DAT340 / DIT867 Applied Machine Learning

Programming assignment: Text classification

Amr Mohamed

Exchange student from CY Tech - France to Gothenburg University CSE Department

Anh Thu DOAN

Exchange student from CY Tech - France to Gothenburg University CSE Department

In [1]:
```
import pandas as pd
import numpy as np

import stanza
from scipy.stats import expon


# the actual classification algorithm
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline

# for splitting the dataset into training and test sets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# for evaluating the quality of the classifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

from sklearn.inspection import permutation_importance

import warnings
warnings.filterwarnings('ignore')

import seaborn as sns
import matplotlib.pyplot as plt

%config InlineBackend.figure_format = 'svg'
plt.style.use('bmh')
plt.rcParams['image.cmap'] = 'Paired_r'
```

In [2]:
```
train = pd.read_csv('PA3_train.tsv', sep='\t', names = ['annotation','review'], h
test = pd.read_csv('PA3_test_clean.tsv', sep='\t', names = ['annotation','review'
```

Training Data Cleaning

In [3]:
```
train.head()
```

Out[3]:

	annotation	review
0	0/0	Ordered my food the hole meal looked dead. pla...
1	1/1	We stopped her whilst walking in the Haga area...
2	0/0	Bad experience, On 23/03/19 Myself and my part...
3	0/0	Extremely underwhelming experience here last n...
4	0/0	Waited 30 minutes to get a table…that was ok. ...

In [4]:
```
train.shape
```

Out[4]:
```
(7018, 2)
```

In [5]:
```
train.review[1]
```

Out[5]:
```
'We stopped her whilst walking in the Haga area. The Cafe is well recommended.
Good service and we enjoyed our teas and a Cinamon Roll. The latter was large b
ut so good that between us we finished it! Recommended stop off.'
```

In [6]:
```
train[['annotation1','annotation2']]=train.annotation.str.split('/', expand=True)
```

In [7]:
```
train.head()
```

Out[7]:

	annotation	review	annotation1	annotation2
0	0/0	Ordered my food the hole meal looked dead. pla...	0	0
1	1/1	We stopped her whilst walking in the Haga area...	1	1
2	0/0	Bad experience, On 23/03/19 Myself and my part...	0	0
3	0/0	Extremely underwhelming experience here last n...	0	0
4	0/0	Waited 30 minutes to get a table…that was ok. ...	0	0

In [8]:
```
train.annotation.unique()
```

Out[8]:
```
array(['0/0', '1/1', '1/0', '-1/0', '-1/1', '0/1', '2/1', '2/0', '1/',
       '9/1'], dtype=object)
```

In [9]:
```
train.annotation1.unique()
```

Out[9]:
```
array(['0', '1', '-1', '2', '9'], dtype=object)
```

In [10]:
```
train.annotation2.unique()
```

Out[10]:
```
array(['0', '1', ''], dtype=object)
```

In [11]:
```
total_disagreement = round(len(train[~train.annotation.isin(['1/1', '0/0'])].annc
print("The percentage of disagreement between the annotators is: "+ str(total_dis
```

```
The percentage of disagreement between the annotators is: 0.056
```

In [12]:
```
true_disagreement = round(len(train[train.annotation.isin(['1/0', '0/1'])].annota
print("The percentage of real disagreement between the annotators is (0/1 or 1/0)
```

```
The percentage of real disagreement between the annotators is (0/1 or 1/0): 0.0
38
```

In [13]:
```
false_disagreement = round(len(train[~train.annotation.isin(['0/0', '1/1', '1/0',
print("The percentage of misannotated reviews led to\na difference in the annotat
```

```
The percentage of misannotated reviews led to
a difference in the annotations between the annotators is: 0.019
```

Sentiment analysis on rows with an annotation disagreement

In [14]:
```
# Stanza library for sentiment analysis
nlp = stanza.Pipeline(lang='en', processors='tokenize, sentiment')
```

```
2022-10-31 23:23:06 INFO: Checking for updates to resources.json in case models
have been updated.  Note: this behavior can be turned off with download_method=
None or download_method=DownloadMethod.REUSE_RESOURCES

Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/mai
n/resources_1.4.1.json: (https://raw.githubusercontent.com/stanfordnlp/stanza-r
esources/main/resources_1.4.1.json:)   0%|    …

2022-10-31 23:23:09 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
| sentiment | sstplus  |
========================

2022-10-31 23:23:09 INFO: Use device: cpu
2022-10-31 23:23:09 INFO: Loading: tokenize
2022-10-31 23:23:10 INFO: Loading: sentiment
2022-10-31 23:23:11 INFO: Done loading processors!
```

In [15]:
```
# collecting the sentiment scores for instances with disagreement
sent_scores = []
cpt=0
for index, row in train.iterrows():
    if row.annotation in ['1/0', '-1/0', '-1/1', '0/1', '2/1', '2/0', '1/','9/1']:
        doc = nlp(row.review)
        for sentence in doc.sentences:
            temp = []
            temp.append(int(sentence.sentiment)-1)
        sent_scores.append(sum(temp)/len(temp))
        print(cpt, end='\r')
    else:
        sent_scores.append(3)
    cpt+=1
```

```
7016
```

In [16]:
```
train['sent_ana']=sent_scores
train.head(10)
```

Out[16]:

	annotation	review	annotation1	annotation2	sent_ana
0	0/0	Ordered my food the hole meal looked dead. pla...	0	0	3.0
1	1/1	We stopped her whilst walking in the Haga area...	1	1	3.0
2	0/0	Bad experience, On 23/03/19 Myself and my part...	0	0	3.0
3	0/0	Extremely underwhelming experience here last n...	0	0	3.0
4	0/0	Waited 30 minutes to get a table…that was ok. ...	0	0	3.0
5	0/0	A mediocre burger, not tasteful	0	0	3.0
6	1/0	As a Kiwi guy constantly on the hunt for decen...	1	0	-1.0
7	0/0	If you go in here, don't go in before mystery ...	0	0	3.0
8	0/0	It's in a great location. That's it. Their pri...	0	0	3.0
9	1/1	About 200 people queuing in a small street at ...	1	1	3.0

