In [1]:
```
import numpy as np
import pandas as pd
import json
import sklearn
import eli5
import seaborn as sns
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

from sklearn.pipeline import make_pipeline
from lime.lime_text import LimeTextExplainer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer, TfidfVector
izer
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")
```

```
In [12]: def visualize_gender_references_frequency():
             ''' Returns matplotlib plot summarizing frequency of gender mentions
         '''

             # set the figure size
             plt.figure(figsize=(12, 6))

             # Extract percentage
             percentage_male_female = (summary_gender.pct_male + summary_gender.p
         ct_female) / 100
             df_perc = pd.DataFrame({'stars': ['1 star', '2 stars', '3 stars', '4
         stars', '5 stars'],
                                     'Gender references': round(percentage_male_f
         emale * 100, 3)})
             df_total = pd.DataFrame({'stars': ['1 star', '2 stars', '3 stars',
         '4 stars', '5 stars'],
                                      'Gender references': [100, 100, 100, 100, 1
         00]})

             # Create 2 bars
             bar1 = sns.barplot(x="stars", y="Gender references", data=df_total,
         color='Gray')
             bar2 = sns.barplot(x="stars", y="Gender references", data=df_perc, c
         olor='Black')

             # add legend
             top_bar = mpatches.Patch(color='Gray', label='Gender reference = No'
         )
             bottom_bar = mpatches.Patch(color='Black', label='Gender reference =
         Yes')
             plt.legend(handles=[top_bar, bottom_bar])

             # show the graph
             return plt.show()


         def visualize_gender_references_split():
             ''' Returns matplotlib plot summarizing split of genders '''

             # set the figure size
             plt.figure(figsize=(12, 6))

             # Extract percentage
             percentage_male = summary_gender.pct_male / (summary_gender.pct_male
         + summary_gender.pct_female)
             df_male = pd.DataFrame({'stars': ['1 star', '2 stars', '3 stars', '4
         stars', '5 stars'],
                                     'Gender': round(percentage_male * 100, 3)})
             df_total = pd.DataFrame({'stars': ['1 star', '2 stars', '3 stars',
         '4 stars', '5 stars'],
                                      'Gender': [100, 100, 100, 100, 100]})

             # Create 2 bars
             bar1 = sns.barplot(x="stars", y="Gender", data=df_total, color='Pin
         k')
             bar2 = sns.barplot(x="stars", y="Gender", data=df_male, color='Blue'
```

```
)

    # add legend
    top_bar = mpatches.Patch(color='Pink', label='Female')
    bottom_bar = mpatches.Patch(color='Blue', label='Male')
    plt.legend(handles=[top_bar, bottom_bar])

    # show the graph
    return plt.show()


def run_logistic_reg(df):
    '''Takes in pd df, then creates embeddings, trains logistic regressi
on and returns most important parameters '''

    # Split train and test set
    training_data, test_data = train_test_split(df, train_size=0.8, rand
om_state=123)
    te_y = test_data['stars']

    # Creating unigram + bigram embeddings
    vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1, 2
),
                                 min_df=3, lowercase=True, max_features=
100000)
    bow_representation = vectorizer.fit_transform(training_data['clean_t
ext'])
    bow_representation_test = vectorizer.transform(test_data['clean_tex
t'])

    best_logit = LogisticRegression(C=1, solver='liblinear',
                                    penalty='l1', max_iter=1000).fit(bow
_representation, training_data['stars'])

    # predict
    y_test_pred = best_logit.predict(bow_representation_test)

    # Evaluate model
    c_matrix_test = confusion_matrix(te_y, y_test_pred)
    # Accuracy
    acc = np.round(sklearn.metrics.accuracy_score(te_y, y_test_pred), 5)
    # Precision
    prec = np.round(sklearn.metrics.precision_score(te_y, y_test_pred, a
verage=None), 3)
    prec_micro = np.round(sklearn.metrics.precision_score(te_y, y_test_p
red, average='micro'), 5)
    # Recall
    rec = np.round(sklearn.metrics.recall_score(te_y, y_test_pred, avera
ge=None), 3)
    rec_micro = np.round(sklearn.metrics.recall_score(te_y, y_test_pred,
average='micro'), 5)
    # F1
    f1 = np.round(sklearn.metrics.f1_score(te_y, y_test_pred, average=No
ne), 3)
    f1_micro = np.round(sklearn.metrics.f1_score(te_y, y_test_pred, aver
age='micro'), 5)
```

```
    # Print model results
    print('Acc: ', acc, ' Prec: ', prec, ' Rec: ', rec, ' f1: ', f1)

    # Extract vector names
    # feature_names = vectorizer.get_feature_names_out()
    feature_names = vectorizer.get_feature_names()

    # Create summary pd
    top_x, top_y = 50, 50
    weights = eli5.show_weights(estimator=best_logit, top=(top_x, top_y
),
                                target_names=training_data['stars'])
    result = pd.read_html(weights.data)[0]
    result = result.drop([top_x, (top_x + 1)], axis=0)
    result['feature_number'] = list(map(lambda x: int(x[1:]), result.Fea
ture))
    result['feature_name'] = list(map(lambda x: feature_names[x], result
.feature_number))
    result['weight_num'] = list(
        map(lambda x: np.where(x[0] == "+", float(x[1:]), float(x[1:]) *
-1), result['Weight?']))

    return result


def visualize_feature_importance(df, review_index):
    ''' Returns word by word importance for one given review'''
    vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1, 2
),
                                 min_df=3, lowercase=True, max_features=
100000)
    bow_representation = vectorizer.fit_transform(df['clean_text'])
    bow_representation_test = vectorizer.transform(df['clean_text'])

    best_logit = LogisticRegression(C=1, solver='liblinear',
                                    penalty='l1', max_iter=1000).fit(bow
_representation,
                                                                     df[
'stars'])

    class_names = {1: '1_star', 5: '5_star'}
    LIME_explainer = LimeTextExplainer(class_names=class_names)
    c = make_pipeline(vectorizer, best_logit)

    LIME_exp = LIME_explainer.explain_instance(female_df.text[review_ind
ex], c.predict_proba)
    # print results
    print('Document id: %d' % review_index)
    print('Review: ', female_df.text[review_index])
    print('Probability 5 star =', c.predict_proba([female_df.text[review
_index]]).round(3)[0, 1])
    print('True class: %s' % class_names.get(female_df.stars[review_inde
x]))

    return LIME_exp.show_in_notebook(text=True)
```

In [3]:
```
# Read in reviews
reviews = []
with open('/Users/louisgenereux/Desktop/Term 4/Text_analytics/yelp_datas
et/' \
          'yelp_academic_dataset_review.json') as json_file:
    for rec in json_file:
        dic = json.loads(rec)
        reviews.append(dic)
print("- JSON format review data has been read")
```

```
- JSON format review data has been read
```

In [4]:
```
# Convert to pd
reviews_df = pd.DataFrame.from_records(reviews)
print("- Data converted to pd format")
```

```
- Data converted to pd format
```

In [5]:
```
# Produce summary of df
label = reviews_df['stars'].value_counts().index
observed = reviews_df['stars'].value_counts()
pct = reviews_df['stars'].value_counts() / len(reviews_df['stars'])
summary = pd.DataFrame({'stars': label,
                        'observed': observed,
                        'pct': round(pct, 4) * 100})
summary = summary.sort_values(['stars'])
```

In [6]:
```
# Reading in pre-processed gendered df
gender_mentionned = pd.read_csv('yelp_gendered.csv')
gender_mentionned_unique = gender_mentionned[(gender_mentionned['male_pr
esent'] +
                                              gender_mentionned['female_
present']) == 1]
print("- Gendered CSV read, entries referencing both genders are remove
d")
```

```
- Gendered CSV read, entries referencing both genders are removed
```

In [7]:

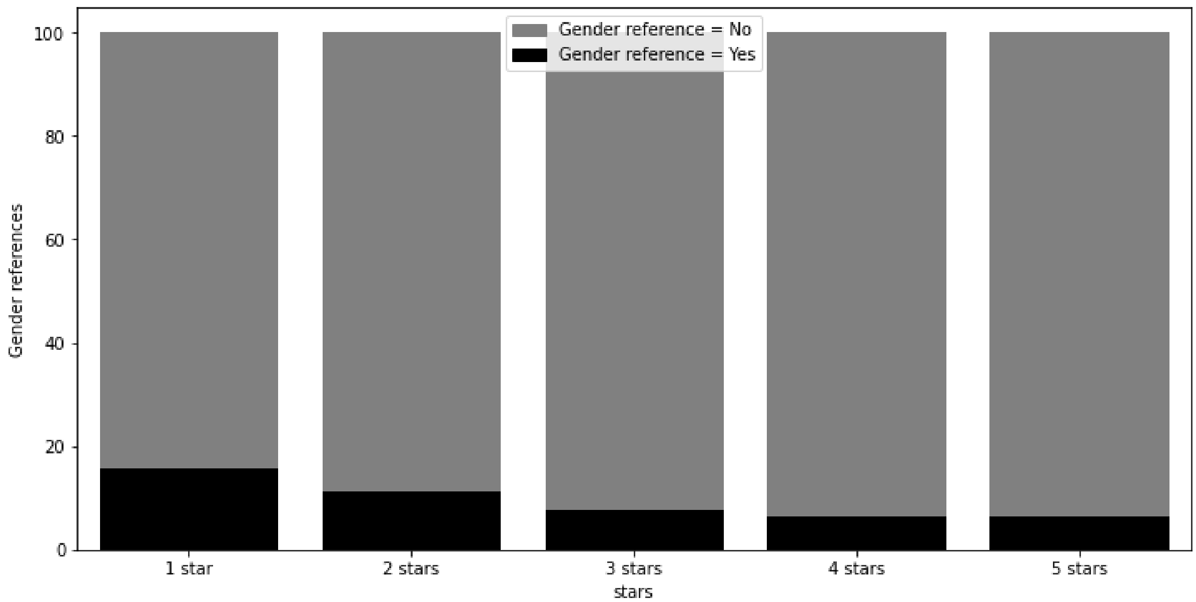
```
# Producing summary
summary_gender = gender_mentionned_unique.groupby(['stars']).agg({
                                                     'male_present': [('sum')],
                                                      'female_present': [('sum'
)]})
summary_gender['pct_male'] = round(100*summary_gender['male_present']['s
um']/
                                   list(summary['observed'].values),3)
summary_gender['pct_female'] = round(100*summary_gender['female_present'
]['sum']/
                                     list(summary['observed'].values),3
)
print("- Summary of corpus: ")
print(summary_gender)
```

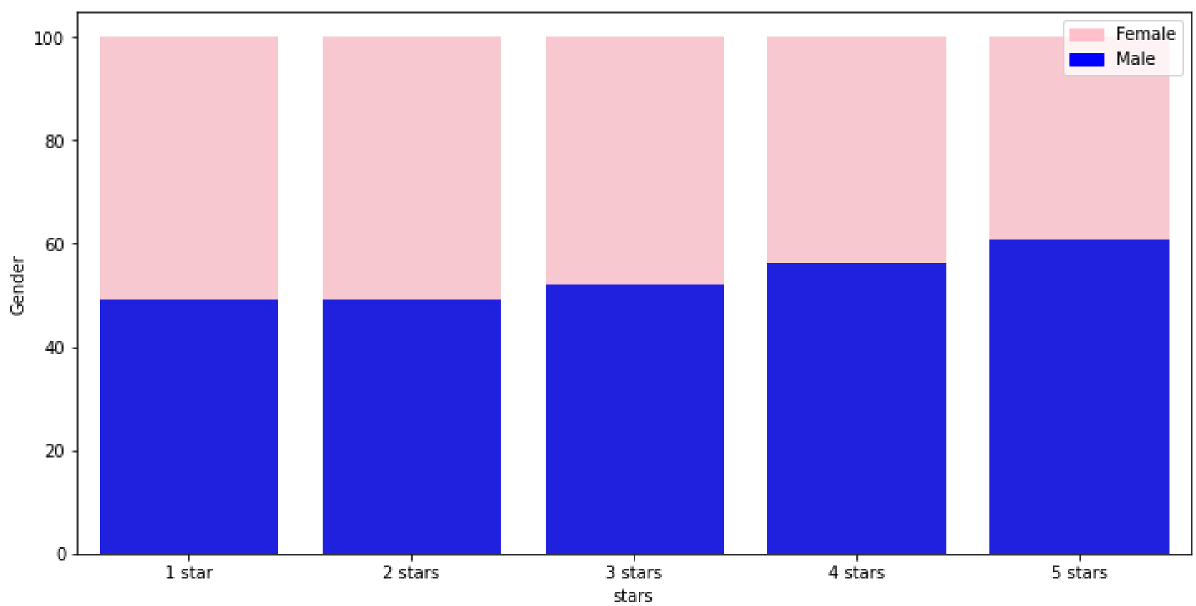
```
- Summary of corpus:
      male_present female_present pct_male pct_female
               sum            sum
stars
1.0          96876         100370    7.672      7.948
2.0          39107          40362    5.497      5.674
3.0          37504          34401    4.047      3.712
4.0          68114          52955    3.548      2.758
5.0         144697          93389    3.793      2.448
```

