evaluation_optimization_solutions

October 11, 2021

1 Model evaluation and optimization for text classification

1.1 Outline

- Section ??
- Section ??
  - Section ??
  - Section ??
  - Section ??
  - Section ??
  - Section ??
- Section ??
  - Section ??
  - Section ??
  - Section ??
  - Section ??
  - Section ??

2 Data preparation

2.1 Import modules

[1]:
```
import os
import re
import numpy as np
import pandas as pd
import warnings
import graphviz
from sklearn.tree import export_graphviz
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt

#set options
warnings.simplefilter(action='ignore', category=FutureWarning)
sns.set()
%matplotlib inline
```

```
#scikit-learn is a huge library. We import what we need.
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score, train_test_split #sklearn utilities
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report #For model evaluation
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer␣
↪#Vectorizers
from sklearn.linear_model import LogisticRegressionCV #Logit with␣
↪cross-validation
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier #Random␣
↪Forest and AdaBoost classifiers
from sklearn.tree import DecisionTreeClassifier #Decision Tree classifier
from sklearn.svm import LinearSVC #Linear Support Vector classifier
```

2.2 Read and inspect dataset

[2]:
```
df = pd.read_csv('tweets.csv')
df.head(3)
```

[2]:
```
             tweet_id airline_sentiment  airline_sentiment_confidence  \
0  570306133677760513           neutral                        1.0000
1  570301130888122368          positive                        0.3486
2  570301083672813571           neutral                        0.6837

  negativereason  negativereason_confidence         airline  \
0            NaN                        NaN  Virgin America
1            NaN                        0.0  Virgin America
2            NaN                        NaN  Virgin America

  airline_sentiment_gold        name negativereason_gold  retweet_count  \
0                    NaN     cairdin                 NaN              0
1                    NaN    jnardino                 NaN              0
2                    NaN  yvonnalynn                 NaN              0

                                                text tweet_coord  \
0                @VirginAmerica What @dhepburn said.         NaN
1  @VirginAmerica plus you've added commercials t…         NaN
2  @VirginAmerica I didn't today… Must mean I n…         NaN

               tweet_created tweet_location               user_timezone
0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US & Canada)
1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US & Canada)
2  2015-02-24 11:15:48 -0800      Lets Play  Central Time (US & Canada)
```

2.2.1 Challenge

- How many tweets are in the dataset?
- How many tweets are positive, neutral and negative?
- What **proportion** of tweets are positive, neutral and negative?
- Visualize these last two questions.
- What are the three main reasons why people are tweeting negatively?

Hint: To visualize counts, you can use the `sns.countplot()` function. To visualize proportions, use the `.plot(kind='bar')` method.

[3]:
```
# solutions
print("Length is", len(df))

df['airline_sentiment'].value_counts()
```

```
Length is 14640
```

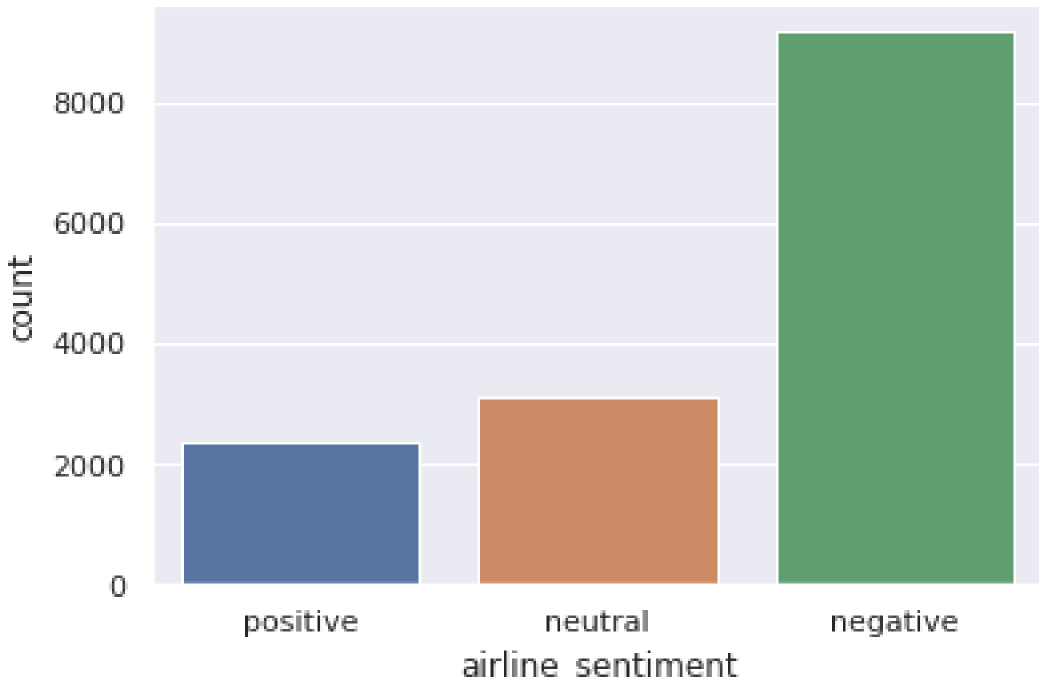
[3]:
```
negative    9178
neutral     3099
positive    2363
Name: airline_sentiment, dtype: int64
```

