08_doc2vec_yelp_sentiment

September 29, 2021

1 Yelp Sentiment Analysis with doc2vec Document Vectors

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/stefan/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
True
```

[3]:
```
from pathlib import Path
import logging
from random import shuffle

import numpy as np
import pandas as pd

from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

from nltk import RegexpTokenizer
from nltk.corpus import stopwords

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
from sklearn.utils import class_weight

import lightgbm as lgb

import matplotlib.pyplot as plt
import seaborn as sns
```

1.1.1 Settings

[4]:
```
sns.set_style('white')
pd.set_option('display.expand_frame_repr', False)
np.random.seed(42)
```

1.1.2 Paths

[5]:
```
data_path = Path('..', 'data', 'yelp')
```

[6]:
```
results_path = Path('results', 'yelp')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.1.3 Logging Config

[7]:
```
logging.basicConfig(
        filename=results_path / 'doc2vec.log',
        level=logging.DEBUG,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        datefmt='%H:%M:%S')
```

1.2 Load Data

Refer to download information here.

We'll create a smaller sample of 100,000 reviews per star rating.

[8]:
```
df = pd.read_parquet(data_path / 'user_reviews.parquet').loc[:, ['stars',␣
↪'text']]
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8021122 entries, 0 to 8021121
Data columns (total 2 columns):
 #   Column  Dtype
---  ------  -----
 0   stars   float64
 1   text    object
dtypes: float64(1), object(1)
memory usage: 183.6+ MB
```

[10]:
```
df.stars.value_counts()
```

[10]:
```
5.0    3586460
4.0    1673404
1.0    1283897
```

```
3.0     842289
2.0     635072
Name: stars, dtype: int64
```

[11]:
```
stars = range(1, 6)
```

[12]:
```
sample = pd.concat([df[df.stars == s].sample(n=100000) for s in stars])
```

[13]:
```
sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 500000 entries, 7812276 to 4618307
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   stars   500000 non-null  float64
 1   text    500000 non-null  object
dtypes: float64(1), object(1)
memory usage: 11.4+ MB
```

[14]:
```
sample.stars.value_counts()
```

[14]:
```
3.0    100000
5.0    100000
4.0    100000
2.0    100000
1.0    100000
Name: stars, dtype: int64
```

[15]:
```
sample.to_parquet(results_path / 'review_sample.parquet')
```

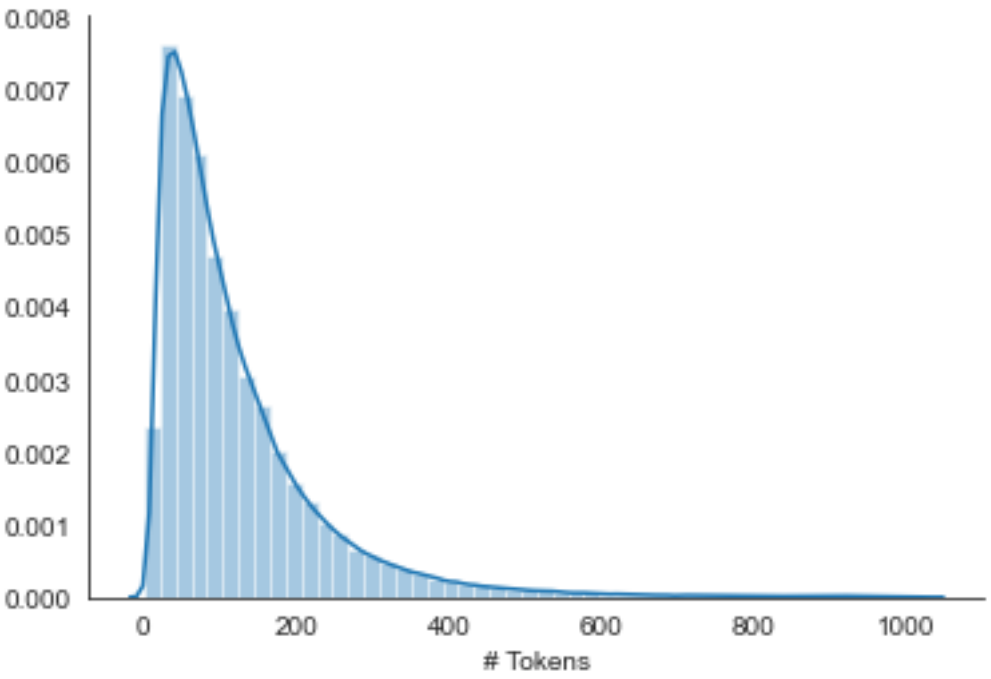
[16]:
```
sample = pd.read_parquet(results_path / 'review_sample.parquet').
↪reset_index(drop=True)
```

[17]:
```
sample.head()
```

[17]:

	stars	text
0	1.0	I have worked with Peter at Green Arrow for a…
1	1.0	Worst salon ever!!!! They make an appointment …
2	1.0	could be happenin' if anybody showed up. pitt…
3	1.0	Yeah, so nothing has changed with their servic…
4	1.0	I sent a parts request through their online em…

[18]:
```
ax = sns.distplot(sample.text.str.split().str.len())
ax.set_xlabel('# Tokens')
sns.despine();
```



1.3 Doc2Vec

1.3.1 Basic text cleaning

[19]:
```
tokenizer = RegexpTokenizer(r'\w+')
stopword_set = set(stopwords.words('english'))

def clean(review):
    tokens = tokenizer.tokenize(review)
    return ' '.join([t for t in tokens if t not in stopword_set])
```

[20]:
```
sample.text = sample.text.str.lower().apply(clean)
```

[21]:
```
sample.sample(n=10)
```

[21]:

	stars	text
106676	2.0	ordered guinea pig pizza read great reviews ca…
388751	4.0	great stop us way highpoint hubby craving shri…
235104	3.0	read roger review waiting show start thought h…
422459	5.0	finally broke tried ocp past weekend found cli…
366460	4.0	mary neill couple times drinks apps apps prett…
84223	1.0	going honest came dominoes temporarily closed …
190363	2.0	tell much wanted like place counting days open…
282482	3.0	three stars may seem little harsh one sandwich…
388041	4.0	family goes breakfast brunch weekends great li…

```
136831    2.0  gone twice food amazing however staff much car…
```

[22]:
```
sample = sample[sample.text.str.split().str.len()>10]
sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 485725 entries, 0 to 499999
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   stars   485725 non-null  float64
 1   text    485725 non-null  object
dtypes: float64(1), object(1)
memory usage: 11.1+ MB
```

1.3.2 Create sentence stream

[23]:
```
sentences = []
for i, (_, text) in enumerate(sample.values):
    sentences.append(TaggedDocument(words=text.split(), tags=[i]))
```

1.3.3 Formulate the model

[24]:
```
model = Doc2Vec(documents=sentences,
                dm=1,           # 1=distributed memory, 0=dist.BOW
                epochs=5,
                size=300,       # vector size
                window=5,       # max. distance betw. target and context
                min_count=50,   # ignore tokens w. lower frequency
                negative=5,     # negative training samples
                dm_concat=0,    # 1=concatenate vectors, 0=sum
                dbow_words=0,   # 1=train word vectors as well
                workers=4)
```

[25]:
```
pd.DataFrame(model.most_similar('good'), columns=['token', 'similarity'])
```

[25]:

	token	similarity
0	great	0.757694
1	decent	0.746465
2	awesome	0.622731
3	amazing	0.611350
4	bad	0.605432
5	yummy	0.600379
6	ok	0.600137
7	tasty	0.596531
8	excellent	0.594958
9	okay	0.592447

1.3.4 Continue training

[26]:
```
model.train(sentences, total_examples=model.corpus_count, epochs=model.epochs)
```

[27]:
```
most_similar = pd.DataFrame(model.most_similar('good'), columns=['token',␣
↪'similarity'])
most_similar
```

