(3,3))
tfidf33_chr.fit(top20_names)
```

Out[203]:
```
TfidfVectorizer(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=(3, 3), norm='l2', 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)
```

In [204]:
```
X_train_tfidf33_chr_sw = tfidf33_chr.transform(train3_clean['text'])
y_train_tfidf33_chr_sw = train3_clean['Recommended']
X_test_tfidf33_chr_sw = tfidf33_chr.transform(test3_clean['text'])
y_test_tfidf33_chr_sw = test3_clean['Recommended']
feature_names_tfidf33_chr_sw = tfidf33_chr.get_feature_names()
logreg_tfidf33_chr_sw = LogisticRegressionCV(scoring = 'average_precision')
logreg_tfidf33_chr_sw.fit(X_train_tfidf33_chr_sw,y_train_tfidf33_chr_sw)
print('test score:',logreg_tfidf33_chr_sw.score(X_test_tfidf33_chr_sw,y_test_tfidf33_c

print("Test Avg Precision score: ", average_precision_score(logreg_tfidf33_chr_sw.pred
print("Test F1 score: ", f1_score(logreg_tfidf33_chr_sw.predict(X_test_tfidf33_chr_sw)
#print("Test ROC AUC score: ", roc_auc_score(logreg_tfidf33_chr_sw.predict(X_test_tfid


plot(logreg_tfidf33_chr_sw.coef_[0],feature_names_tfidf33_chr_sw,'tfidf & stopwords')
```

```
test score: 0.8255648280073274
Test Avg Precision score:  0.9923656786600952
```

```
Test F1 score:  0.9030213873486477
[0.62978991 0.8681424  1.43005756 1.70062776 1.98760741 2.07432436
 2.08773766 2.22089793 2.82117228 3.43250548 3.60999337 3.61651236
 3.62434087 3.64092872 3.65729999 3.72658233 3.91197097 4.02150967
 4.25037578 4.49548995]
['fit true size', 'does run large', 'small fit perfectly', 'fits true size', 'size fits perfectl
[-5.13277821 -3.26080294 -3.18066545 -3.13737648 -3.12430067 -3.05663908
 -3.0099685  -2.49133229 -2.4844754  -2.43253141 -2.42604465 -2.37218787
 -2.06802344 -2.06324777 -1.97851051 -1.96235476 -1.8126975  -1.743979
 -1.50967487  0.09343865]
['looks great model', 'wanted love wanted', 'dress looked like', 'looked like wearing', 'really
[-5.13277821 -3.26080294 -3.18066545 -3.13737648 -3.12430067 -3.05663908
 -3.0099685  -2.49133229 -2.4844754  -2.43253141 -2.42604465 -2.37218787
 -2.06802344 -2.06324777 -1.97851051 -1.96235476 -1.8126975  -1.743979
 -1.50967487  0.09343865  0.62978991  0.8681424   1.43005756  1.70062776
  1.98760741  2.07432436  2.08773766  2.22089793  2.82117228  3.43250548
  3.60999337  3.61651236  3.62434087  3.64092872  3.65729999  3.72658233
  3.91197097  4.02150967  4.25037578  4.49548995]
['looks great model' 'wanted love wanted' 'dress looked like'
 'looked like wearing' 'really wanted like' 'looked like maternity'
 'looked like sack' 'looks like maternity' 'really looking forward'
 'wanted love just' 'just wasn flattering' 'really wanted love'
 'like maternity dress' 'wanted like dress' 'wanted love really'
 'wanted love dress' 'really wanted work' 'just didn work'
 'felt like wearing' 'love wanted love' 'fit true size' 'does run large'
 'small fit perfectly' 'fits true size' 'size fits perfectly'
 'looks great jeans' 'fits like glove' 'runs true size' 'love love love'
 'love love dress' 'absolutely love dress' 'runs little big'
 'wish came colors' 'looks better person' 'compliments time wear'
 'great skinny jeans' 'small fits perfectly' 'perfect summer dress'
 'dress love dress' 'dress bought dress']
```




```
In [205]: X_train_tfidf33_chr_sw
```

```
Out[205]: <14762x40 sparse matrix of type '<class 'numpy.float64'>'
                with 1676 stored elements in Compressed Sparse Row format>
```

```
In [166]: X_test_tfidf33_chr_sw
```

```
Out[166]: <4913x1 sparse matrix of type '<class 'numpy.float64'>'
                with 0 stored elements in Compressed Sparse Row format>
```

```
In [ ]:
```

Now we train the tfidf with stop words and n_grams = (3,3) model on an svm with l1 and l2 penalties

```
In [ ]:
```

```
In [356]: from sklearn.svm import LinearSVC
          from sklearn.model_selection import GridSearchCV

          svcl1 = LinearSVC(penalty='l1', dual = False)
          svcl2 = LinearSVC(penalty='l2')
          parameters = {'C':[0.01,0.1,0.5,1,5, 10]}
          clf1 = GridSearchCV(svcl1, parameters)
          clf2 = GridSearchCV(svcl2, parameters)
```

```
In [ ]:
```

```
In [357]: tfidf33_sw = TfidfVectorizer(stop_words='english', ngram_range=(3,3))

          X_train_tfidf33_sw = tfidf33_sw.fit_transform(train3_clean['text'])
          y_train_tfidf33_sw = train3_clean['Recommended']
          X_test_tfidf33_sw = tfidf33_sw.transform(test3_clean['text'])
          y_test_tfidf33_sw = test3_clean['Recommended']
          feature_names_tfidf33_sw = tfidf33_sw.get_feature_names()


          #svm_tfidf33_sw = LogisticRegressionCV(scoring = 'average_precision')
          clf1.fit(X_train_tfidf33_sw,y_train_tfidf33_sw)
          clf2.fit(X_train_tfidf33_sw,y_train_tfidf33_sw)


          print('l1 test score:',clf1.score(X_test_tfidf33_sw,y_test_tfidf33_sw))

          print("l1 Test Avg Precision score: ", average_precision_score(clf1.predict(X_test_tfi
          print("l1 Test F1 score: ", f1_score(clf1.predict(X_test_tfidf33_sw),y_test_tfidf33_sw
          print("l1 Test ROC AUC score: ", roc_auc_score(clf1.predict(X_test_tfidf33_sw),y_test_



          print('l2 test score:',clf2.score(X_test_tfidf33_sw,y_test_tfidf33_sw))
```

```
print("l2 Test Avg Precision score: ", average_precision_score(clf2.predict(X_test_tfi
print("l2 Test F1 score: ", f1_score(clf2.predict(X_test_tfidf33_sw),y_test_tfidf33_sw
print("l2 Test ROC AUC score: ", roc_auc_score(clf2.predict(X_test_tfidf33_sw),y_test_
```

```
l1 test score: 0.8214480130647795
l1 Test Avg Precision score:  0.9739109331248541
l1 Test F1 score:  0.8994481912936848
l1 Test ROC AUC score:  0.691152803069538
l2 test score: 0.8255307566684812
l2 Test Avg Precision score:  0.9875369314309514
l2 Test F1 score:  0.9026871109761652
l2 Test ROC AUC score:  0.7480933779761906
```

We see that using svc with l1 and l2 penalties leads to a significant drop in performance.

In [ ]:

We could explore: -Length of text

```
-Presence of Emojis

-Number of out-of-vocabularly words

-Presence / frequency of ALL CAPS

-Presence of punctuation like !

-Sentiment words (fantastic, great, amazing vs disappointing, bad, etc.)
```

In [ ]: