I combine my report with code together, since they overlapped a lot.

In [17]:
```
# Logistic Regression Classifier from PA2
def train_classifier(X, y):
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import GridSearchCV
    cls = LogisticRegression(random_state=0,  max_iter=10000)
    parameters = {'solver' :  [ 'lbfgs', 'liblinear', 'sag', 'saga'], 'C':[0.0
01, 0.01, 0.1,1]}
    clf = GridSearchCV(cls,parameters,cv=3)
    clf.fit(X, y)
    return clf
# The coefficient generated by SVM is needed for producing word clouds
def svc(X,y):
    from sklearn.svm import SVC
    from sklearn.svm import LinearSVC
    clf = LinearSVC()
    #clf = SVC(kernel = 'linear',probability=True)
    clf.fit(X, y)
    return clf
def evaluate(X, yt, cls, name='data'):
    """Evaluated a classifier on the given labeled data using accuracy."""
    from sklearn import metrics
    yp = cls.predict(X)
    acc = metrics.accuracy_score(yt, yp)
    print("  Accuracy on %s  is: %s" % (name, acc))
    return acc
```


In [18]:
```
#!/bin/python
def read_files(tarfname,m,n):
    import tarfile
    from nltk.corpus import stopwords
    stop = stopwords.words('english')
    tar = tarfile.open(tarfname, "r:gz")
    trainname = "train.tsv"
    devname = "dev.tsv"
    for member in tar.getmembers():
        if 'train.tsv' in member.name:
            trainname = member.name
        elif 'dev.tsv' in member.name:
            devname = member.name


    class Data: pass
    sentiment = Data()
    print("-- train data")
    sentiment.train_data, sentiment.train_labels = read_tsv(tar,trainname)
    print(len(sentiment.train_data))

    print("-- dev data")
    sentiment.dev_data, sentiment.dev_labels = read_tsv(tar, devname)
    print(len(sentiment.dev_data))
    print("-- transforming data and labels")
    from sklearn.feature_extraction.text import TfidfVectorizer
    sentiment.tf_vect = TfidfVectorizer(min_df=m, max_df=n)
    sentiment.trainX = sentiment.tf_vect.fit_transform(sentiment.train_data)
    sentiment.devX = sentiment.tf_vect.transform(sentiment.dev_data)
    from sklearn import preprocessing
    sentiment.le = preprocessing.LabelEncoder()
    sentiment.le.fit(sentiment.train_labels)
    sentiment.target_labels = sentiment.le.classes_
    sentiment.trainy = sentiment.le.transform(sentiment.train_labels)
    sentiment.devy = sentiment.le.transform(sentiment.dev_labels)
    tar.close()
    return sentiment

def read_tsv(tar, fname):
    import nltk
    nltk.download('punkt')
    member = tar.getmember(fname)
    print(member.name)
    tf = tar.extractfile(member)
    data = []
    labels = []
    for line in tf:
        line = line.decode("utf-8")
        (label,text) = line.strip().split("\t")
        labels.append(label)
        data.append(' '.join([word.lower() for word in nltk.word_tokenize(text
)
                              if  word.isalpha()]))
    return data, labels

def write_pred_kaggle_file(unlabeled, cls, outfname, sentiment):
```

```
    yp = cls.predict(unlabeled.X)
    labels = sentiment.le.inverse_transform(yp)
    f = open(outfname, 'w')
    f.write("ID,LABEL\n")
    for i in range(len(unlabeled.data)):
        f.write(str(i+1))
        f.write(",")
        f.write(labels[i])
        f.write("\n")
    f.close()
```


In [19]:

```
# Build model for the first part
def model(tarfname):
    sentiment = read_files(tarfname,1,0.5)
    print("\nTraining classifier")
    clf = train_classifier(sentiment.trainX, sentiment.trainy)
    #clf = svc(sentiment.trainX, sentiment.trainy)
    print("\nEvaluating")
    print("train acc",evaluate(sentiment.trainX, sentiment.trainy, clf, 'trai
n'))
    print("dev acc",evaluate(sentiment.devX, sentiment.devy, clf, 'dev'))
    return clf,sentiment

# This function is designed to show word cloud and explanation of sentence for
any input sentences.
def inputtext(clf,doc,num_features,arr,sentiment):
    import matplotlib.pyplot as plt
    from wordcloud import WordCloud
    import lime
    from lime import lime_text
    from sklearn.pipeline import make_pipeline
    c = make_pipeline( sentiment.tf_vect, clf)
    from lime.lime_text import LimeTextExplainer
    explainer = LimeTextExplainer(class_names=arr)
    cl = svc(sentiment.trainX, sentiment.trainy)
    coefficients=cl.coef_[0]
    for i in range(0,len(doc)):
        dd = sentiment.tf_vect.transform(doc)[i].todok()
        import operator
        sorted_x = sorted(dd.items(), key=operator.itemgetter(1))
        top_k_words = {}
        bottom_k_words = {}
        keys = [sorted_x[r][0][1] for r in range(0,len(sorted_x))]
        for j in range(0,len(keys)):
            if coefficients[keys[j]]>0:
                top_k_words[sentiment.tf_vect.get_feature_names()[keys[j]]] =
coefficients[keys[j]]
            else:
                bottom_k_words[sentiment.tf_vect.get_feature_names()[keys[j]]]
= -1*coefficients[keys[j]]
        print('The tf value for positive words are ',top_k_words)
        print('The absolute tf value for negative words are ',bottom_k_words)
        wordcloud = WordCloud(width=1000, height=500)
        wordcloud.generate_from_frequencies(frequencies=top_k_words)
        plt.figure( figsize=(10,7), facecolor='k')
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.show()
        wordcloud = WordCloud(width=1000, height=500)
        wordcloud.generate_from_frequencies(frequencies=bottom_k_words)
        plt.figure( figsize=(10,7), facecolor='k')
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.show()
        exp = explainer.explain_instance(doc[i], c.predict_proba, num_features
=num_features)
        pre = c.predict_proba([doc[i]])[0][0]
```

```
        l = ''
        if pre>0.5:
            l = arr[0]
        else:
            l = arr[1]
        print('We predited: ',l)
        if  pre>0.7 or  pre<0.3:
            print ('We are sure that the prediction is right')
        elif  pre>0.55 and pre<0.7 or pre<0.45 and pre>0.3:
            print ('We are not strongly confident about the prediction')
        elif pre<0.55 and pre>0.45:
            print ('We are unsure about the prediction')
        %matplotlib inline
        exp.show_in_notebook(text=True)

# Produce the cloud of the most important k words at the two end, i.e. words w
ith the top k smallest
# coefficient usually have large absolute value, so they also have a high weig
hts on determining one
# of catogories in the binary classification.
def topk(k,sentiment):
    import numpy as np
    cl = svc(sentiment.trainX, sentiment.trainy)
    coefficients=cl.coef_[0]
    top_k =np.argsort(coefficients)[-k:]
    top_k_words = {}
    print('-'*k)
    print('Top k=%d' %k)
    print('-'*k)
    for i in top_k:
        top_k_words[sentiment.tf_vect.get_feature_names()[i]] = coefficients[i
]
    print(top_k_words.keys())
    import matplotlib.pyplot as plt
    from wordcloud import WordCloud
    wordcloud = WordCloud(width=1500, height=1000)
    wordcloud.generate_from_frequencies(frequencies=top_k_words)
    plt.figure( figsize=(20,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
    #top_k = np.argpartition(coefficients, -k)[-k:]
    print('-'*k)
    print('Bottom k=%d' %k)
    print('-'*k)
    bottom_k =np.argsort(coefficients)[:k]
    bottom_k_words = {}
    for i in bottom_k:
        bottom_k_words[sentiment.tf_vect.get_feature_names()[i]] = -1*coeffici
ents[i]
    print(bottom_k_words.keys())
    wordcloud = WordCloud(width=1500, height=1000)
    wordcloud.generate_from_frequencies(frequencies=bottom_k_words)
    plt.figure( figsize=(20,10), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
```



```
# Users can remove any words and check the difference of prediction before and
after
# and thus get a general idea about the effect of these words
def PredictionDecomposition(clf,sentiment,idx,index,features,arr):
    tmp = sentiment.devX[idx].copy()
    for f in features:
        tmp[0,sentiment.tf_vect.vocabulary_[f]] = 0
    diff = clf.predict_proba(tmp)[0,index] - clf.predict_proba(sentiment.devX[
idx])[0,index]
    if diff > 0.2 or diff < -0.2:
        print(features," have a large effect on the decision of this sentence"
)
    else:
        print(features," have a small effect on the decision of this sentence"
)
    print('The difference between Original prediction for',clf.predict_proba(s
entiment.devX[idx])[0,index],' and Prediction removing some features: ',
          clf.predict_proba(tmp)[0,index], ' is ',diff)

# Print out explanation and our confidence for sentences from file
def PredictionExplaination(clf,sentiment,idx,index,num_features,arr):
    import lime
    from lime import lime_text
    from sklearn.pipeline import make_pipeline
    c = make_pipeline(sentiment.tf_vect, clf)
    from lime.lime_text import LimeTextExplainer
    explainer = LimeTextExplainer(class_names=arr)
    exp = explainer.explain_instance(sentiment.dev_data[idx], c.predict_proba,
num_features=num_features)
    print('Document id: %d' % idx)
    pre = c.predict_proba([sentiment.dev_data[idx]])[0][index]
    l = ''
    if pre>0.5:
        l = arr[index]
    elif index == 0:
        l = arr[1]
    else:
        l = arr[0]
    print('True class: %s' % sentiment.dev_labels[idx], 'and we predited: ',l)
    if (l == sentiment.dev_labels[idx] and pre>0.7) or (l == sentiment.dev_lab
els[idx] and pre<0.3):
        print ('We are sure that the prediction is right')
    elif (l == sentiment.dev_labels[idx] and pre>0.55 and pre<0.7) or (l == se
ntiment.dev_labels[idx] and pre<0.45 and pre>0.3):
        print ('The prediction is right but we are not strongly confident')
    elif pre<0.55 and pre>0.45:
        print ('We are unsure about the prediction')
    elif (l != sentiment.dev_labels[idx] and pre>0.7) or (l != sentiment.dev_l
abels[idx] and pre<0.3):
        print ('We were confident that the prediction is right, but it is wron
g')
    elif (l != sentiment.dev_labels[idx] and pre>0.55 and pre<0.7) or (l != se
ntiment.dev_labels[idx] and pre<0.45 and pre>0.3):
        print ('We were not strongly confident that the prediction is right, a
nd it is wrong')
```

```
    %matplotlib inline
    exp.show_in_notebook(text=True)
```

In [20]:
```
# Accuracy for the first task
tarfname = "data/sentiment.tar.gz"
clf,sentiment = model(tarfname)
```

```
-- train data
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\xinme\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
sentiment/train.tsv
4582
-- dev data
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\xinme\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
sentiment/dev.tsv
458
-- transforming data and labels

Training classifier

Evaluating
  Accuracy on train  is: 0.8967699694456569
train acc 0.8967699694456569
  Accuracy on dev  is: 0.7707423580786026
dev acc 0.7707423580786026
```

In [21]:
```
# read file for the second task
import numpy as np
import csv
with open('emails.csv') as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    label = []
    message = []
    for row in readCSV:
        if row[1] == '1' or row[1] == '0':
            label.append(row[1])
            message.append(row[0])
```

In [22]:
```
# Positive labels(spam) are around 0:1300, so our training data consists of 70
0 spam emails
# and ~2500 non-spam messages.
class Data: pass
sp = Data()
sp.train_data  = message[:700]+message[3000:]
sp.train_labels = label[:700]+label[3000:]
sp.dev_data  = message[700:3000]
sp.dev_labels = label[700:3000]
```


In [23]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
sp.tf_vect = TfidfVectorizer(min_df=1, max_df=0.5)
sp.trainX = sp.tf_vect.fit_transform(sp.train_data)
sp.devX = sp.tf_vect.transform(sp.dev_data)
from sklearn import preprocessing
sp.le = preprocessing.LabelEncoder()
sp.le.fit(sp.train_labels)
sp.target_labels = sp.le.classes_
sp.trainy = sp.le.transform(sp.train_labels)
sp.devy = sp.le.transform(sp.dev_labels)
```

In [24]:
```
# The accuracy for the second task is high and might has a risk of overfitting
spclf = train_classifier(sp.trainX, sp.trainy)
print("\nEvaluating")
print("train acc",evaluate(sp.trainX, sp.trainy, spclf, 'train'))
print("dev acc",evaluate(sp.devX, sp.devy, spclf, 'dev'))
```

```
Evaluating
  Accuracy on train  is: 0.9918272037361354
train acc 0.9918272037361354
  Accuracy on dev  is: 0.9578260869565217
dev acc 0.9578260869565217
```

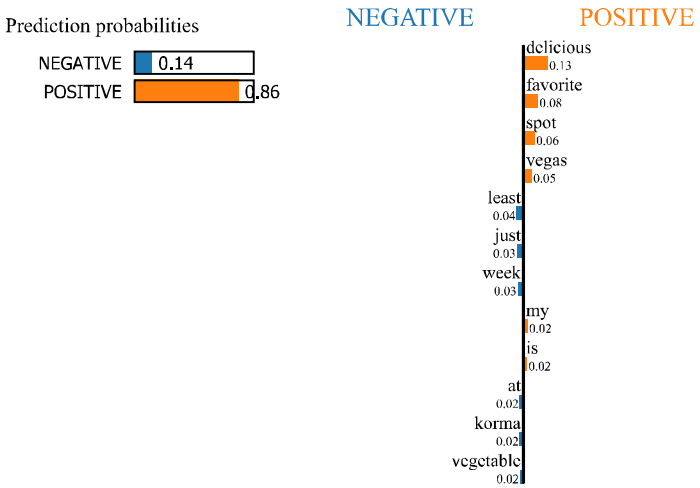
1.1

In the following example, we are strongly confident (i.e. one side of probability > 0.7) about the prediction and it turns out that it is right. This sentence contains many indicative words for the positive sentiment, such as delicious, favorite and spot. We can see that without these three words, the probability for positive will only be around 0.6.

In [160]:
```
#Sure and correct
PredictionExplaination(clf,sentiment,55,0,12,['NEGATIVE','POSITIVE'])
PredictionDecomposition(clf,sentiment,55,1,['delicious','favorite','spot'],['NEGATIVE','POSITIVE'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() requires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 55
True class: POSITIVE and we predited:  POSITIVE
We are sure that the prediction is right
```



Text with highlighted words

my new favorite spot in vegas i go at least once a week the malai kofta korma is just delicious as well as the vegetable samosa garlic naan and i

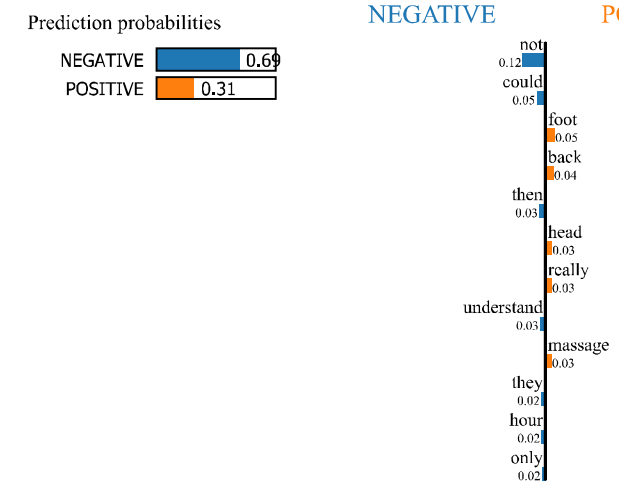
```
['delicious', 'favorite', 'spot']  have a large effect on the decision of this sentence
The difference between Original prediction for 0.8589244256895844  and Prediction removing some features:  0.6042263810499489  is  -0.2546980446396355
```

This is an example that we are somewhat confident but the prediction is wrong. By removing not, we notice that without 'not', the probability for negative is just around 0.56, which shows a problem of this classifier. Since the classifier is linear and we use the tfidf weights, there exsit words with very high weights such that if a sentence contains one or two high weighted words, the decision will be dramatically changed. A way to improve this method is to create a list of top and bottom k words and check if the sentence contains that word before training on it. If the word exist and has a large difference with other words, then an algorithm to balance the words weights in this sentence is needed.

In [161]:
```
#Somewhat confident but incorrect
PredictionExplaination(clf,sentiment,88,0,12,['NEGATIVE','POSITIVE'])
PredictionDecomposition(clf,sentiment,88,0,['not','could'],['NEGATIVE','POSITIVE'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() requires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 88
True class: POSITIVE and we predited:  NEGATIVE
We were not strongly confident that the prediction is right, and it is wrong
```



Text with highlighted words

foot massage for an hour not entirely only working on the foot they syart with the head shoulders arms foot legs and then the back for could really understand

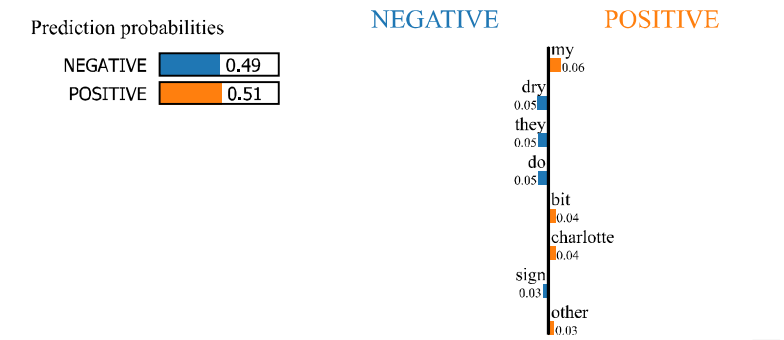
```
['not', 'could']  have a small effect on the decision of this sentence
The difference between Original prediction for 0.6863415027951717  and Prediction removing some features:  0.5585075444847576  is  -0.1278339583104141
```

This is a case when the positive and negative probabilities are very close, and therefore we are not sure about our prediction at all. Obviously, the weights and number of positive and negative words in this sentence are close and balanced. Thus, it is hard for the classifier to make a decision. The method to improve this is to utilize the context, such as n-gram or LSTM, to have more information to decide the sentence sentiment.

In [11]:
```
#Unsure
PredictionExplaination(clf,sentiment,80,0,8,['NEGATIVE','POSITIVE'],)
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 80
True class: POSITIVE and we predited:  POSITIVE
We are unsure about the prediction
```



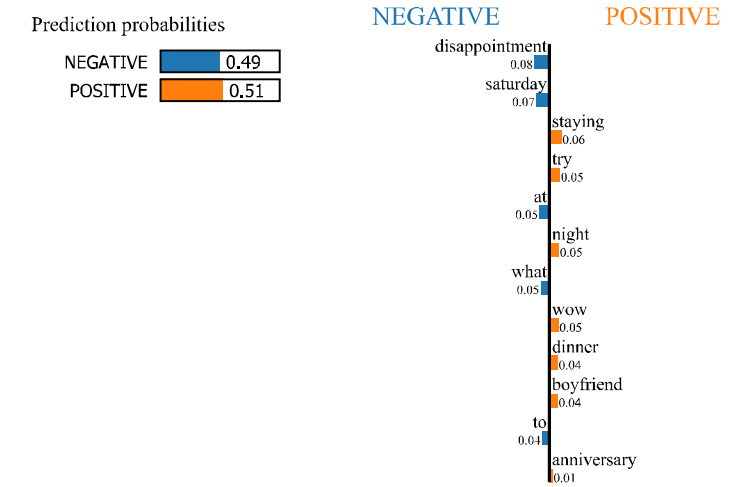
Text with highlighted words

my dry cleaner of choice in charlotte other than closing a bit early for my they do open bright and early sign up for their mailing list they send

In [8]:
```
PredictionExplaination(clf,sentiment,92,0,12,['NEGATIVE','POSITIVE'],)
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 92
True class: NEGATIVE and we predited:  POSITIVE
We are unsure about the prediction
```



Text with highlighted words

wow what a disappointment the boyfriend and i were celebrating out anniversary by staying at the pointe and decided to try this place for dinner on a saturday night at

1.2

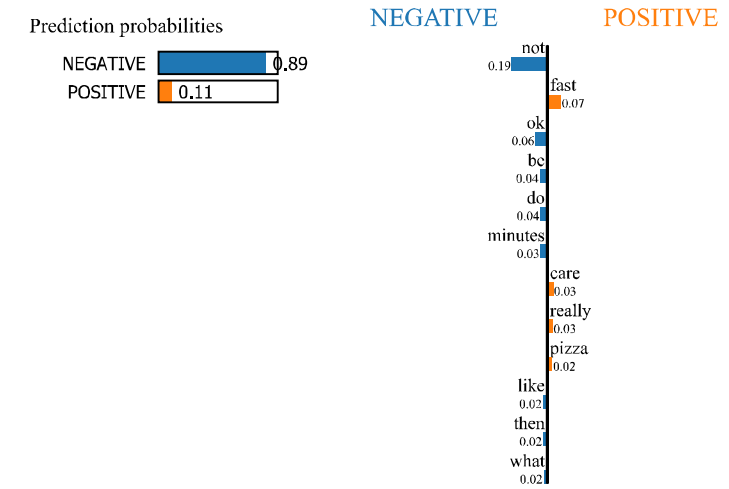
These are examples of overconfidence. For the first one, the wrong prediction probability is 0.89 since I did not find one greater than 0.9. This example is wierd, as after reading this sentence, I do not think the customer is satisfied, but it turns our that this sentence was labeled to be positive. The second example makes me feel surprised, as some frequent neutral words such as 'went', 'to', 'could' are weighted similar as words indicating strong negative attitude in the daily life such as 'but'. I guess this is because these words do appear more in the negative sentences. The issue is that these words also appear in neutral or positive sentences easily. To solve this issue, I suggest to create a list to store frequent words with high weights and somewhat decrease their weights.

In [163]:
```
# Overconfident
PredictionExplaination(clf,sentiment,260,0,12,['NEGATIVE','POSITIVE'])
PredictionExplaination(clf,sentiment,42,0,12,['NEGATIVE','POSITIVE'])
```



```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 260
True class: POSITIVE and we predited:  NEGATIVE
We were confident that the prediction is right, but it is wrong
```

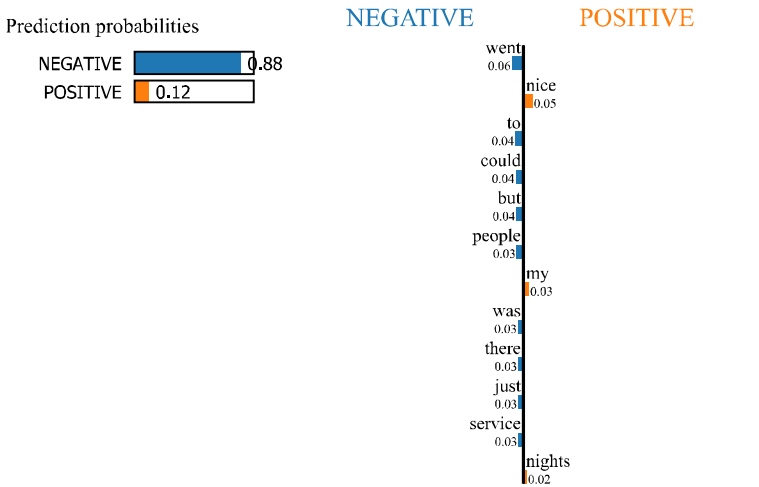


Text with highlighted words

ok so this is not fast pizza so if you need it in ten minutes and do really care what it tastes like then this may not be your place

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 42
True class: POSITIVE and we predited:  NEGATIVE
We were confident that the prediction is right, but it is wrong
```

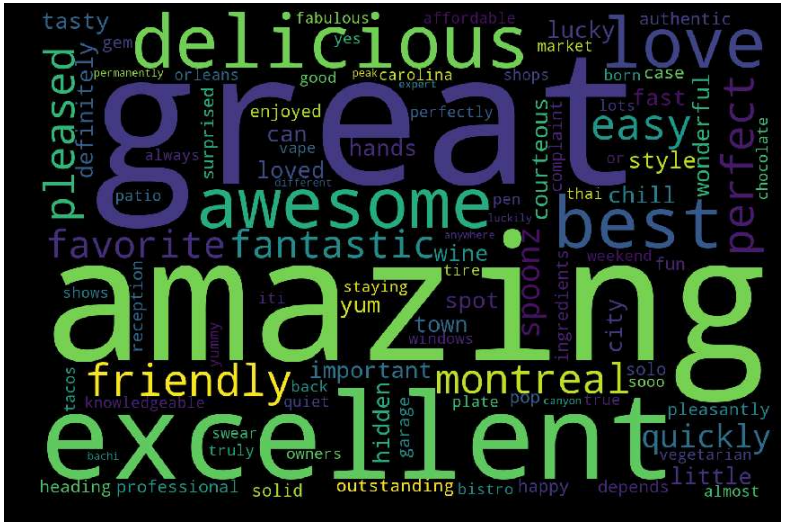


3.2

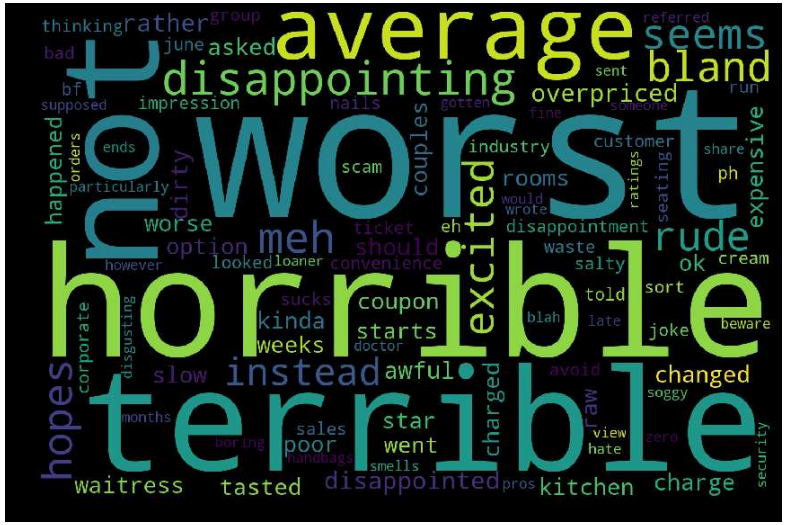
The creative part of my project is to show the most weighted words of each categories in a cloud. The word cloud picture is very straightforward and intuitive, since if we just print our words with their weights, although we can have an idea about the order of word weights, it is hard for us to compare the actual values of any two of them. By looking at the picture, we observe that althouth 'friendly' is the 9th important words for positive sentiment, it is much smaller than top 3 words such as 'great','amazing', and 'excellent'. Therefore, this is a good tool for people to understand the model easily and to determine whether we should trust this model by checking whether these words are expected and reasonable.

In [226]:
```
topk(100,sentiment)
```

```
--------------------------------------------------------------------------------
----------------------
Top k=100
--------------------------------------------------------------------------------
----------------------
dict_keys(['luckily', 'canyon', 'different', 'anywhere', 'peak', 'expert', 'b
achi', 'permanently', 'born', 'fabulous', 'lots', 'pen', 'tire', 'patio', 'go
od', 'weekend', 'always', 'staying', 'tacos', 'truly', 'carolina', 'or', 'sho
ps', 'vape', 'windows', 'yummy', 'affordable', 'shows', 'bistro', 'plate', 'o
rleans', 'yes', 'swear', 'owners', 'iti', 'sooo', 'perfectly', 'market', 'cho
colate', 'vegetarian', 'almost', 'thai', 'quiet', 'true', 'back', 'knowledgea
ble', 'complaint', 'gem', 'garage', 'reception', 'heading', 'ingredients', 'c
ase', 'surprised', 'authentic', 'solo', 'depends', 'pleasantly', 'solid', 'en
joyed', 'happy', 'fun', 'professional', 'pop', 'outstanding', 'chill', 'yum',
'city', 'fast', 'hidden', 'loved', 'wine', 'hands', 'style', 'town', 'lucky',
'tasty', 'wonderful', 'courteous', 'definitely', 'spot', 'little', 'can', 'im
portant', 'spoonz', 'quickly', 'favorite', 'fantastic', 'montreal', 'please
d', 'easy', 'perfect', 'friendly', 'love', 'best', 'awesome', 'delicious', 'e
xcellent', 'amazing', 'great'])
```




```
--------------------------------------------------------------------------------
----------------------
Bottom k=100
--------------------------------------------------------------------------------
----------------------
dict_keys(['worst', 'horrible', 'terrible', 'not', 'average', 'disappointin
g', 'meh', 'excited', 'seems', 'bland', 'hopes', 'rude', 'instead', 'disappoi
nted', 'rather', 'overpriced', 'awful', 'dirty', 'should', 'coupon', 'charg
e', 'asked', 'slow', 'expensive', 'star', 'changed', 'tasted', 'raw', 'kitche
n', 'kinda', 'worse', 'waitress', 'rooms', 'went', 'ok', 'poor', 'weeks', 'ha
ppened', 'option', 'starts', 'couples', 'charged', 'disappointment', 'nails',
'waste', 'impression', 'sort', 'eh', 'told', 'run', 'customer', 'group', 'p
h', 'scam', 'looked', 'thinking', 'bad', 'salty', 'avoid', 'corporate', 'conv
enience', 'sales', 'industry', 'cream', 'bf', 'ticket', 'june', 'joke', 'suck
s', 'seating', 'particularly', 'would', 'soggy', 'orders', 'share', 'view',
'late', 'gotten', 'smells', 'zero', 'wrote', 'boring', 'sent', 'beware', 'rat
ings', 'ends', 'supposed', 'however', 'doctor', 'referred', 'handbags', 'disg
usting', 'blah', 'security', 'someone', 'pros', 'fine', 'loaner', 'months',
'hate'])
```



To test our model, I simulate customer to create the input sentences and let the model explain it. I use positive words frequently used in the daily life in the first sentence and negative words in the second sentence. For the thrid sentence, I try to use indicative words of both postive and negative sentiment to confuse the model, although the model is not strongly confident about is result, it makes the right prediction. The model perform well on classfying these three types of sentences, so we can trust this model to do easy classification.

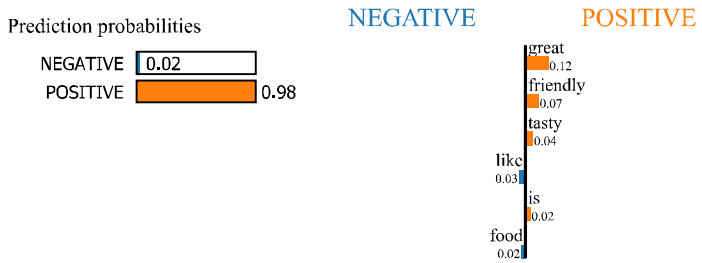
In [12]:
```
doc = ['the food is tasty and great, the staff is nice and friendly, I really
 like it']
inputtext(clf,doc,6,['NEGATIVE','POSITIVE'],sentiment)
```



```
The tf value for positive words are  {'it': 0.015934223922744243, 'great': 3.
1375644645444654, 'really': 0.38301920312156446, 'staff': 0.5651027695954007,
'nice': 0.4592422445764084, 'friendly': 1.7717976246953597, 'is': 0.364076986
1511341, 'tasty': 1.3731390600211306}
The absolute tf value for negative words are  {'food': 0.7191329090512484, 'l
ike': 0.32527870888237814}
```


```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  POSITIVE
We are sure that the prediction is right
```



Text with highlighted words

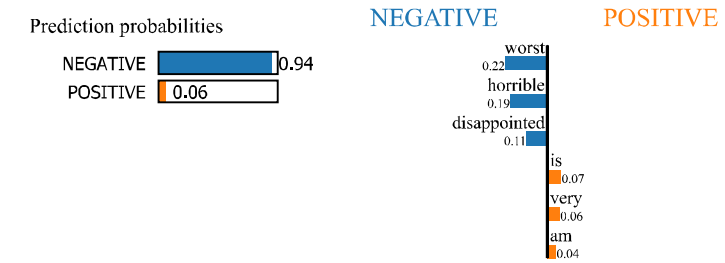
the food is tasty and great, the staff is nice and friendly, I really like it

In [13]:
```
doc = ['this is the worst place I have ever been, the service is horrible and
 I am very disappointed']
inputtext(clf,doc,6,['NEGATIVE','POSITIVE'],sentiment)
```

```
The tf value for positive words are  {'have': 0.06815268710571232, 'very': 0.
701215011475535, 'been': 0.22747466620416476, 'ever': 0.6133601047239609, 'i
s': 0.3640795279155526, 'am': 0.39911407176038916}
The absolute tf value for negative words are  {'this': 0.11751169129159791,
'place': 0.23288409500823218, 'service': 0.4171074053299077, 'horrible': 2.79
9687374332022, 'worst': 3.5026038173106273, 'disappointed': 1.613556427351660
5}
```


```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  NEGATIVE
We are sure that the prediction is right
```

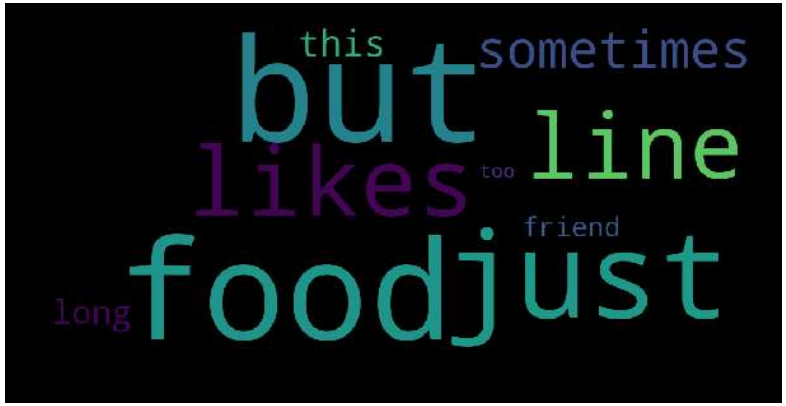
Text with highlighted words

this is the worst place I have ever been, the service is horrible and I am very disappointed

In [14]:
```
doc = ['My friend Amy really likes this resterant, but I think the food is jus
t and sometimes the line is way too long.']
inputtext(clf,doc,6,['NEGATIVE','POSITIVE'],sentiment)
```

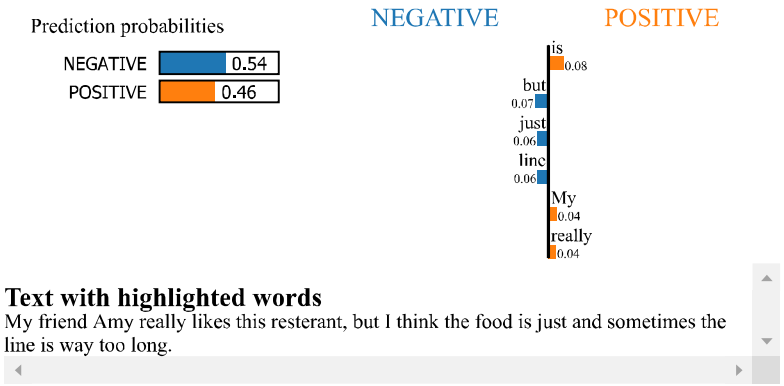


```
The tf value for positive words are  {'my': 0.6262139266251737, 'really': 0.3
830155910348299, 'is': 0.3640800452943595, 'way': 0.4922825068065372, 'thin
k': 0.06508862184455214, 'amy': 0.6889166676266647}
The absolute tf value for negative words are  {'this': 0.11751295559580446,
'food': 0.7191304446244566, 'but': 0.7749861752442907, 'just': 0.528613192183
9009, 'too': 0.029408248468671624, 'friend': 0.06262641272437801, 'long': 0.0
990982084594576, 'line': 0.42947513774633794, 'sometimes': 0.2894035766788951
6, 'likes': 0.476871892605034}
```




```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  NEGATIVE
We are unsure about the prediction
```



2.1

In the second task, I performed a spam v.s. non-spam classification using the dataset https://www.kaggle.com/karthickveerakumar/spam-filter (https://www.kaggle.com/karthickveerakumar/spam-filter) . There are around 5500 emails in total and around 1300 of them are labeled as spams.

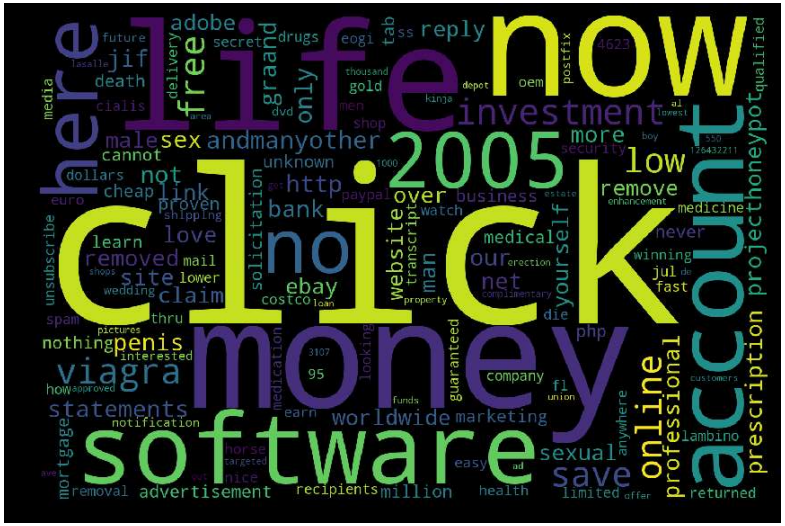
2.2

I use the same model as the first task. Here is the top 150 indicative words for spam v.s. nonspam.

In [227]:
```
topk(150,sp)
```

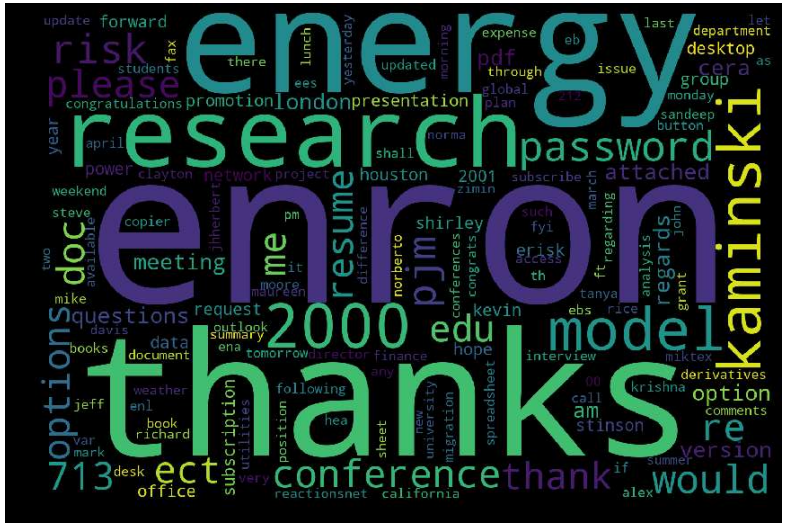
Top k=150

```
dict_keys(['kinja', 'shops', 'ave', 'area', '1000', 'lasalle', 'approved', 'g
et', '3107', 'erection', 'thousand', 'out', '126432211', 'estate', 'funds',
'targeted', 'property', '550', 'de', 'customers', 'offer', 'ad', 'lowest', 'l
oan', 'union', 'enhancement', 'al', 'boy', 'pictures', 'depot', 'complimentar
y', 'postfix', 'shlpplng', 'dvd', 'interested', 'wedding', 'future', 'notific
ation', 'qualified', 'medicine', 'medication', 'horse', 'die', '95', 'earn',
'looking', 'fast', 'eogi', 'watch', 'unsubscribe', 'delivery', 'media', 'thr
u', 'ss', 'euro', 'removal', 'drugs', 'anywhere', 'security', 'gold', 'guaran
teed', 'cialis', 'oem', 'secret', 'jul', 'costco', 'transcript', 'dollars',
'men', 'lambino', 'lower', 'paypal', 'returned', 'winning', '4623', 'health',
'limited', 'company', 'nice', 'spam', 'mail', 'fl', 'easy', 'how', 'shop', 'r
ecipients', 'learn', 'never', 'solicitation', 'cheap', 'business', 'million',
'death', 'cannot', 'medical', 'unknown', 'mortgage', 'marketing', 'proven',
'nothing', 'php', 'tab', 'advertisement', 'male', 'sexual', 'remove', 'adob
e', 'yourself', 'ebay', 'website', 'worldwide', 'claim', 'reply', 'link', 'mo
re', 'man', 'projecthoneypot', 'professional', 'love', 'site', 'over', 'net',
'removed', 'prescription', 'graand', 'sex', 'jif', 'bank', 'statements', 'onl
y', 'penis', 'our', 'http', 'not', 'andmanyother', 'low', 'save', 'free', 'on
line', 'viagra', 'investment', 'no', 'here', '2005', 'account', 'software',
'now', 'life', 'money', 'click'])
```



Bottom k=150

```
dict_keys(['enron', 'thanks', 'energy', 'research', '2000', 'kaminski', 'mode
l', 'password', 'pjm', 'risk', 'would', 'conference', '713', 'doc', 're', 'pl
ease', 'thank', 'options', 'me', 'ect', 'edu', 'resume', 'questions', 'meetin
g', 'london', 'attached', 'cera', 'option', 'version', 'am', 'regards', 'pd
f', 'presentation', 'year', 'group', 'shirley', 'kevin', 'houston', 'reques
t', '2001', 'subscription', 'stinson', 'power', 'desktop', 'office', 'hope',
'data', 'promotion', 'forward', 'erisk', 'network', 'congratulations', 'var',
'very', 'zimin', 'weather', 'enl', 'jeff', 'outlook', 'issue', 'subscribe',
'fax', 'california', 'spreadsheet', 'mike', 'let', 'summer', 'steve', 'globa
l', 'sandeep', 'new', 'finance', 'ft', 'th', 'norma', 'there', 'update', 'fy
i', 'it', 'department', 'clayton', 'weekend', 'following', 'two', 'reactionsn
et', 'book', 'tanya', 'tomorrow', 'miktex', 'updated', 'as', 'plan', 'comment
s', 'mark', 'rice', 'button', 'last', 'derivatives', 'document', 'pm', 'davi
s', 'lunch', 'sheet', 'monday', 'call', 'expense', 'moore', 'conferences', 'r
egarding', 'interview', 'alex', 'through', 'maureen', 'krishna', 'congrats',
'difference', 'john', 'position', 'ebs', 'grant', 'migration', 'project', '0
0', '212', 'such', 'books', 'students', 'summary', 'yesterday', 'eb', 'direct
or', 'shall', 'university', 'jhherbert', 'morning', 'utilities', 'copier', 'n
orberto', 'ees', 'march', 'desk', 'if', 'richard', 'april', 'hea', 'access',
'ena', 'any', 'analysis', 'available'])
```

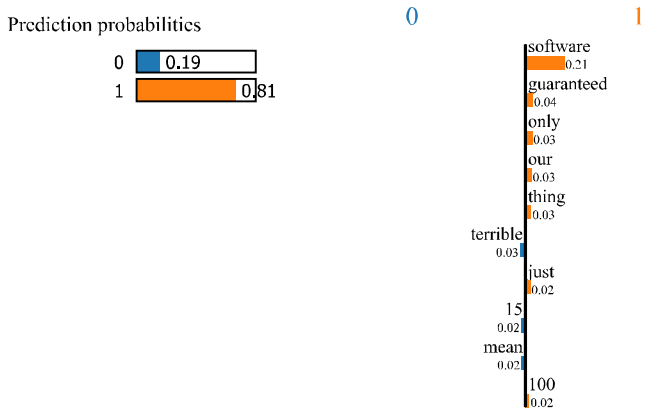


It is surprised that how large is the weight of 'software'. This is dangerous, since an email containing this word might be decided to be spam if it does not have strong indicative words of non-spam. To avoid this misclassification, we should check the distribution of word weights at first and try to make it more balanced such as making their absolute value to be the squareroot.