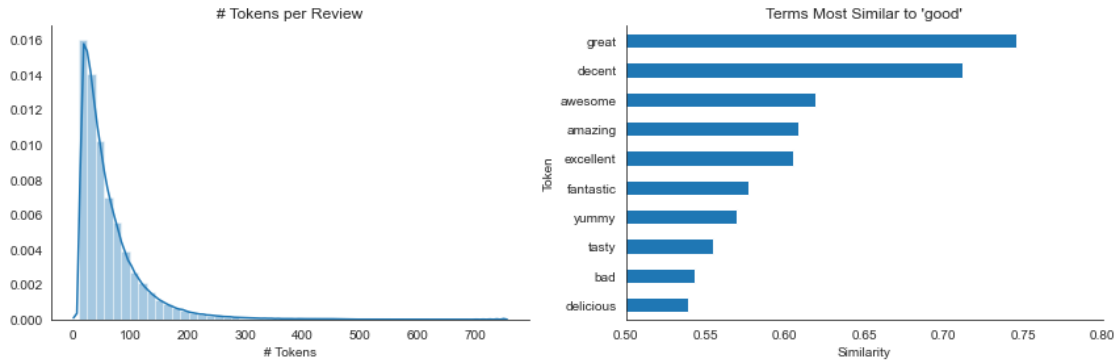
[27]:

	token	similarity
0	great	0.745322
1	decent	0.712028
2	awesome	0.618959
3	amazing	0.608737
4	excellent	0.605081
5	fantastic	0.577577
6	yummy	0.569521
7	tasty	0.555287
8	bad	0.543677
9	delicious	0.539020

[28]:
```
fig, axes =plt.subplots(ncols=2, figsize=(12, 4))
sns.distplot(sample.text.str.split().str.len(), ax=axes[0])
axes[0].set_title('# Tokens per Review')

most_similar.set_index('token').similarity.sort_values().plot.barh(ax=axes[1],
                                                                   title="Terms␣
↪Most Similar to 'good'",
                                                                   xlim=(.5, .8))
axes[1].set_xlabel('Similarity')
axes[1].set_ylabel('Token')
axes[0].set_xlabel('# Tokens')

sns.despine()
fig.tight_layout()
fig.savefig(results_path / 'doc2vec_stats', dpi=300)
```



1.4 Persist Model

[29]:
```
model.save((results_path / 'sample.model').as_posix())
```

[30]:
```
model = Doc2Vec.load((results_path / 'sample.model').as_posix())
```

1.5 Evaluate

[31]:
```
y = sample.stars.sub(1)
```

[32]:
```
size = 300
X = np.zeros(shape=(len(y), size))
for i in range(len(sample)):
    X[i] = model.docvecs[i]
```

[33]:
```
X.shape
```

[33]:
```
(485725, 300)
```

1.5.1 Train-Test Split

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)
```

[35]:
```
mode = pd.Series(y_train).mode().iloc[0]
baseline = accuracy_score(y_true=y_test, y_pred=np.full_like(y_test,␣
 ↪fill_value=mode))
print(f'Baseline Score: {baseline:.2%}')
```

```
Baseline Score: 20.15%
```

[36]:
```
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(y_train),
                                                  y_train)
```

[37]:
```
class_weights
```

[37]:
```
array([0.99618017, 0.9923514 , 0.99524889, 1.00219225, 1.01433084])
```

1.6 LightGBM

[38]:
```
train_data = lgb.Dataset(data=X_train, label=y_train)
test_data = train_data.create_valid(X_test, label=y_test)
```

[39]:
```
params = {'objective': 'multiclass',
          'num_classes': 5}
```

[40]:
```
lgb_model = lgb.train(params=params,
                      train_set=train_data,
                      num_boost_round=5000,
                      valid_sets=[train_data, test_data],
                      early_stopping_rounds=25,
                      verbose_eval=50)
```