In [17]:
```
train.shape
```

Out[17]: (7018, 5)

In [18]:
```
train = train[train.sent_ana!=0]
```

In [19]:
```
train.shape
```

Out[19]: (6935, 5)

In [20]:
```
train.sent_ana = train.sent_ana.replace(-1,0)
```

In [21]:
```
train.head(10)
```

Out[21]:

	annotation	review	annotation1	annotation2	sent_ana
0	0/0	Ordered my food the hole meal looked dead. pla...	0	0	3.0
1	1/1	We stopped her whilst walking in the Haga area...	1	1	3.0
2	0/0	Bad experience, On 23/03/19 Myself and my part...	0	0	3.0
3	0/0	Extremely underwhelming experience here last n...	0	0	3.0
4	0/0	Waited 30 minutes to get a table…that was ok. ...	0	0	3.0
5	0/0	A mediocre burger, not tasteful	0	0	3.0
6	1/0	As a Kiwi guy constantly on the hunt for decen...	1	0	0.0
7	0/0	If you go in here, don't go in before mystery ...	0	0	3.0
8	0/0	It's in a great location. That's it. Their pri...	0	0	3.0
9	1/1	About 200 people queuing in a small street at ...	1	1	3.0

In [22]:
```
# replacing the correctly annotated instances by their annotations after the sent
train.loc[train['sent_ana'] == 3, 'sent_ana'] = train['annotation2']
```

In [23]:
```
train.sent_ana = train.sent_ana.astype(int)
train.head(10)
```

Out[23]:

	annotation	review	annotation1	annotation2	sent_ana
0	0/0	Ordered my food the hole meal looked dead. pla...	0	0	0
1	1/1	We stopped her whilst walking in the Haga area...	1	1	1
2	0/0	Bad experience, On 23/03/19 Myself and my part...	0	0	0
3	0/0	Extremely underwhelming experience here last n...	0	0	0
4	0/0	Waited 30 minutes to get a table…that was ok. ...	0	0	0
5	0/0	A mediocre burger, not tasteful	0	0	0
6	1/0	As a Kiwi guy constantly on the hunt for decen...	1	0	0
7	0/0	If you go in here, don't go in before mystery ...	0	0	0
8	0/0	It's in a great location. That's it. Their pri...	0	0	0
9	1/1	About 200 people queuing in a small street at ...	1	1	1

In [24]:
```
train_df=train[['review', 'sent_ana']]
train_df=train_df.rename(columns={"sent_ana": "annotation"})
train_df.annotation=pd.to_numeric(train_df.annotation)
```

In [25]:
```
train_df.head()
```

Out[25]:

	review	annotation
0	Ordered my food the hole meal looked dead. pla...	0
1	We stopped her whilst walking in the Haga area...	1
2	Bad experience, On 23/03/19 Myself and my part...	0
3	Extremely underwhelming experience here last n...	0
4	Waited 30 minutes to get a table…that was ok. ...	0

In [26]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6935 entries, 0 to 7017
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   review      6935 non-null   object
 1   annotation  6935 non-null   int32 
dtypes: int32(1), object(1)
memory usage: 393.5+ KB
```

In [27]:
```
print('Number of positive reviews in the training set',len(train_df[train_df.annc
```

```
Number of positive reviews in the training set 3604
```

In [28]:
```
print('Number of negative reviews in the training set',len(train_df[train_df.annc
```

```
Number of negative reviews in the training set 3331
```

Test data cleaning

In [29]:
```
test.head()
```

Out[29]:

	annotation	review
0	0	Over all I felt a bit disappointing with abov...
1	1	A wonderful experience!
2	1	Always very delicious dishes and attentive ser...
3	1	Amazing as always
4	1	Amazing food, the aubergine mess and the Tunis...

In [30]:
```
test.shape
```

Out[30]:
```
(1751, 2)
```

In [31]:
```
test.annotation.unique()
```

Out[31]:
```
array([0, 1], dtype=int64)
```

In [32]:
```
test[test.annotation== 1 ].annotation.count()
```

Out[32]:
```
886
```

In [33]:
```
test[test.annotation== 0 ].annotation.count()
```

Out[33]:
```
865
```

In [34]:
```
# cleaning the text by converting all words in all the reviews to lowercase lette
def clean_text(text):
    # filter to allow only alphabets
#     text = re.sub(r'[^a-zA-Z\']', ' ', text)

    # remove Unicode characters
#     text = re.sub(r'[^\x00-\x7F]+', '', text)

    # convert to lowercase to maintain consistency
    text = text.lower()

    return text

train_df['review'] = train_df.review.apply(clean_text)
test['review'] = test.review.apply(clean_text)
```

In [35]:
```
# defining the training and testing sets
X_train = (train_df.review)
Y_train=train_df.annotation
X_test = (test.review)
Y_test=test.annotation
```

ML models

Dummy Classifier

In [36]:
```
#Baseline classifier with most-frequent strategy
dummy = DummyClassifier(strategy='most_frequent',random_state=0)
pipeline = make_pipeline( TfidfVectorizer(), dummy)
pipeline.fit(X_train, Y_train)
print('The accuracy of Dummy Classifier:',accuracy_score(Y_test, pipeline.predict
```

```
The accuracy of Dummy Classifier: 0.5059965733866362
```

Linear SVC

In [37]:
```
LSVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer()),
    ('linearsvc',LinearSVC(random_state=0))])
LSVCpipeline.fit(X_train, Y_train)
Yguess = LSVCpipeline.predict(X_test)
print('The accuracy of Linear SVC with default parameters and random state equal
```

```
The accuracy of Linear SVC with default parameters and random state equal to ze
ro: 0.9640205596801827
```

Support Vector Classification (SVC)

In [38]:
```
SVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer()),
    ('svc',SVC(random_state=0))])
SVCpipeline.fit(X_train, Y_train)
print('The accuracy of SVC with default parameters and random state equal to zero
```

```
The accuracy of SVC with default parameters and random state equal to zero: 0.9
680182752712736
```

Logistic Regression

In [39]:
```
logRpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer()),
    ('logr',LogisticRegression(random_state=0))])
logRpipeline.fit(X_train, Y_train)
print('The accuracy of Logistic regession with default parameters and random stat
```

```
The accuracy of Logistic regession with default parameters and random state equ
al to zero: 0.9605939463163906
```

Gradient Boosting Classifier

In [40]:
```
pipeline = make_pipeline( TfidfVectorizer(), GradientBoostingClassifier(random_st
pipeline.fit(X_train, Y_train)
print('The accuracy of Gradient Boosting Classifier with default parameters and r
```

```
The accuracy of Gradient Boosting Classifier with default parameters and random
state equal to zero: 0.9143346659051971
```

Random Forest Classifier

In [41]:
```
pipeline = make_pipeline( TfidfVectorizer(), RandomForestClassifier(random_state=
pipeline.fit(X_train, Y_train)
print('The accuracy of Random Forest Classifier with default parametersand random
```

```
The accuracy of Random Forest Classifier with default parametersand random stat
e equal to zero: 0.9354654483152485
```

Multinomial Naive Baye

In [42]:
```
pipeline = make_pipeline( TfidfVectorizer(), MultinomialNB())
pipeline.fit(X_train, Y_train)
print('The accuracy of Multinomial Naive Baye with default parameters:',accuracy_
```

```
The accuracy of Multinomial Naive Baye with default parameters: 0.9486007995431
183
```

Multi-layer Perceptron classifier

In [43]:
```
NN = MLPClassifier(early_stopping=True,n_iter_no_change=10)
pipeline = make_pipeline( TfidfVectorizer(),NN)
pipeline.fit(X_train, Y_train)
print('The accuracy of Multi-layer Perceptron classifier:',accuracy_score(Y_test,
```

```
The accuracy of Multi-layer Perceptron classifier: 0.9594517418617933
```

In [44]:
```
NN = MLPClassifier(alpha = 0.1, max_iter=400,early_stopping=True,n_iter_no_change
pipeline = make_pipeline( TfidfVectorizer(),NN)
pipeline.fit(X_train, Y_train)
print('The accuracy of Multi-layer Perceptron classifier:',accuracy_score(Y_test,
```

```
The accuracy of Multi-layer Perceptron classifier: 0.9640205596801827
```

Highest scoring models Tuning and evaluation (SVC and Linear SVC)

Hyperparameter Tuning Linear SVC

In [45]:
```
#hyperparamaeter tuning
LSVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer()),
    ('linearsvc',LinearSVC(random_state=0))])

#construct the paramter grid
param_grid = {'tfidf__max_df': [0.1,0.275,0.3545454545454545,0.3,0.5,0.75,1],
              'linearsvc__loss': ['hinge','squared_hinge'],
               'linearsvc__C': [0.01,0.1,1.0],
              'linearsvc__penalty': ['l1','l2'],
              'linearsvc__random_state': [0]}

LSVCgrid = GridSearchCV(LSVCpipeline, param_grid,n_jobs=-1)

# fitting the model for grid search
LSVCgrid.fit(X_train, Y_train)
```

Out[45]:
```
GridSearchCV(estimator=Pipeline(steps=[('tfidf', TfidfVectorizer()),
                                       ('linearsvc',
                                        LinearSVC(random_state=0))]),
             n_jobs=-1,
             param_grid={'linearsvc__C': [0.01, 0.1, 1.0],
                         'linearsvc__loss': ['hinge', 'squared_hinge'],
                         'linearsvc__penalty': ['l1', 'l2'],
                         'linearsvc__random_state': [0],
                         'tfidf__max_df': [0.1, 0.275, 0.3545454545454545, 0.3,
                                           0.5, 0.75, 1]})
```

In [46]:
```
LSVCgrid.best_params_
```

Out[46]:
```
{'linearsvc__C': 1.0,
 'linearsvc__loss': 'squared_hinge',
 'linearsvc__penalty': 'l2',
 'linearsvc__random_state': 0,
 'tfidf__max_df': 0.75}
```

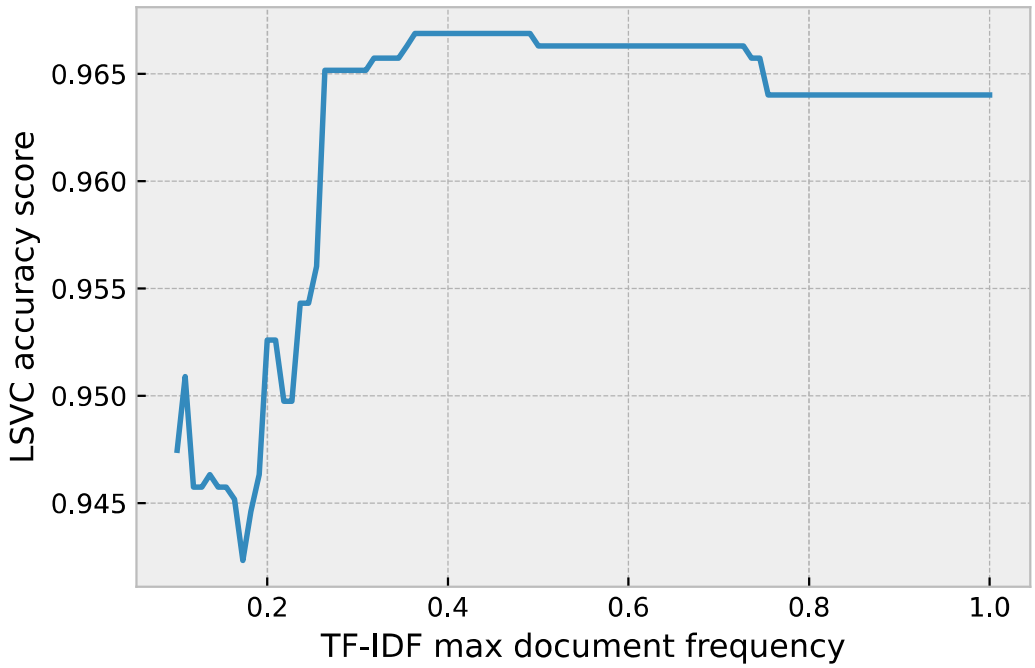
In [47]:
```
LSVCgrid.best_score_
```

Out[47]:
```
0.9603460706560923
```

In [48]:
```
print('The accuracy score of LSVC after tuning by grid search:',accuracy_score(Y_
```

```
The accuracy score of LSVC after tuning by grid search: 0.9640205596801827
```

In [49]:
```
#plotting the max_df tfidf parameter values over the accuracies of the linearsvc

max_dfList=[]
for i in list(np.linspace(0.1,1,100)):
    LSVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer(max_df=i)),
    ('svc',LinearSVC(random_state=0))])
    LSVCpipeline.fit(X_train, Y_train)
    max_dfList.append(accuracy_score(Y_test, LSVCpipeline.predict(X_test)))

plt.plot(list(np.linspace(0.1,1,100)),max_dfList)
plt.xlabel('TF-IDF max document frequency')
plt.ylabel('LSVC accuracy score');
```



In [50]:
```
LSVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer(max_df=0.3545454545454545)),
    ('linearsvc',LinearSVC(random_state=0))])
LSVCpipeline.fit(X_train, Y_train)
LSVC_Yguess = LSVCpipeline.predict(X_test)
print('The accuracy of LSVC with the best value for max_df:',accuracy_score(Y_tes
```

```
The accuracy of LSVC with the best value for max_df: 0.9663049685893775
```

Hyperparameter Tuning SVC

In [51]:
```
#hyperparamaeter tuning
param_grid = {'tfidf__max_df': [0.2,0.225,0.275,0.3],
              'svc__kernel': ['poly', 'rbf', 'sigmoid'],
              'svc__C': [0.3,0.5,0.7,1]}
SVCgs = GridSearchCV(SVCpipeline, param_grid, n_jobs=-1)
SVCgs.fit(X_train, Y_train);
```

In [52]:
```
SVCgs.best_params_
```

Out[52]:
```
{'svc__C': 1, 'svc__kernel': 'rbf', 'tfidf__max_df': 0.275}
```

In [53]:
```
SVCgs.best_score_
```

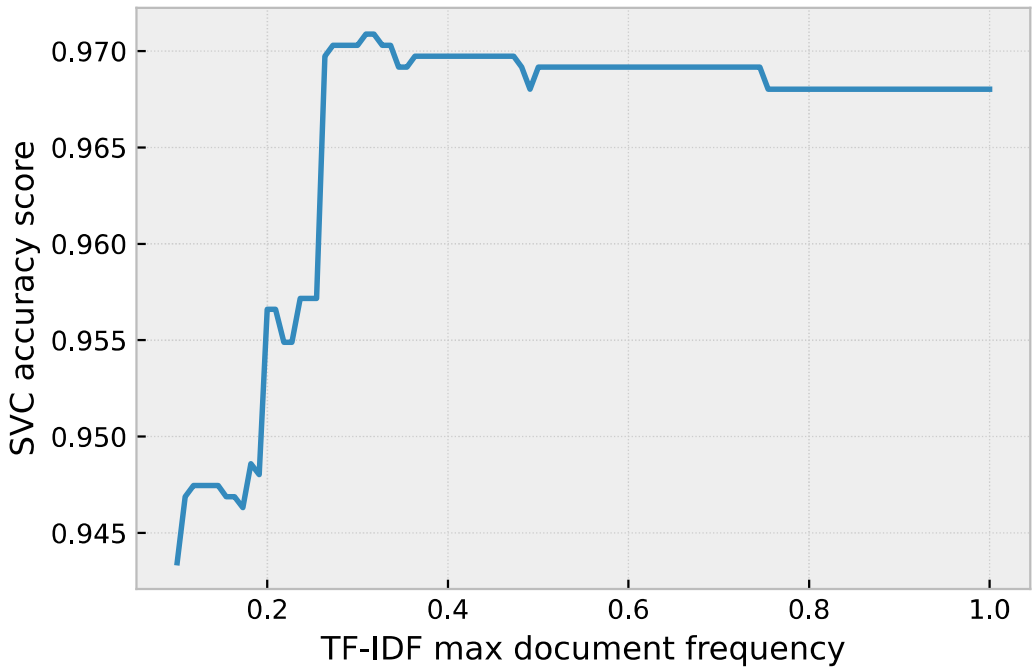
Out[53]:
```
0.9606344628695025
```

In [54]:
```
accuracy_score(Y_test, SVCgs.predict(X_test))
```

Out[54]:
```
0.9703026841804683
```

In [79]:
```
#plotting the max_df tfidf parameter values over the accuracies of the svc model
max_dfList=[]
for i in list(np.linspace(0.1,1,100)):
    SVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer(max_df=i)),
    ('svc',SVC(random_state=0))])
    SVCpipeline.fit(X_train, Y_train)
    max_dfList.append(accuracy_score(Y_test, SVCpipeline.predict(X_test)))
```

In [80]:
```
plt.plot(list(np.linspace(0.1,1,100)),max_dfList)
plt.xlabel('TF-IDF max document frequency')
plt.ylabel('SVC accuracy score');
```



In [81]:
```
dict(zip(max_dfList,list(np.linspace(0.1,0.5,40))))
```

Out[81]:
```
{0.9434608794974301: 0.1,
 0.9468874928612222: 0.1717948717948718,
 0.9474585950885208: 0.1512820512820513,
 0.9463163906339235: 0.18205128205128207,
 0.9486007995431183: 0.19230769230769232,
 0.9480296973158195: 0.20256410256410257,
 0.9565962307252999: 0.2230769230769231,
 0.9548829240434038: 0.24358974358974358,
 0.9571673329525985: 0.2743589743589744,
 0.9697315819531697: 0.5,
 0.9703026841804683: 0.3666666666666667,
 0.970873786407767: 0.34615384615384615,
 0.9691604797258709: 0.38717948717948714}
```

In [82]:
```
SVCpipeline = Pipeline( [(
    'tfidf',TfidfVectorizer(max_df=0.2641025641025641)),
    ('svc',SVC())])
SVCpipeline.fit(X_train, Y_train)
accuracy_score(Y_test, SVCpipeline.predict(X_test))
```

Out[82]:
```
0.9697315819531697
```

Linear SVC evaluation

In [83]:
```
#getting the confusion matrix
cf_matrix =confusion_matrix(Y_test, LSVC_Yguess)
```

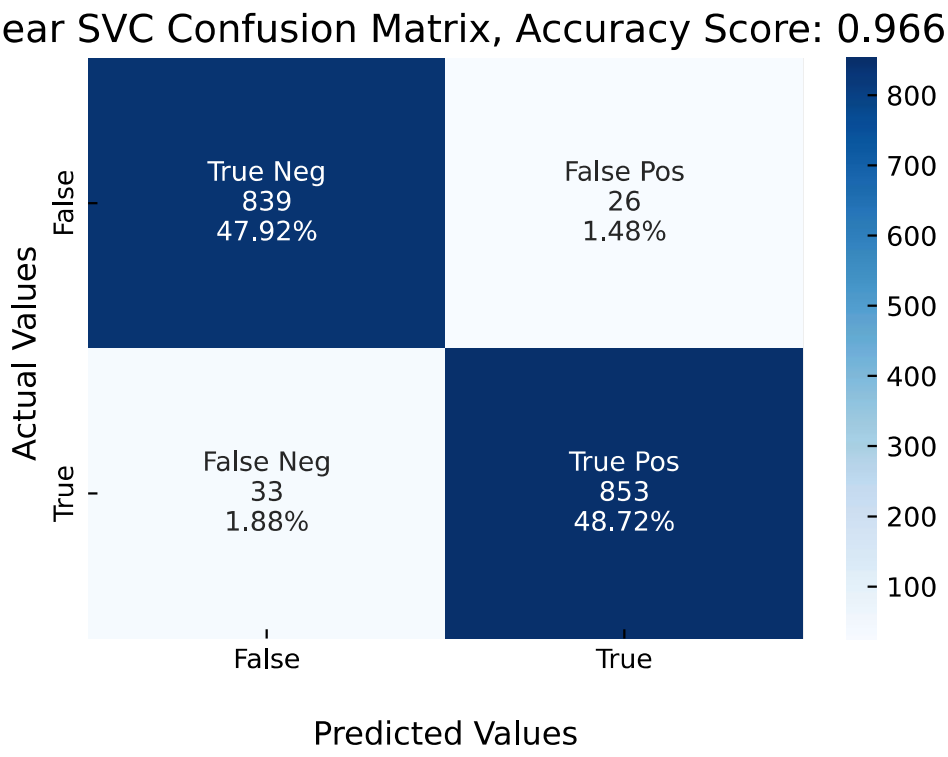
In [84]:
```
print(classification_report(Y_test, LSVC_Yguess))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       865
           1       0.97      0.96      0.97       886

    accuracy                           0.97      1751
   macro avg       0.97      0.97      0.97      1751
weighted avg       0.97      0.97      0.97      1751
```

In [85]:
```
cf_matrix
```

Out[85]:
```
array([[839,  26],
       [ 33, 853]], dtype=int64)
```

In [86]:
```
#Plotting confusion matrix for LSVC
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Linear SVC Confusion Matrix, Accuracy Score: %.3f' % accuracy_score
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



In [87]:
```
print("Precision score of LSVC:",precision_score(Y_test, LSVC_Yguess, pos_label=1
      "\nRecall score of LSVC:",recall_score(Y_test, LSVC_Yguess, pos_label=1))
```

```
Precision score of LSVC: 0.9704209328782708
Recall score of LSVC: 0.9627539503386005
```

SVC evaluation

In [88]:
```
cf_matrix =confusion_matrix(Y_test, SVCpipeline.predict(X_test))
```

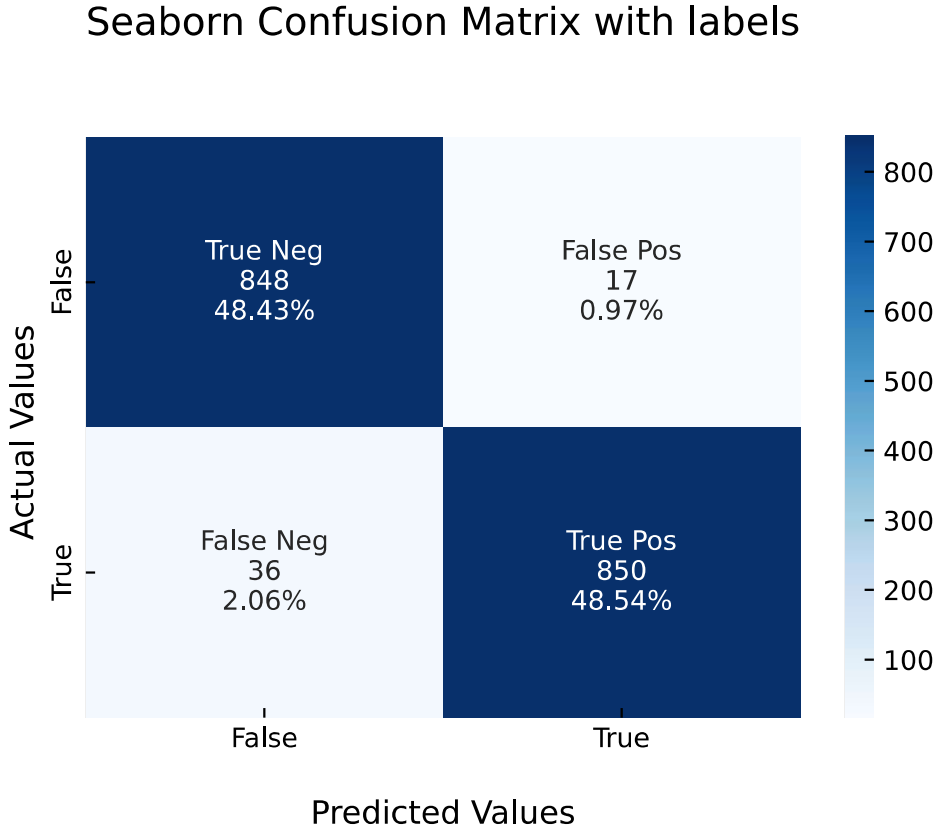
In [89]:
```
print(classification_report(Y_test, SVCpipeline.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       865
           1       0.98      0.96      0.97       886

    accuracy                           0.97      1751
   macro avg       0.97      0.97      0.97      1751
weighted avg       0.97      0.97      0.97      1751
```

In [90]:
```
cf_matrix
```

Out[90]:
```
array([[848,  17],
       [ 36, 850]], dtype=int64)
```

In [91]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



In [92]:
```
print("Precision score of SVC:",precision_score(Y_test, SVCpipeline.predict(X_tes
      "\nRecall score of SVC:",recall_score(Y_test, SVCpipeline.predict(X_test),
```

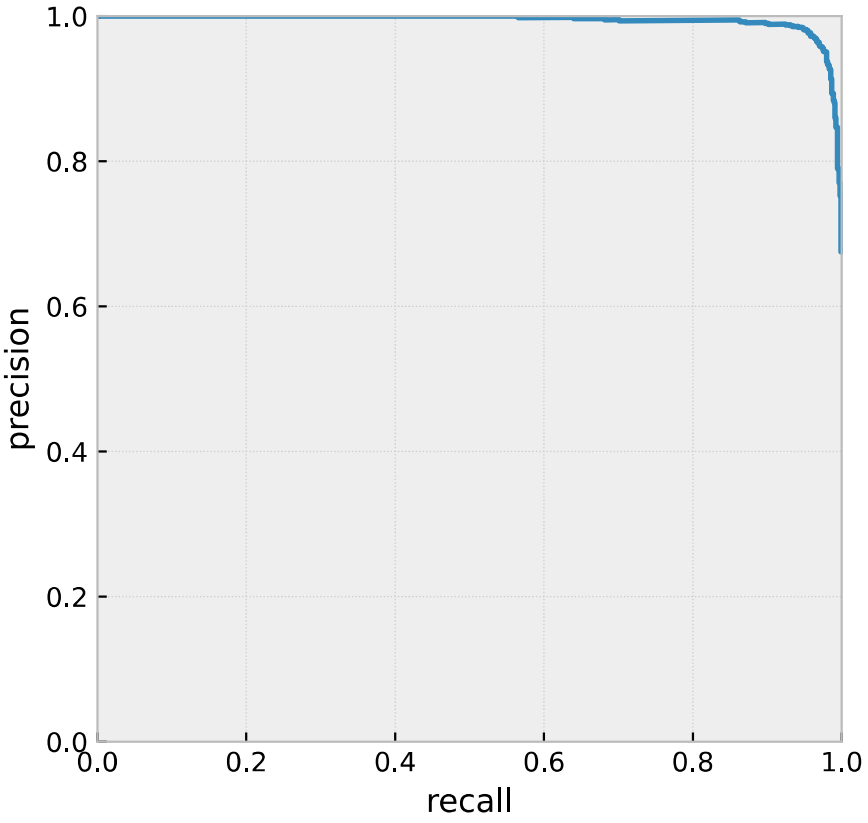
```
Precision score of SVC: 0.9803921568627451
Recall score of SVC: 0.9593679458239278
```

In [93]:
```
print(classification_report(Y_test, SVCpipeline.predict(X_test)))
```