In [8]:

```
# Produce visuals of dataset stats
visualize_gender_references_frequency()
visualize_gender_references_split()

# Create male and female only corpa
male_df = gender_mentionned_unique[(gender_mentionned_unique['male_present'] == 1) &
                                   (gender_mentionned_unique['stars'].isin([1, 5]))]
female_df = gender_mentionned_unique[(gender_mentionned_unique['female_present'] == 1) &
                                     (gender_mentionned_unique['stars'].isin([1, 5]))]
print("- Male only and female only gendered corpa created")
```






```
- Male only and female only gendered corpa created
```

In [9]:

```python
# Identify top word predictors of star ratings in gendered corpus
print("- Running LOGISTIC REGRESSION")
print("- Testing result for men:")
result_men = run_logistic_reg(male_df)
print("- Testing result for women:")
result_women = run_logistic_reg(female_df)
print('- Top predictors of high ratings male')
print(result_men[0:10])
print('- Top predictors of low ratings female')
print(result_women[90:100])
```

```
- Running LOGISTIC REGRESSION
- Testing result for men:
Acc:  0.96608  Prec:  [0.959 0.971]  Rec:  [0.956 0.973]  f1:  [0.958
0.972]
- Testing result for women:
Acc:  0.96754  Prec:  [0.969 0.966]  Rec:  [0.969 0.966]  f1:  [0.969
0.966]
- Top predictors of high ratings male
   Weight? Feature  feature_number      feature_name weight_num
0  +29.459   x1514            1514           amazing     29.459
1  +28.732  x20195           20195         delicious     28.732
2  +23.495  x35392           35392             great     23.495
3  +22.874  x26315           26315         excellent     22.874
4  +22.726   x4562            4562           awesome     22.726
5  +22.596   x6391            6391              best     22.596
6  +21.920  x63281           63281           perfect      21.92
7  +20.010  x27827           27827         fantastic      20.01
8  +17.653  x63825           63825        phenomenal     17.653
9  +17.309  x39551           39551  highly recommend     17.309
- Top predictors of low ratings female
     Weight? Feature  feature_number    feature_name weight_num
92   -15.547   x7496            7496           bland    -15.547
93   -17.150  x92519           92519       used love     -17.15
94   -17.445  x21951           21951      disgusting    -17.445
95   -19.377   x4977            4977           awful    -19.377
96   -19.538  x86084           86084        terrible    -19.538
97   -21.051  x21837           21837   disappointing    -21.051
98   -21.170  x39454           39454        horrible     -21.17
99   -21.360  x92099           92099  unprofessional     -21.36
100  -28.067  x73041           73041            rude    -28.067
101  -33.179  x98796           98796           worst    -33.179
```

```
In [10]: # Extracting positive and negative words in the male and female situatio
n
results_pos_men = list(result_men[0:50].feature_name.values)
results_pos_women = list(result_women[0:50].feature_name.values)
results_neg_men = list(result_men[51:100].feature_name.values)
results_neg_women = list(result_women[51:100].feature_name.values)

# Identifying which words appear only in subset
men_only_pos = list(set(results_pos_men) - set(results_pos_women))
women_only_pos = list(set(results_pos_women) - set(results_pos_men))
men_only_neg = list(set(results_neg_men) - set(results_neg_women))
women_only_neg = list(set(results_neg_women) - set(results_neg_men))

# Print unique words
print('- Words that are POSITIVE predictors of strong ratings when MEN a
re mentioned only:')
print(men_only_pos)
print('- Words that are POSITIVE predictors of strong ratings when WOMEN
are mentioned only:')
print(women_only_pos)
print('- Words that are NEGATIVE predictors of strong ratings when MEN a
re mentioned only:')
print(men_only_neg)
print('- Words that are NEGATIVE predictors of strong ratings when WOMEN
are mentioned only:')
print(women_only_neg)
```

```
- Words that are POSITIVE predictors of strong ratings when MEN are men
tioned only:
['satisfied', 'notch', 'did charge', 'good', 'nicest', 'happier', 'hesi
tate', 'hesitation', 'exceptional', 'saved', 'pleasantly', 'enjoyed']
- Words that are POSITIVE predictors of strong ratings when WOMEN are m
entioned only:
['delightful', 'tasty', 'hooked', 'exactly', 'thankful', 'sweetest', 'l
oved', 'beautiful', 'accommodating', 'efficient', 'fabulous', 'gorgeou
s']
- Words that are NEGATIVE predictors of strong ratings when MEN are men
tioned only:
['condescending', 'wtf', 'excuse', 'shame', 'incompetent', 'tasteless',
'negative stars', 'response', 'ignored', 'wasted', 'unfriendly', 'flavo
rless']
- Words that are NEGATIVE predictors of strong ratings when WOMEN are m
entioned only:
['lack', 'ridiculous', 'poisoning', 'unhelpful', 'wanted love', 'charge
d', 'attitude', 'ruined', 'left', 'asked', 'zero', 'acted']
```