```
Training until validation scores don't improve for 25 rounds
[50]    training's multi_logloss: 1.44166       valid_1's multi_logloss: 1.45977
[100]   training's multi_logloss: 1.38412       valid_1's multi_logloss: 1.41755
[150]   training's multi_logloss: 1.3404        valid_1's multi_logloss: 1.38813
[200]   training's multi_logloss: 1.30368       valid_1's multi_logloss: 1.36524
[250]   training's multi_logloss: 1.272 valid_1's multi_logloss: 1.34701
[300]   training's multi_logloss: 1.24457       valid_1's multi_logloss: 1.33252
[350]   training's multi_logloss: 1.22035       valid_1's multi_logloss: 1.32092
[400]   training's multi_logloss: 1.1986        valid_1's multi_logloss: 1.31161
[450]   training's multi_logloss: 1.17894       valid_1's multi_logloss: 1.30414
[500]   training's multi_logloss: 1.16084       valid_1's multi_logloss: 1.29773
[550]   training's multi_logloss: 1.14419       valid_1's multi_logloss: 1.29242
[600]   training's multi_logloss: 1.1284        valid_1's multi_logloss: 1.28762
[650]   training's multi_logloss: 1.11312       valid_1's multi_logloss: 1.28313
[700]   training's multi_logloss: 1.09771       valid_1's multi_logloss: 1.27818
[750]   training's multi_logloss: 1.08344       valid_1's multi_logloss: 1.27396
[800]   training's multi_logloss: 1.06931       valid_1's multi_logloss: 1.26982
[850]   training's multi_logloss: 1.05587       valid_1's multi_logloss: 1.26615
[900]   training's multi_logloss: 1.0426        valid_1's multi_logloss: 1.26248
[950]   training's multi_logloss: 1.02926       valid_1's multi_logloss: 1.25856
[1000]  training's multi_logloss: 1.01651       valid_1's multi_logloss: 1.25499
[1050]  training's multi_logloss: 1.00401       valid_1's multi_logloss: 1.25151
[1100]  training's multi_logloss: 0.991996      valid_1's multi_logloss: 1.24837
[1150]  training's multi_logloss: 0.980094      valid_1's multi_logloss: 1.24507
[1200]  training's multi_logloss: 0.968459      valid_1's multi_logloss: 1.2419
[1250]  training's multi_logloss: 0.956903      valid_1's multi_logloss: 1.23866
[1300]  training's multi_logloss: 0.945488      valid_1's multi_logloss: 1.23559
[1350]  training's multi_logloss: 0.934443      valid_1's multi_logloss: 1.23264
[1400]  training's multi_logloss: 0.923515      valid_1's multi_logloss: 1.22982
[1450]  training's multi_logloss: 0.91273       valid_1's multi_logloss: 1.22701
[1500]  training's multi_logloss: 0.902185      valid_1's multi_logloss: 1.2242
[1550]  training's multi_logloss: 0.891922      valid_1's multi_logloss: 1.22155
[1600]  training's multi_logloss: 0.881297      valid_1's multi_logloss: 1.21854
```