[4]:
```
df['airline_sentiment'].value_counts(normalize=True)
```

[4]:
```
negative    0.626913
neutral     0.211680
positive    0.161407
Name: airline_sentiment, dtype: float64
```

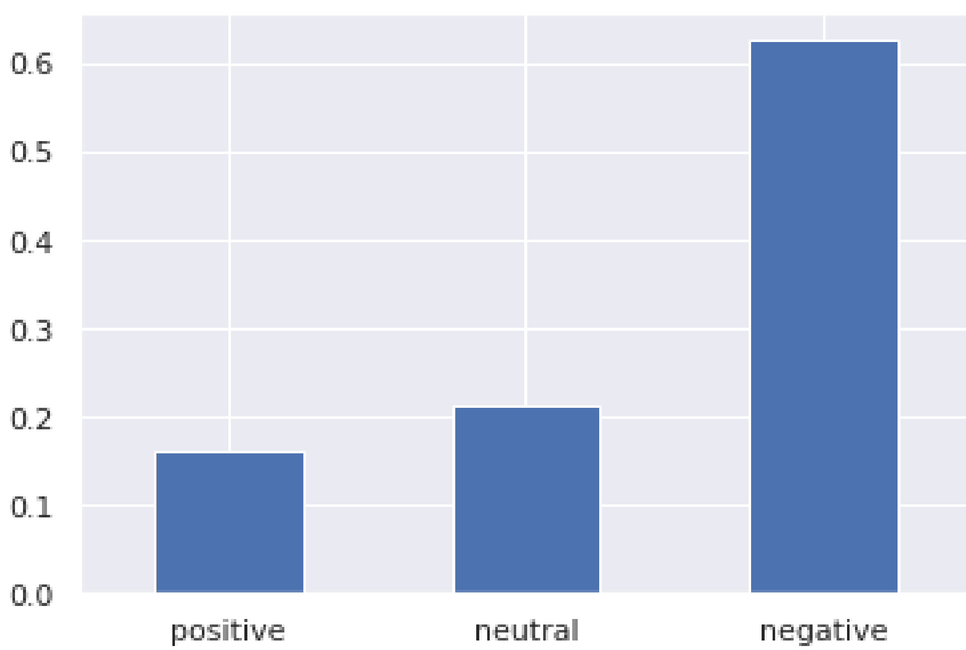
[5]:
```
sns.countplot(df['airline_sentiment'], order=['positive', 'neutral',␣
 ↪'negative'])
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f211789a3d0>
```

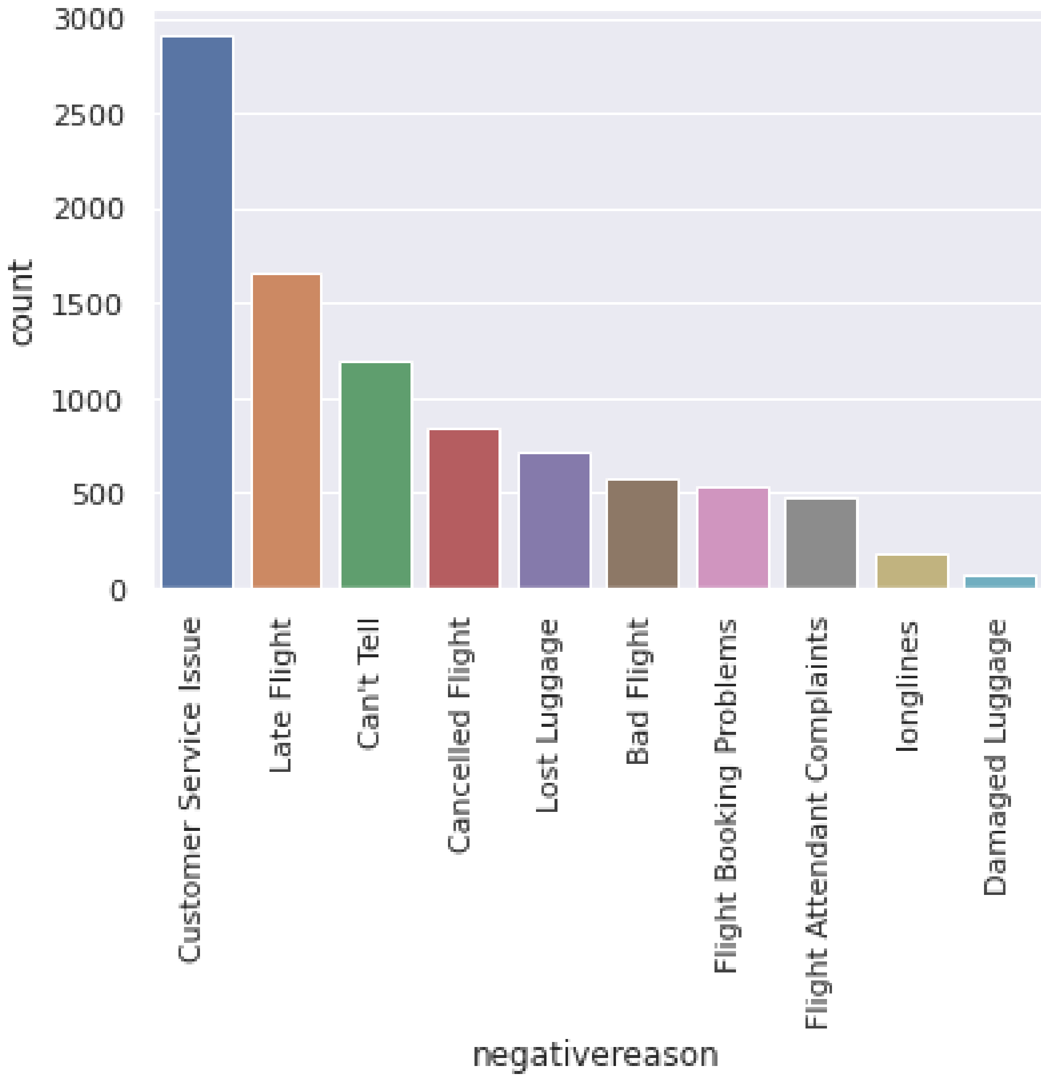


[6]:
```
df['airline_sentiment'].value_counts(normalize=True, ascending=True).
↪plot(kind='bar', rot=0)
```

[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f211672cc40>



[7]:
```
sns.countplot(df['negativereason'], order=df['negativereason'].value_counts().
 ↪index)
plt.xticks(rotation=90);
```



2.3 Preprocess data

[8]:
```
twitter_handle_pattern = r'@(\w+)'
hashtag_pattern = r'(?:^|\s)[＃#]{1}(\w+)'
url_pattern = r'https?:\/\/.*.com'
```

2.3.1 Challenge

1. Create a function called `clean_tweet()` that takes in a tweet text; replaces hashtags, users, and URLs; and returns the result.
2. Test it out on the `my_tweets` list object just defined: the output should be the same as that of the string methods we just used!
3. Apply the `clean_tweet()` function to the tweets dataframe above to create a new `clean_text` column.

Hint: Use the handy `re.sub()` method for pattern replacement and `df[col].apply()` for applying a function to a specific DataFrame column.

[9]:
```python
# solution
def clean_tweet(tweet):
    tweet = re.sub(hashtag_pattern, ' HASHTAG', tweet)
    tweet = re.sub(twitter_handle_pattern, 'USER', tweet)
    return re.sub(url_pattern, 'URL', tweet)

# apply function to DF
df['clean_text'] = (df['text'].apply(clean_tweet))
df.head(3)
```

[9]:
```
             tweet_id airline_sentiment  airline_sentiment_confidence  \
0  570306133677760513           neutral                        1.0000
1  570301130888122368          positive                        0.3486
2  570301083672813571           neutral                        0.6837

  negativereason  negativereason_confidence         airline  \
0            NaN                        NaN  Virgin America
1            NaN                        0.0  Virgin America
2            NaN                        NaN  Virgin America

  airline_sentiment_gold        name negativereason_gold  retweet_count  \
0                    NaN     cairdin                 NaN              0
1                    NaN    jnardino                 NaN              0
2                    NaN  yvonnalynn                 NaN              0

                                                text tweet_coord  \
0                 @VirginAmerica What @dhepburn said.         NaN
1  @VirginAmerica plus you've added commercials t…         NaN
2  @VirginAmerica I didn't today… Must mean I n…         NaN

               tweet_created tweet_location               user_timezone  \
0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US & Canada)
1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US & Canada)
2  2015-02-24 11:15:48 -0800      Lets Play  Central Time (US & Canada)

                                          clean_text
```

```
0                                 USER What USER said.
1  USER plus you've added commercials to the expe…
2  USER I didn't today… Must mean I need to tak…
```

2.4 Vectorization

[10]:
```
countvectorizer = CountVectorizer(max_features=5000, binary=True)
X = countvectorizer.fit_transform(df['clean_text'])
features = X.toarray() # convert matrix to sparse format for easy modeling
response = df['airline_sentiment'].values # corresponds to entries in␣
↪`features`
feature_names = countvectorizer.get_feature_names() # get features for later
```

2.5 Divide data into training and test sets

[11]:
```
X_train, X_test, y_train, y_test = train_test_split(features, response,␣
↪test_size=0.2)

X_train.shape, X_test.shape #look at number of rows and columns in training and␣
↪test data
```

[11]:
```
((11712, 5000), (2928, 5000))
```

3 More classification with supervised machine learning

3.1 Train decision tree model

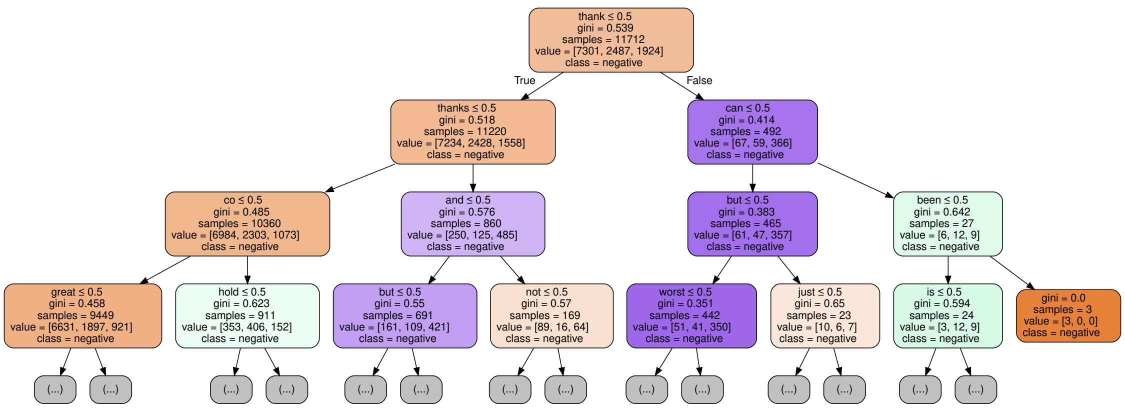
[12]:
```
dt_classifier = DecisionTreeClassifier(criterion='gini',  # or 'entropy' for␣
↪information gain
                                       splitter='best',  # or 'random' for␣
↪random best split
                                       max_depth=None,  # how deep tree nodes␣
↪can go
                                       min_samples_split=2,  # samples needed␣
↪to split node
                                       min_samples_leaf=1,  # samples needed␣
↪for a leaf
                                       min_weight_fraction_leaf=0.0,  # weight␣
↪of samples needed for a node
                                       max_features=None,  # number of features␣
↪to look for when splitting
                                       max_leaf_nodes=None,  # max nodes
                                       min_impurity_decrease=1e-07, #early␣
↪stopping
                                       random_state = 10) #random seed
```

```python
dt_classifier.fit(X_train, y_train) # fit model on training set
```

[12]: DecisionTreeClassifier(min_impurity_decrease=1e-07, random_state=10)

[13]:
```python
dt_data = export_graphviz(dt_classifier,
                          out_file=None,
                          feature_names=feature_names,
                          class_names=y_train,
                          max_depth=3,
                          filled=True,
                          rounded=True,
                          special_characters=True)
graphviz.Source(dt_data)
```

[13]:



[14]:
```python
#predict the labels on the test set using the trained model
predictions_dt = dt_classifier.predict(X_test)

accuracy_score(predictions_dt, y_test)
```

[14]: 0.6953551912568307

[15]:
```python
# We can also use the built-in `model.score()` method instead this time,␣
↪results are same as for `accuracy_score()`
print(dt_classifier.score(X_test, y_test))
```

0.6953551912568307

3.1.1 Challenge

Part 1

Train a logistic regression with cross-validation (LogisticRegressionCV()) classifier on the training set and evaluate its accuracy on the test set. To avoid a long delay, set cv to 3, solver to `liblinear', and penalty to `l1'. Examine the other model parameters.

[16]:
```python
# solution
# Initialize model
logitcv = LogisticRegressionCV(Cs=5,
                               penalty='l1',
                               cv=3,
                               solver='liblinear',
                               refit=True)

logitcv.fit(X_train, y_train) # Train on training set
predictions_logitcv = logitcv.predict(X_test) # Make predictions
accuracy_score(predictions_logitcv, y_test) # Evaluate accuracy
```

[16]: 0.787568306010929

Part 2

Use the model you just built to predict the label for three unlabeled reviews. Predict both the label and the probability of being in each category. Do the predictions make sense? Do the probabilities provide any useful information beyond the predicted labels?

Hint: Remember to use countvectorizer() and convert to sparse format with .toarray() to vectorize the texts prior to prediction. Use model.predict() to predict the class and model.predict_proba() to predict class probabilities.

[17]:
```python
careless_review = "I made my flight reservation with UnitedAirlines and the␣
 ↪service is ridiculous and they just bluntly \
deny solving your problem. I booked my flight but they were having some␣
 ↪technical issues and didn't appear \
in their system. United customer service just said they can not do anything and␣
 ↪asked me to book a new ticket to fly. \
This is just insane and troubling and also really costly at the airport. Thus,␣
 ↪I made my flight reservation \
with United Airlines for the next day through an online flight booking portal␣
 ↪that offered much cheaper flights. \
You can contact them over the phone by calling a toll-free number␣
 ↪+1-888-720-1433 and book flights that are light \
on your pocket."

good_review = "United was very accommodating for our vacation recently and even␣
 ↪got us home a little quicker. \
We were travelling in a group of 12 coming home from a vacation and because of␣
 ↪storms back east our connecting flight \
was delayed by hours. Knowing this was completely out of their control we␣
 ↪started talking with one of the United \
```

```
employees and told them where we were headed. She said she could get us on a␣
↪flight that actually leaves earlier \
than our original flight. It was such great service and resulted in all of us␣
↪getting home hours later. The only \
issue was our bags, but United had them delivered to our homes that next␣
↪morning."

canceled_review = "Canceled my trip twice, the first time I received no␣
↪notification. The second time I got a notice \
that one of my trips may have been canceled or changed, the website didn't say␣
↪which it was. It instructed me to cancel \
it myself. Probably so they could avoid having to refund me. Now I all the␣
↪money I spent on tickets sitting in a United \
account but I never want to fly with them in the future so that's a nice chunk␣
↪of about $2000 wasted. \
EDIT: After I cancelled my second trip I was able to select an option for␣
↪refund, the refund was not only \
for the second trip but also my first trip that was cancelled earlier. I will␣
↪upgrade the rating from 1 to 3 stars \
would have given higher with better customer service/website service/website␣
↪information."
```

[18]:
```
# solution
# Transform these into DTMs with the same feature-columns as previously
unknown_dtm = countvectorizer.transform([careless_review, good_review,␣
↪canceled_review]).toarray()

# Use model to predict the class for these three. Predict class
print("Predicted classes:")
print(logitcv.predict(unknown_dtm))
print()
print(f"Predicted probabilities for each review\n(order: {logitcv.classes_}):")
print(logitcv.predict_proba(unknown_dtm)) # Predict probability of membership␣
↪in each category
```

```
Predicted classes:
['negative' 'negative' 'negative']

Predicted probabilities for each review
(order: ['negative' 'neutral' 'positive']):
[[9.99950920e-01 1.91445398e-07 4.88888149e-05]
 [9.89676333e-01 4.84768860e-08 1.03236183e-02]
 [9.99999226e-01 4.85944713e-07 2.87678353e-07]]
```

3.2 Cross-validation

[19]:
```
scores = cross_val_score(dt_classifier, X_train, y_train, cv=5)
scores
```

[19]:
```
array([0.68501921, 0.69654289, 0.69470538, 0.67250213, 0.67805295])
```

[20]:
```
# Show average accuracy across folds along with sigma (std. squared)
print("Accuracy: %0.4f (+/- %0.4f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.6854 (+/- 0.0186)
```

3.2.1 Challenge

Why is the cross validation accuracy different than our classifier? Which estimate is more accurate?

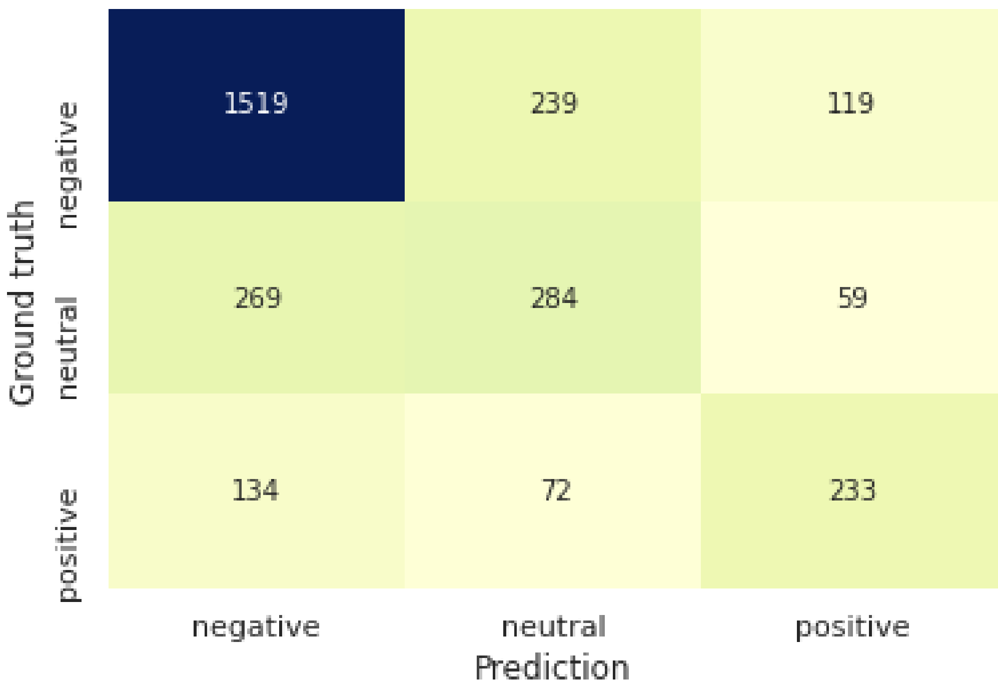
This assumes any unseen data has similar features as your training and test data. If the features are different, you can not assume you will get similar accuracy. When might the unseen data be different from the labeled data?