In [13]:

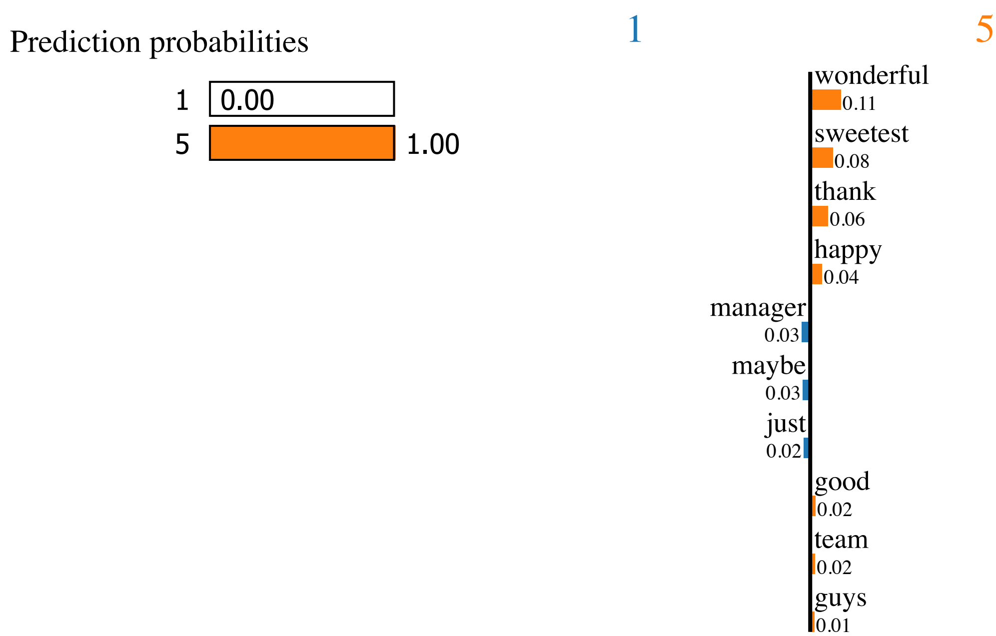
```
# Which data entries have the word attitude
female_df['attitude'] = list(map(lambda x: np.where(('attitude' in x), 1
, 0), female_df.text))

# Visualize feature importance
visualize_feature_importance(female_df ,762)
```

```
Document id: 762
Review:  I don't know what it is but the coffee I get just tastes so go
od from here. I usually go for large coffee with almond milk.

However, the 5 starts are for the morning crew that are there around 8:
30Am. You guys are absolutely wonderful. Especially the lady with the b
lue shirt on (manager maybe) she's just the sweetest and happiest perso
n ever. It's not fake happy either. I was a Starbucks fan before but
I've just turned to team Dunkin!!!

Looking forward to my next coffee and thank you for the wonderful servi
ce!!
Probability 5 star = 0.998
True class: 5_star
```



Text with highlighted words

I don't know what it is but the coffee I get just tastes so good from here. I usually go for large coffee with almond milk.

However, the 5 starts are for the morning crew that are there around 8:30Am. You guys are absolutely wonderful. Especially the lady with the blue shirt on (manager maybe) she's just the sweetest and happiest person ever. It's not fake happy either. I was a Starbucks fan before but I've just turned to team Dunkin!!!

Looking forward to my next coffee and thank you for the wonderful service!!