```
[1650]  training's multi_logloss: 0.871119      valid_1's multi_logloss: 1.21585
[1700]  training's multi_logloss: 0.861338      valid_1's multi_logloss: 1.2134
[1750]  training's multi_logloss: 0.851694      valid_1's multi_logloss: 1.21102
[1800]  training's multi_logloss: 0.842194      valid_1's multi_logloss: 1.20858
[1850]  training's multi_logloss: 0.832426      valid_1's multi_logloss: 1.20597
[1900]  training's multi_logloss: 0.822651      valid_1's multi_logloss: 1.20326
[1950]  training's multi_logloss: 0.813483      valid_1's multi_logloss: 1.20088
[2000]  training's multi_logloss: 0.804723      valid_1's multi_logloss: 1.19878
[2050]  training's multi_logloss: 0.795233      valid_1's multi_logloss: 1.19622
[2100]  training's multi_logloss: 0.786525      valid_1's multi_logloss: 1.19402
[2150]  training's multi_logloss: 0.77798       valid_1's multi_logloss: 1.19201
[2200]  training's multi_logloss: 0.769765      valid_1's multi_logloss: 1.19002
[2250]  training's multi_logloss: 0.761259      valid_1's multi_logloss: 1.18796
[2300]  training's multi_logloss: 0.753083      valid_1's multi_logloss: 1.18612
[2350]  training's multi_logloss: 0.744692      valid_1's multi_logloss: 1.18399
[2400]  training's multi_logloss: 0.736341      valid_1's multi_logloss: 1.18183
[2450]  training's multi_logloss: 0.728198      valid_1's multi_logloss: 1.17986
[2500]  training's multi_logloss: 0.720343      valid_1's multi_logloss: 1.17794
[2550]  training's multi_logloss: 0.712334      valid_1's multi_logloss: 1.17588
[2600]  training's multi_logloss: 0.704237      valid_1's multi_logloss: 1.17371
[2650]  training's multi_logloss: 0.696492      valid_1's multi_logloss: 1.1719
[2700]  training's multi_logloss: 0.688707      valid_1's multi_logloss: 1.16979
[2750]  training's multi_logloss: 0.681284      valid_1's multi_logloss: 1.16797
[2800]  training's multi_logloss: 0.673507      valid_1's multi_logloss: 1.16587
[2850]  training's multi_logloss: 0.666472      valid_1's multi_logloss: 1.16426
[2900]  training's multi_logloss: 0.658977      valid_1's multi_logloss: 1.16235
[2950]  training's multi_logloss: 0.6515        valid_1's multi_logloss: 1.16047
[3000]  training's multi_logloss: 0.644463      valid_1's multi_logloss: 1.15874
[3050]  training's multi_logloss: 0.637526      valid_1's multi_logloss: 1.15713
[3100]  training's multi_logloss: 0.630485      valid_1's multi_logloss: 1.15538
[3150]  training's multi_logloss: 0.623511      valid_1's multi_logloss: 1.15352
[3200]  training's multi_logloss: 0.616588      valid_1's multi_logloss: 1.15168
[3250]  training's multi_logloss: 0.609799      valid_1's multi_logloss: 1.14997
[3300]  training's multi_logloss: 0.603035      valid_1's multi_logloss: 1.14819
[3350]  training's multi_logloss: 0.5965        valid_1's multi_logloss: 1.14651
[3400]  training's multi_logloss: 0.589903      valid_1's multi_logloss: 1.14495
[3450]  training's multi_logloss: 0.583236      valid_1's multi_logloss: 1.14322
[3500]  training's multi_logloss: 0.576622      valid_1's multi_logloss: 1.14159
[3550]  training's multi_logloss: 0.570311      valid_1's multi_logloss: 1.14016
[3600]  training's multi_logloss: 0.564048      valid_1's multi_logloss: 1.13847
[3650]  training's multi_logloss: 0.557675      valid_1's multi_logloss: 1.13685
[3700]  training's multi_logloss: 0.551757      valid_1's multi_logloss: 1.13547
[3750]  training's multi_logloss: 0.545784      valid_1's multi_logloss: 1.13408
[3800]  training's multi_logloss: 0.539929      valid_1's multi_logloss: 1.13269
[3850]  training's multi_logloss: 0.534238      valid_1's multi_logloss: 1.13129
[3900]  training's multi_logloss: 0.528451      valid_1's multi_logloss: 1.12995
[3950]  training's multi_logloss: 0.522741      valid_1's multi_logloss: 1.12865
[4000]  training's multi_logloss: 0.517369      valid_1's multi_logloss: 1.12746
```

```
[4050]  training's multi_logloss: 0.511513      valid_1's multi_logloss: 1.12589
[4100]  training's multi_logloss: 0.505852      valid_1's multi_logloss: 1.12452
[4150]  training's multi_logloss: 0.50013       valid_1's multi_logloss: 1.12302
[4200]  training's multi_logloss: 0.494568      valid_1's multi_logloss: 1.1216
[4250]  training's multi_logloss: 0.488841      valid_1's multi_logloss: 1.12011
[4300]  training's multi_logloss: 0.483695      valid_1's multi_logloss: 1.11896
[4350]  training's multi_logloss: 0.478507      valid_1's multi_logloss: 1.11775
[4400]  training's multi_logloss: 0.473078      valid_1's multi_logloss: 1.11634
[4450]  training's multi_logloss: 0.467839      valid_1's multi_logloss: 1.11507
[4500]  training's multi_logloss: 0.462725      valid_1's multi_logloss: 1.11397
[4550]  training's multi_logloss: 0.45761       valid_1's multi_logloss: 1.11264
[4600]  training's multi_logloss: 0.452837      valid_1's multi_logloss: 1.11158
[4650]  training's multi_logloss: 0.447713      valid_1's multi_logloss: 1.11034
[4700]  training's multi_logloss: 0.442784      valid_1's multi_logloss: 1.10909
[4750]  training's multi_logloss: 0.438093      valid_1's multi_logloss: 1.10796
[4800]  training's multi_logloss: 0.433102      valid_1's multi_logloss: 1.10669
[4850]  training's multi_logloss: 0.428328      valid_1's multi_logloss: 1.10542
[4900]  training's multi_logloss: 0.423541      valid_1's multi_logloss: 1.1041
[4950]  training's multi_logloss: 0.41876       valid_1's multi_logloss: 1.10292
[5000]  training's multi_logloss: 0.414335      valid_1's multi_logloss: 1.1019
Did not meet early stopping. Best iteration is:
[5000]  training's multi_logloss: 0.414335      valid_1's multi_logloss: 1.1019
```

[41]:
```
lgb_pred = np.argmax(lgb_model.predict(X_test), axis=1)
```

[42]:
```
lgb_acc = accuracy_score(y_true=y_test, y_pred=lgb_pred)
print(f'Accuracy: {lgb_acc:.2%}')
```