In [25]:
```
#Sure and correct
PredictionExplaination(spclf,sp,340,0,10,['0','1'])
PredictionDecomposition(spclf,sp,340,1,['software'],['0','1'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 340
True class: 1 and we predited:  1
We are sure that the prediction is right
```



Text with highlighted words

Subject: softwares cds all software under $ 15 and $ 99 ! only our software is guaranteed 100 % legal . a waist is a terrible thing to mind . it is impossible to say just what i mean !

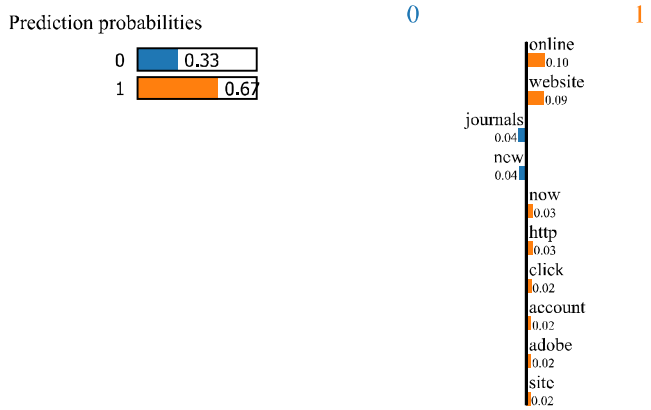
```
['software']  have a small effect on the decision of this sentence
The difference between Original prediction for 0.8146909149848426  and Predic
tion removing some features:  0.6394464215012393  is  -0.17524449348360327
```

This is an example that could be easily misclassified, since it is an advertisement from an official institution and its wording is similar as the language used in the spam email. It is very difficult to distinguish between normal advertisement and spam by just depending on word weights. To understand the specific differences between them, we need to train model on a dataset of nonspam advertisement v.s. spam and try to interprete it.

In [194]:
```
#Somewhat confident but incorrect
PredictionExplaination(spclf,sp,945,0,10,['0','1'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 945
True class: 0 and we predited:  1
We were not strongly confident that the prediction is right, and it is wrong
```



Text with highlighted words

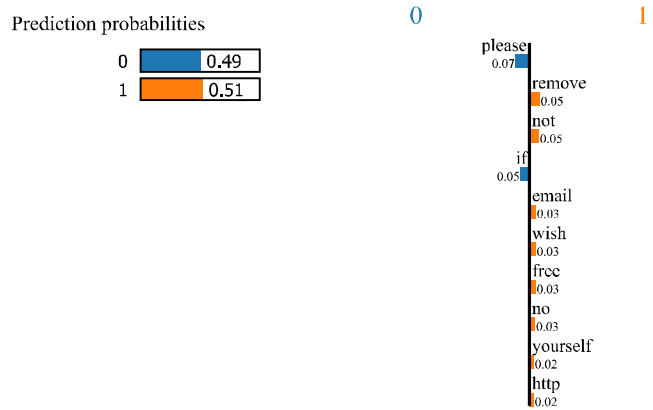
Subject: introducing the new iijournals online ! institutional investor journals are now published online ! dear subscriber , your institutional investor journals subscription now includes free access to a full - text website . visit www . iijournals . com and log onto the new iijournals homepage - - from there you can go to any of the journal specific sites . you can logon today with the following account information . userid : 13417514 password : 13417514 new site features at http : / / www . iijournals . com : - read the full - text of the current issue online before the paper edition reaches your mailbox . - use a keyword search engine to search through the entire listing of iijournals abstracts . - access online any article that has been published since january 1999 . - update your personal information , change your mailing address , or sample other institutional investor journals . to read full - text articles : before accessing the full - text of articles on any institutional investor journals web site , you need to install adobe acrobat 4 . 0 and fileopen . this is a one - time process and should take only a few minutes . please visit our " how to read articles " page on the website , click here to link : http : / / www

This example is interesting, since it is a spam email, but it disguises well. It uses formal phrases such as 'please let me know', 'negotiation', and 'sincerely, michael'. As a result, the model is unsure about the prediction. This is another problem of word weights. However, I cannot come up with an idea to solve this problem, because we are doing text analysis, and the writer of this message is good at concealing the goal of fraud under a piece of formal style of text. This might be one of the weakness of automated machine learning. It needs more personal experience to distinguish spam email like this.

In [202]:
```
#Unsure
PredictionExplaination(spclf,sp,439,0,10,['0','1'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 439
True class: 1 and we predited:  1
We are unsure about the prediction
```



Text with highlighted words

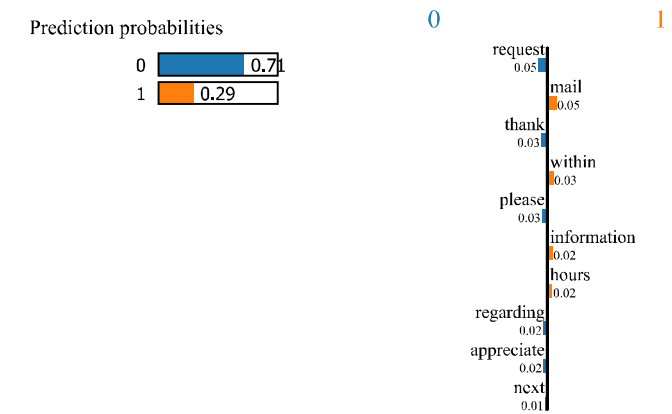
Subject: glasglow follow up hello objective - view . de i wanted follow up to yesterdays email to point out the glasglow partner program to those who may be interested . we offer a 40 % discount and exclusive license agreement for your region . we do dropship using dhl free of charge . if this may be something that interests you please let me know . if we are already in negotiation , exchanged links or you have listed yourself on the remove list please disregard this email and no further reply is required . sincerely michael www . glasglow . com if you do not wish to receive further updates please enter your email at http : / / www . cbxsales . com / un . html . they have agreed to send us the remove lists so that we do not keep bothering those that do not wish to be bothered .

The label of this email is out of my expectation. To be honest, after reading this email, I don't think it is spam, as it did not ask the reciever to click any link or buy anything. So I think this is the edge case of spam classification and we cannot avoid it.