3.3 More model evaluation

[21]:
```
model = dt_classifier # change this one line to visualize confusion matrix for␣
 ↪other models
y_predict = model.predict(X_test)
labels = model.classes_
conmat = confusion_matrix(y_test, y_predict)

sns.heatmap(conmat,
            annot=True,
            fmt='d',
            xticklabels=labels,
            yticklabels=labels,
            cmap="YlGnBu",
            cbar=False)
plt.ylabel('Ground truth')
plt.xlabel('Prediction')
```

[21]:
```
Text(0.5, 12.5, 'Prediction')
```



[22]:
```
dt_predicted = dt_classifier.predict(X_test)
print("Classification report:")
print(classification_report(y_test, dt_predicted))
```

```
Classification report:
              precision    recall  f1-score   support

    negative       0.79      0.81      0.80      1877
     neutral       0.48      0.46      0.47       612
    positive       0.57      0.53      0.55       439

    accuracy                           0.70      2928
   macro avg       0.61      0.60      0.61      2928
weighted avg       0.69      0.70      0.69      2928
```

3.4 Optimize parameters with grid search

[22]:
```
# Grid searching is computationally expensive, so for time's sake we'll test a␣
↪minimal range for each parameter
# In real-world data analytics there would be a much larger range, making␣
↪exponentially more permutations (and time cost)
```

```
param_grid = {'min_samples_split': [2,10], # define dictionary with possible␣
 ↪model parameters
              'min_samples_leaf': [2,10]}

param_grid
```

[22]:
```
{'min_samples_split': [2, 10], 'min_samples_leaf': [2, 10]}
```

[23]:
```
# Warning: This can take a long time!
model_dt = GridSearchCV(dt_classifier,
                        param_grid,
                        cv=3,
                        return_train_score=True)

model_dt.fit(X_train, y_train)
```

[23]:
```
GridSearchCV(cv=3,
             estimator=DecisionTreeClassifier(min_impurity_decrease=1e-07,
                                              random_state=10),
             param_grid={'min_samples_leaf': [2, 10],
                         'min_samples_split': [2, 10]},
             return_train_score=True)
```

[24]:
```
# Get information on model parameters that perform best
model = model_dt # Change this to visualize other models

# Get index of best model parameters
best_index = np.argmax(
    model.cv_results_["mean_train_score"]
                      )

print('Best parameter values are:',
      model.cv_results_["params"][best_index]
     )
print('Best mean cross-validation train accuracy: %.03f' %
      model.cv_results_["mean_train_score"][best_index]
     )
print('Overall mean test accuracy: %.03f' %
      model.score(X_test, y_test)
     )
```

```
Best parameter values are: {'min_samples_leaf': 2, 'min_samples_split': 2}
Best mean cross-validation train accuracy: 0.893
Overall mean test accuracy: 0.734
```

[25]:
```
# Prepare for visualization: get combinations on tested model parameters
n_grid_points = len(model_dt.cv_results_['params'])
```

```python
min_samples_leaf_vals = np.empty((n_grid_points,))
min_samples_split_vals = np.empty((n_grid_points,))
mean_train_scores = np.empty((n_grid_points,))
mean_test_scores = np.empty((n_grid_points,))

for i in range(n_grid_points):
    min_samples_leaf_vals[i] = model.
 ↪cv_results_['params'][i]['min_samples_leaf']
    min_samples_split_vals[i] = model.
 ↪cv_results_['params'][i]['min_samples_split']
    mean_train_scores[i] = model.cv_results_['mean_train_score'][i]
    mean_test_scores[i] = model.cv_results_['mean_test_score'][i]
```

[26]:
```python
def gridsearch_viz(mean_scores,
                   param1_vals,
                   param2_vals,
                   param1_name,
                   param2_name,
                   scorename = "Mean Model Accuracy on Training Set"):

    """
    Build 3-d visualization to show model parameter combinations and their␣
 ↪resulting scores.
    Takes in only two parameters with any number of candidate settings thereof.

    Args:
        mean_scores: List of model scores, as long as list of model parameters␣
 ↪being tested.
        param1_vals: list of values for first parameter, as long as list of␣
 ↪model parameters being tested.
        param2_vals: list of values for second parameter, as long as list of␣
 ↪model parameters being tested.
        param1_name: str indicating name of first parameter being tested
        param2_name: str indicating name of second parameter being tested
        scorename: str indicating name of scores being displayed (train or␣
 ↪test).

    Output:
        3-d visualization of model params & performance
    """

    fig = plt.figure(figsize=(20,10))
    ax = fig.gca(projection='3d')

    surf = ax.plot_trisurf(param1_vals,
                           param2_vals,
```

```
                            mean_scores,
                            cmap=cm.coolwarm,
                            linewidth=10,
                            antialiased=False)

    ax.set_title(scorename, fontsize=18)
    ax.set_xlabel(param1_name, fontsize=18)
    ax.set_ylabel(param2_name, fontsize=18)
```