```
Accuracy: 55.01%
```

1.7 Random Forest

[44]:
```
rf = RandomForestClassifier(n_jobs=-1,
                            n_estimators=500,
                            verbose=1,
                            class_weight='balanced_subsample')
rf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  5.3min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 12.5min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 14.3min finished
```

[44]:
```
RandomForestClassifier(class_weight='balanced_subsample', n_estimators=500,
                       n_jobs=-1, verbose=1)
```

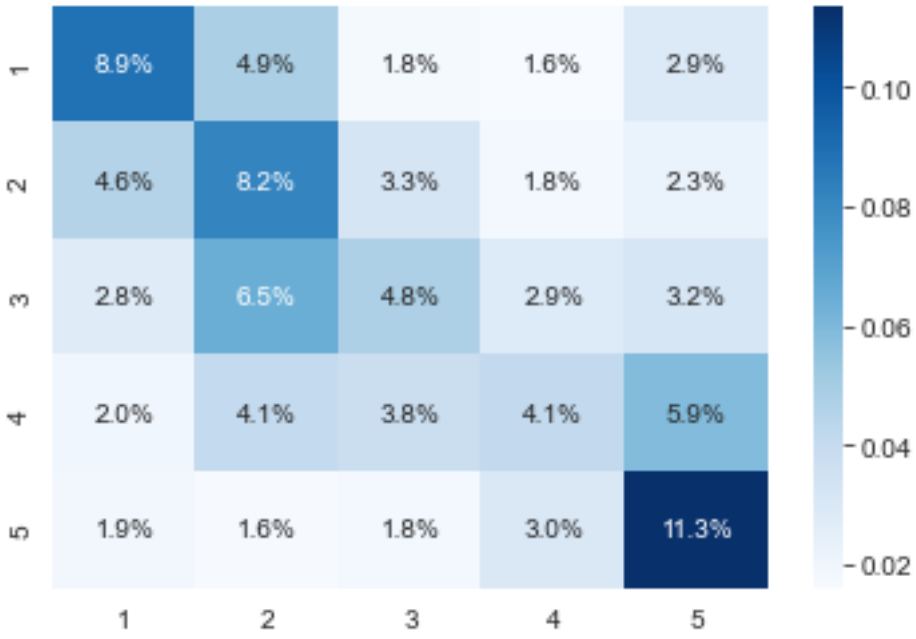
[45]:
```
rf_pred = rf.predict(X_test)
rf_acc = accuracy_score(y_true=y_test, y_pred=rf_pred)
print(f'Accuracy: {rf_acc:.2%}')
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    2.7s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:    6.2s
```