In [223]:
```
# Somewhat Overconfident
PredictionExplaination(spclf,sp,529,0,10,['0','1'])
```

```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

Document id: 529
True class: 1 and we predited:  0
We were confident that the prediction is right, but it is wrong
```



Text with highlighted words

Subject: information request received we are in receipt of your e - mail regarding additional information . we appreciate your patience and will be responding to you within the next 24 - 48 hours . please be advised that this is an automated e - mail response . thank you , millenium precision , inc .

Next, I will simulate spam and nonspam email to check the model.

In the first example, although the classification result is correct, I notice that 'promotion' is regarded as non-spam words, probably because it is relatively formal compared with 'sales' and 'discount'. In the second example, the model made a wrong decision, which made me kind of doubt its reliability. The words used in the second example should be indicative enough to be confident and make a right deicsion. From the third example, we can see that the model tends to classify professional words and phrases in to non-spam. The fourth example shows that if most of words of an email are neither frequently used in the spam email nor the non-spam email, although the confidence might not be very high, it will be correctly classified.

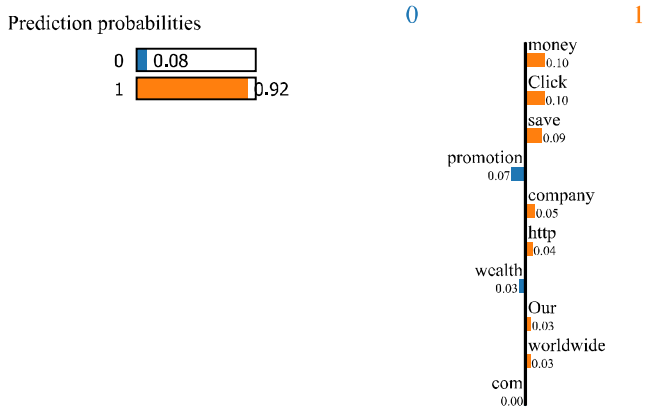
In [26]:
```
doc = ['Our company is the top 500 wealth company worldwide. Click http://prom
otion.com to save money!']
inputtext(spclf,doc,10,['0','1'],sp)
```



```
The tf value for positive words are  {'our': 0.8487406319286634, 'http': 0.85
37918315549862, 'click': 1.5052997232752554, 'money': 1.3695999979636113, 'sa
ve': 1.0003258686836138, 'top': 0.3290017566117128, '500': 0.1888513003190279
7, 'worldwide': 0.6839789184262668, 'company': 0.5442988583581195, 'wealth':
0.13671517044757786}
The absolute tf value for negative words are  {'com': 0.02762090674651639, 'p
romotion': 0.5907795224712764}
```


```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  1
We are sure that the prediction is right
```



Text with highlighted words

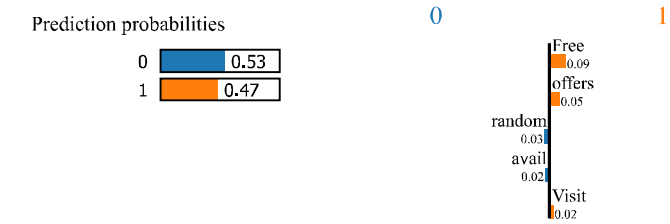
Our company is the top 500 wealth company worldwide. Click http://promotion.com to save money!

In [34]:
```
doc = ['Free Watches. Really great deals. Visit ww.random.com to avail the offers.']
inputtext(spclf,doc,5,['0','1'],sp)
```

```
The tf value for positive words are  {'free': 1.0352133661848535, 'visit': 0.
34216901661395244, 'offers': 0.35509009170571526, 'deals': 0.0108676099917694
88, 'ww': 0.12596308575759552, 'avail': 0.01105786835082094, 'watches': 0.114
76574539863212}
The absolute tf value for negative words are  {'com': 0.027622151857384685,
'great': 0.047952906266405, 'really': 0.07726991650920995, 'random': 0.052704
74935927705}
```


```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  0
We are unsure about the prediction
```

Text with highlighted words

Free Watches. Really great deals. Visit ww.random.com to avail the offers.

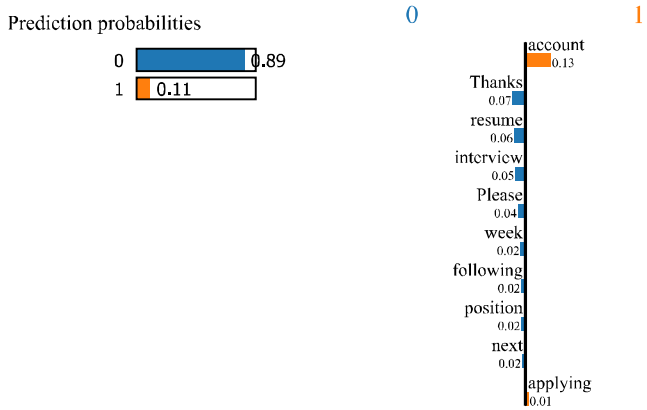
In [28]:
```
doc = ['Thanks for applying this position. Please upload your resume to your a
ccount and a following interview will be next week']
inputtext(spclf,doc,10,['0','1'],sp)
```



```
The tf value for positive words are  {'account': 1.2277217957944033}
The absolute tf value for negative words are  {'please': 0.804250913567651,
'thanks': 1.2872812774639828, 'week': 0.3410402540685759, 'following': 0.4908
677179423149, 'next': 0.1791978023561316, 'interview': 0.44617641640479133,
'position': 0.43105997405092444, 'resume': 0.7399661132224007, 'applying': 0.
08651553251143287}
```


```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  0
We are sure that the prediction is right
```

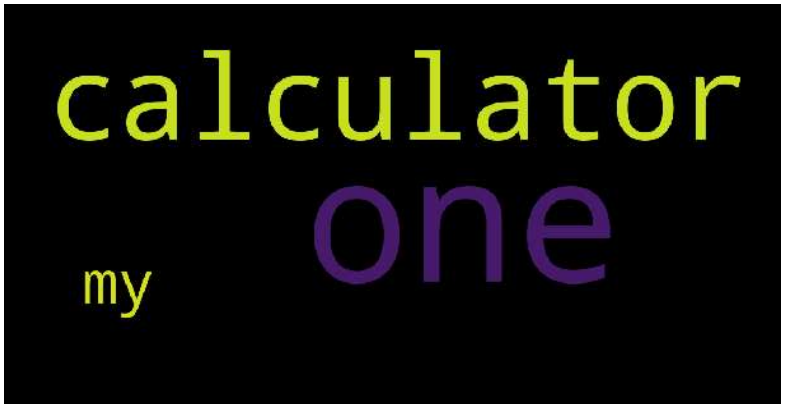


Text with highlighted words

Thanks for applying this position. Please upload your resume to your account and a following interview will be next week

In [30]:
```
doc = ['Hi Amy, do you have a graphical calculator? I need one for my exam']
inputtext(spclf,doc,10,['0','1'],sp)
```

```
The tf value for positive words are  {'my': 0.005660139217920673, 'one': 0.22
621712064781394, 'calculator': 0.17187488848371538}
The absolute tf value for negative words are  {'do': 0.03406938885356907, 'ne
ed': 0.35077836411285945, 'hi': 0.2616851902749893, 'amy': 0.1945076801724119
3, 'graphical': 0.06755448909049885, 'exam': 0.0902974181439419}
```




```
C:\Users\xinme\Anaconda2\envs\py3.6\lib\re.py:212: FutureWarning: split() req
uires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

We predited:  0
We are not strongly confident about the prediction
```