[27]:
```
# Visualize model performance on training set
gridsearch_viz(mean_scores = mean_train_scores,
               param1_vals = min_samples_leaf_vals,
               param2_vals = min_samples_split_vals,
               param1_name = "min_samples_split",
               param2_name = "min_samples_leaf")
```

3.5 Train and optimize Random Forest model

[28]:
```
rf_classifier = RandomForestClassifier(n_estimators=10,  # number of trees
                       criterion='gini',  # or 'entropy' for information gain
                       max_depth=None,  # how deep tree nodes can go
                       min_samples_split=2,  # samples needed to split node
                       min_samples_leaf=1,  # samples needed for a leaf
                       min_weight_fraction_leaf=0.0,  # weight of samples␣
 ↪needed for a node
                       max_features='auto',  # number of features for best split
                       max_leaf_nodes=None,  # max nodes
                       min_impurity_decrease=1e-07,  # early stopping
```

```
                             n_jobs=1,  # CPUs to use
                             random_state = 10,  # random seed
                             class_weight="balanced")  # adjusts weights inverse of␣
 ↪freq, also "balanced_subsample" or None

rf_model = rf_classifier.fit(X_train, y_train) # fit model on training data
```

[29]:
```
print("Accuracy of Random Forest model (with 3-fold CV) with test data defined␣
 ↪above:")
scores = cross_val_score(rf_model, X_test, y_test, cv=3)
print("%0.4f (+/- %0.4f)" % (scores.mean(), scores.std() * 2)) # Show average␣
 ↪accuracy across folds
print()

predicted = rf_model.predict(X_test)
print("Classification report:")
print(classification_report(y_test, predicted))
print()
```

```
Accuracy of Random Forest model (with 3-fold CV) with test data defined above:
0.7162 (+/- 0.0398)

Classification report:
              precision    recall  f1-score   support

    negative       0.78      0.92      0.84      1881
     neutral       0.61      0.41      0.49       591
    positive       0.76      0.49      0.59       456

    accuracy                           0.75      2928
   macro avg       0.72      0.61      0.64      2928
weighted avg       0.74      0.75      0.73      2928
```

3.5.1 Challenge: Grid Search on Random Forest

Do another grid search to determine the best parameters for the Random Forest we just created. Use two possible levels for min_samples_split and min_samples_leaf, each between 1 and 10.

[30]:
```
# solution
param_grid = {'min_samples_split': [2,10],
              'min_samples_leaf': [2,10]}

rf_model = GridSearchCV(RandomForestClassifier(n_estimators=10,
                                               min_impurity_decrease=1e-07,  #␣
 ↪early stopping
```

```
                                              random_state = 10,  # random seed
                                              ␣
↪class_weight="balanced_subsample"), # adjusts weights inverse of freq with␣
↪subsampling
                       param_grid,
                       cv=3,
                       return_train_score=True)

rf_model.fit(X_train, y_train)
```

[30]:
```
GridSearchCV(cv=3,
             estimator=RandomForestClassifier(class_weight='balanced_subsample',
                                              min_impurity_decrease=1e-07,
                                              n_estimators=10,
                                              random_state=10),
             param_grid={'min_samples_leaf': [2, 10],
                         'min_samples_split': [2, 10]},
             return_train_score=True)
```

[31]:
```
best_index = np.argmax(rf_model.cv_results_["mean_train_score"])

print("Best parameter values:", rf_model.cv_results_["params"][best_index])
print("Best mean cross-validated training accuracy:", rf_model.
 ↪cv_results_["mean_train_score"][best_index])
print("Overall mean test accuracy:", rf_model.score(X_test, y_test))
```

```
Best parameter values: {'min_samples_leaf': 2, 'min_samples_split': 2}
Best mean cross-validated training accuracy: 0.8347848360655737
Overall mean test accuracy: 0.7377049180327869
```

3.5.2 Extra challenge: Adjust text preprocessing

Preprocessing methods matter for machine learning performance: Depending on the algorithm, less or more preprocessing may be better. Let's see how a simple Random Forest model does with minimal preprocessing--without removing usernames, hashtags, or URLs. Compare this model's performance with that of Random Forest trained on cleaned text.