```
Accuracy: 37.37%
```

```
[Parallel(n_jobs=8)]: Done 500 out of 500 | elapsed:    7.1s finished
```

[46]:
```
cm = confusion_matrix(y_true=y_test, y_pred=rf_pred)
sns.heatmap(pd.DataFrame(cm/np.sum(cm),
                         index=stars,
                         columns=stars),
            annot=True,
            cmap='Blues',
            fmt='.1%');
```



1.8 Multinomial Logistic Regression

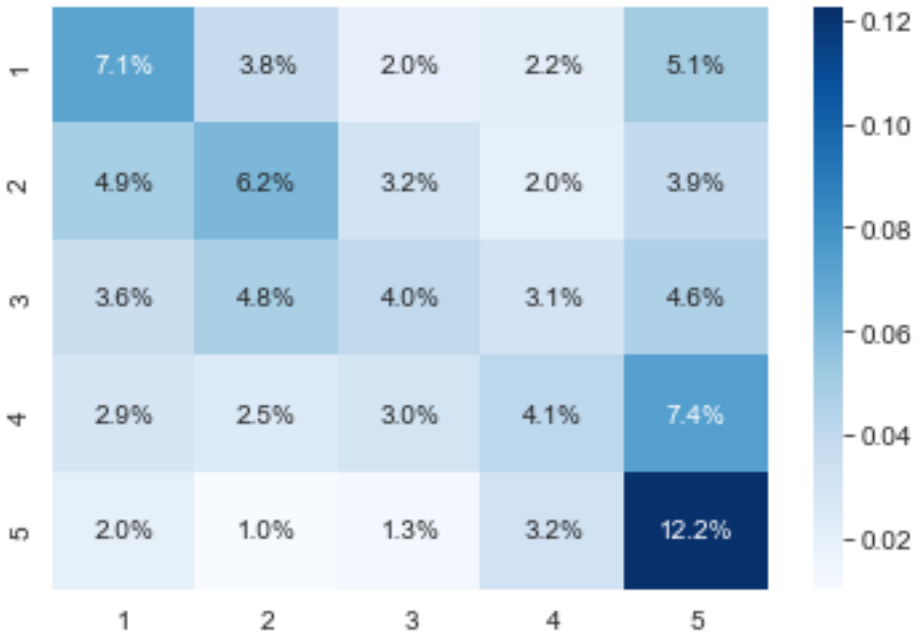
[47]:
```
lr = LogisticRegression(multi_class='multinomial',
                        solver='lbfgs',
                        class_weight='balanced')
lr.fit(X_train, y_train)
```

[47]:
```
LogisticRegression(class_weight='balanced', multi_class='multinomial')
```

[48]:
```
lr_pred = lr.predict(X_test)
lr_acc = accuracy_score(y_true=y_test, y_pred=lr_pred)
print(f'Accuracy: {lr_acc:.2%}')
```

```
Accuracy: 33.72%
```

[49]:
```
cm = confusion_matrix(y_true=y_test, y_pred=lr_pred)
sns.heatmap(pd.DataFrame(cm/np.sum(cm),
                         index=stars,
                         columns=stars),
            annot=True,
            cmap='Blues',
            fmt='.1%');
```



1.9 Comparison

[50]:
```
fig, axes = plt.subplots(ncols=3, figsize=(15, 5), sharex=True, sharey=True)

lgb_cm = confusion_matrix(y_true=y_test, y_pred=lgb_pred)
sns.heatmap(pd.DataFrame(lgb_cm/np.sum(lgb_cm), index=stars, columns=stars),
            annot=True, cmap='Blues', fmt='.1%', ax=axes[0], cbar=False)
axes[0].set_title(f'Gradient Boosting: Accuracy {lgb_acc:.2%}')

rf_cm = confusion_matrix(y_true=y_test, y_pred=rf_pred)
sns.heatmap(pd.DataFrame(rf_cm/np.sum(rf_cm), index=stars, columns=stars),
            annot=True, cmap='Blues', fmt='.1%', ax=axes[1], cbar=False)
axes[1].set_title(f'Random Forest: Accuracy {rf_acc:.2%}')

lr_cm = confusion_matrix(y_true=y_test, y_pred=lr_pred)
sns.heatmap(pd.DataFrame(lr_cm/np.sum(lr_cm), index=stars, columns=stars),
            annot=True, cmap='Blues', fmt='.1%', ax=axes[2], cbar=False)
axes[2].set_title(f'Logistic Regression: Accuracy {lr_acc:.2%}')
axes[0].set_ylabel('Actuals')
for i in range(3):
    axes[i].set_xlabel('Predicted')

sns.despine()
fig.tight_layout()
fig.savefig(results_path / 'confusion_matrix', dpi=300)
```