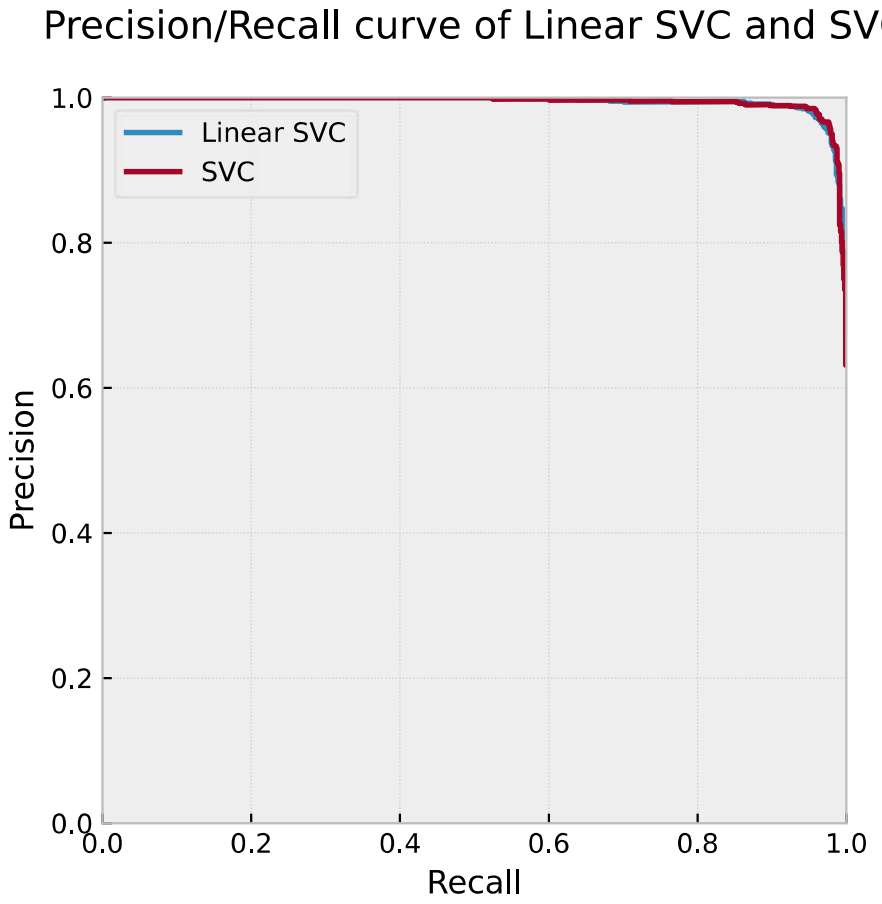
```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       865
           1       0.98      0.96      0.97       886

    accuracy                           0.97      1751
   macro avg       0.97      0.97      0.97      1751
weighted avg       0.97      0.97      0.97      1751
```

Precision/Recall curves comparison for Linear SVC and SVC

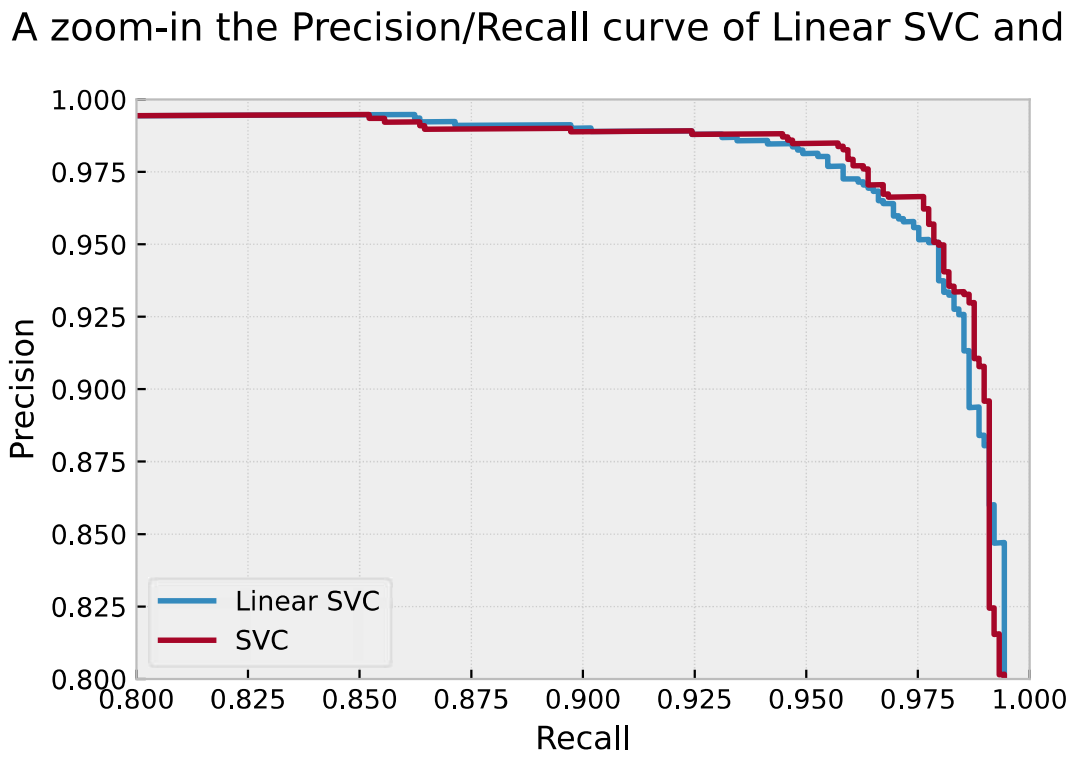
In [94]:
```
scores = LSVCpipeline.decision_function(X_test)

precs, recs, _ = precision_recall_curve(Y_test, scores, pos_label=1)

plt.figure(figsize=(5,5))
plt.plot(recs, precs)
plt.xlabel('recall')
plt.ylabel('precision')
plt.axis([0, 1, 0, 1]);
```



In [95]:
```
#Plotting the precision & recall curve for SVC & LSVC
pipelines = [LSVCpipeline,SVCpipeline]

APs = []
plt.figure(figsize=(5,5))
for pipeline in pipelines:
    scores = pipeline.decision_function(X_test)
    precisions, recalls, _ = precision_recall_curve(Y_test, scores, pos_label=1)

    plt.plot(recalls, precisions)
    plt.xlabel('Recall')
    plt.ylabel('Precision')

plt.legend(['Linear SVC','SVC']);
plt.title('Precision/Recall curve of Linear SVC and SVC\n' )
plt.axis([0, 1, 0, 1]);
```