[32]:
```
# solution
# Vectorize texts
countvec_dirty = CountVectorizer(max_features=5000, binary=True)
features_dirty = countvec_dirty.fit_transform(df['text']).toarray() # convert␣
 ↪matrix to sparse format for easy modeling
response_dirty = df['airline_sentiment'].values # corresponds to entries in␣
 ↪`features`

X_train_dirty, X_test_dirty, y_train_dirty, y_test_dirty =␣
 ↪train_test_split(features_dirty, response_dirty, test_size=0.2)
```

```
X_train_dirty.shape, X_test_dirty.shape #look at number of rows and columns in␣
↪training and test data
```

[32]: ((11712, 5000), (2928, 5000))

[33]:
```
rf_dirty = RandomForestClassifier(n_estimators=10,  # number of trees
                                  min_impurity_decrease=1e-07,  # early stopping
                                  random_state = 10,  # random seed
                                  class_weight="balanced")  # adjusts weights␣
↪inverse of freq

rf_dirty = rf_dirty.fit(X_train_dirty, y_train_dirty)
```

[34]:
```
scores = cross_val_score(rf_dirty, X_train_dirty, y_train_dirty, cv=5)
print("Mean cross-validation train accuracy: %0.4f (+/- %0.4f)" % (scores.
↪mean(), scores.std() * 2)) # Show average accuracy across folds
print("Overall mean test accuracy:", rf_dirty.score(X_test_dirty, y_test_dirty))
```

```
Mean cross-validation train accuracy: 0.7352 (+/- 0.0104)
Overall mean test accuracy: 0.735655737704918
```

Random Forest with text preprocessing gave a mean accuracy of just over 0.7, so removing text preprocessing improves model accuracy by **3 or 4 percent**!

3.5.3 Extra extra challenge: Adaboost

Adaboost is another ensemble method that relies on `boosting'. Similar to `bagging', `boosting' samples many subsets of data to fit multiple classifiers, but resamples preferentially for misclassified data points.

Part 1

Using the scikit-learn documentation, build your own AdaBoost model to test on our review tweets! Start off with n_estimators at 100, and learning_rate at .5. Use 10 as the random_state value.

[35]:
```
# solution
adaboost_model = AdaBoostClassifier(n_estimators=100,  # number of trees
                                    learning_rate=0.5,  # or 'entropy' for␣
↪information gain
                                    random_state = 10)  # random seed

adaboost_model = adaboost_model.fit(X_train, y_train)
```

[36]:
```
print("Score of Adaboost model with test data defined above:")
print(adaboost_model.score(X_test, y_test))
print()

predicted = adaboost_model.predict(X_test)
```

```
print("Classification report:")
print(classification_report(y_test, predicted))
print()
```

```
Score of Adaboost model with test data defined above:
0.7482923497267759

Classification report:
              precision    recall  f1-score   support

    negative       0.75      0.94      0.84      1881
     neutral       0.68      0.27      0.39       591
    positive       0.75      0.57      0.65       456

    accuracy                           0.75      2928
   macro avg       0.73      0.59      0.63      2928
weighted avg       0.74      0.75      0.72      2928
```

Part 2

Now use a grid search to determine what are the best values for the n_estimators and learning_rate parameters. For n_estimators try both 100 and 500, and for learning_rate try 0.1 and 1.0.

[37]:
```
# solution
param_grid = {'n_estimators': [100,500],
              'learning_rate': [0.1,1.0]}

adaboost_model = GridSearchCV(AdaBoostClassifier(n_estimators=10),
                              param_grid,
                              cv=3,
                              return_train_score=True)
adaboost_model.fit(X_train, y_train)

# Get index on model parameters that perform best
best_index = np.argmax(adaboost_model.cv_results_["mean_train_score"]) # Get␣
↪index of best model parameters

# Display information on best model parameters
print("Best parameter values:", adaboost_model.
↪cv_results_["params"][best_index])
print("Best mean cross-validated train accuracy:", adaboost_model.
↪cv_results_["mean_train_score"][best_index])
print("Overall mean test accuracy:", adaboost_model.score(X_test, y_test))
```

```
Best parameter values: {'learning_rate': 1.0, 'n_estimators': 500}
```

Best mean cross-validated train accuracy: 0.8078466530054644
Overall mean test accuracy: 0.7817622950819673

Part 3

Build a 3-d visualization showing combinations of model parameters and their resulting performance on the training set.

Hint: Use the gridsearch_viz() function we defined earlier.

[38]:
```
# solution
model = adaboost_model # Change this to visualize other models

# Prepare for visualization: get combinations on tested model parameters
n_grid_points = len(model.cv_results_['params'])
n_estimators_vals = np.empty((n_grid_points,))
learning_rate_vals = np.empty((n_grid_points,))
mean_train_scores = np.empty((n_grid_points,))
mean_test_scores = np.empty((n_grid_points,))

for i in range(n_grid_points):
    n_estimators_vals[i] = model.cv_results_['params'][i]['n_estimators']
    learning_rate_vals[i] = model.cv_results_['params'][i]['learning_rate']
    mean_train_scores[i] = model.cv_results_['mean_train_score'][i]
    mean_test_scores[i] = model.cv_results_['mean_test_score'][i]
```

[39]:
```
# Visualize model performance on training set
gridsearch_viz(mean_scores = mean_train_scores,
               param1_vals = n_estimators_vals,
               param2_vals = learning_rate_vals,
               param1_name = "n_estimators",
               param2_name = "learning_rate",
               scorename = "Mean Train Scores")
```