In [96]:
```
#Zoom-in the PR of SVC and LSVC
for pipeline in pipelines:
    scores = pipeline.decision_function(X_test)
    precisions, recalls, _ = precision_recall_curve(Y_test, scores, pos_label=1)

    plt.plot(recalls, precisions)
    plt.xlabel('Recall')
    plt.ylabel('Precision')

# leg_labels = [ '{:3}: AP = {:.3f}'.format(n, AP) for n, AP in APs ]
plt.legend(['Linear SVC','SVC']);
plt.title('A zoom-in the Precision/Recall curve of Linear SVC and SVC\n' )
plt.axis([0.8, 1, 0.8, 1]);
```



LSVC Misclassified instances sample

In [97]:
```
tempdf= test
tempdf['LSVC_predicted'] = LSVC_Yguess
tempdf[tempdf.LSVC_predicted != tempdf.annotation].head().review.values
```

Out[97]:
```
array(['no reasonably priced set lunch menu.\xa0',
       'every indian restaurant on brick lane woo you at the door with offers o
f free beer or wine, a history of great reviews and celebrity endorsements. wha
t sets the monsoon aside is that once they sit you down they act like they do
n’t need your business. they were the rudest staff i have dealt with in a long
time. they kept storming around the restaurant floor, the head waiter twice lea
ning against my table (which i hate but can forgive), they kept taking away the
courses before i had finished, and by the timely came to dessert or coffee i fe
lt rushed out the door so just paid and left. on and food was poor, with the ex
ception of the bhaji. i will never eat there again. brilliant restaurants just
down the street that have given me excellent food and service every time.',
       "we ate here last night after a film, the food was great and prices reas
onable. better still, i accidentally ordered a biryani with chicken in it (i'm
a vegetarian) and they took it back and changed it for a veggie biryani without
charging me for it, even though it was totally my fault. that's classy. they al
so wouldn't take any service charge...would definitely come again!",
       "while visiting london and staying in brick lane, we checked into our ho
tel in the early hours of the morning. we saw that the monsoon was open so head
ed in for a meal. they had closed for the evening but cooked us a meal for us t
o take away . while waiting, we were able to have a drink. as we were in a hote
l room, we had no cutlery or bowls to use, so staff sorted us some plastic cutl
ery and two plastic take away pots for us to use as bowls. they even threw in a
bottle of cobra. they couldn't have been more helpful, in the early hours of th
e morning, to make sure we could eat, especially as they were probably about to
head home themselves...thankyou.",
       'who can resist the huge kanelbulle or cinnamon rolls stacked up by the
window? we went in, and it was more about the size and the fun than the taste.
the cinnamon roll was dense, a bit dry, and too sweet for my taste. we barely a
te a quarter of it.'],
      dtype=object)
```

SVC Misclassified instances sample

In [98]:
```
tempdf['SVC_predicted'] = SVCpipeline.predict(X_test)
tempdf[tempdf.SVC_predicted != tempdf.annotation].head().review.values
```

Out[98]:
```
array(['no reasonably priced set lunch menu.\xa0',
       "we ate here last night after a film, the food was great and prices reas
onable. better still, i accidentally ordered a biryani with chicken in it (i'm
a vegetarian) and they took it back and changed it for a veggie biryani without
charging me for it, even though it was totally my fault. that's classy. they al
so wouldn't take any service charge...would definitely come again!",
       "while visiting london and staying in brick lane, we checked into our ho
tel in the early hours of the morning. we saw that the monsoon was open so head
ed in for a meal. they had closed for the evening but cooked us a meal for us t
o take away . while waiting, we were able to have a drink. as we were in a hote
l room, we had no cutlery or bowls to use, so staff sorted us some plastic cutl
ery and two plastic take away pots for us to use as bowls. they even threw in a
bottle of cobra. they couldn't have been more helpful, in the early hours of th
e morning, to make sure we could eat, especially as they were probably about to
head home themselves...thankyou.",
       'great service. computerised order screen make it really easy. order on
the screen and just sit down..and they bring you your food to the table...yes i
did say they bring you your food. this was faster than going to any drive in th
at i have visited before.',
       'my favourite place in hong kong for peking duck. i have never been disa
ppointed in 15 years. fabulous dishes such as beggar chicken or dim sum (xiaolo
ngbao) are a must. decoration is chinese chic. there is sufficient space betwee
n the tables so that it does not feel overcrowded or too noisy, even with large
group tables.the service is polite and efficient. can’t wait to visit again.'],
      dtype=object)
```

Features importance

LSVC

In [99]:
```
feature_names = LSVCpipeline.named_steps["tfidf"].get_feature_names()
```

In [100]:
```
coefs = LSVCpipeline.named_steps["linearsvc"].coef_.flatten()
```

In [101]:
```
zipped = zip(feature_names, coefs)
df = pd.DataFrame(zipped, columns=["feature", "value"])
# Sort the features by the absolute value of their coefficient
df["abs_value"] = df["value"].apply(lambda x: abs(x))
df["colors"] = df["value"].apply(lambda x: "green" if x > 0 else "red")
df = df.sort_values("abs_value", ascending=False)
```

In [103]:
```
sns.set_style( {"grid.color": ".8", "grid.linestyle": ":"})

fig, ax = plt.subplots(1, 1, figsize=(5, 6))
sns.barplot(y="feature",
            x="value",
            data=df.head(30),
           palette=df.head(30)["colors"])
ax.set_title("Top 30 words (features) contributing to \nthe Linear SVC classifica
ax.set_ylabel("Word", fontsize=12)
ax.set_xlabel("Coefficient", fontsize=12)
plt.grid()
```