3-Model development

August 31, 2022

1 Custom modules

[3]:

```python
import os
import sys
# lib_src = os.path.join(os.getcwd(), os.pardir, 'src')
lib_src = '../../machine-learning'
sys.path.insert(1, lib_src)
```

[4]:

```python
from dummies import get_dummies_indices
from metrics.log_loss import multi_multi_log_loss
from preprocessing.combine_text_columns import combine_text_columns
from model_selection.multilabel import multilabel_sample_dataframe,␣
  ↪multilabel_train_test_split
from size import size
from to_csv_to_zip import to_csv_to_zip
from fit_cache import fit_cache
```

2 Standard modules

[5]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('dark_background')
import seaborn as sns
import pickle
from time import time
from datetime import datetime
import warnings
from zipfile import ZipFile, ZIP_DEFLATED
import psutil
from tempfile import mkdtemp
from shutil import rmtree
```

3 Load data, randomize, optimize category type and get dummy labels

[7]:

```
prediction_dir = '/data/drivendata/predictions/'
model_dir = '/data/drivendata/models/'
df = pd.read_csv('/data/drivendata/TrainingData.csv', index_col=0)
df = df.sample(frac=1, random_state=1)
LABELS = ['Function', 'Object_Type', 'Operating_Status', 'Position_Type',␣
 ↪'Pre_K', 'Reporting', 'Sharing', 'Student_Type', 'Use']
FEATURES = [feature for feature in df.columns if feature not in LABELS]
NUMERIC_FEATURES = ['FTE', 'Total']
TEXT_FEATURES = [text_feature for text_feature in FEATURES if text_feature not␣
 ↪in NUMERIC_FEATURES]
df[LABELS] = df[LABELS].apply(lambda x: x.astype('category'), axis=0)
y = pd.get_dummies(df[LABELS], prefix_sep='__')
holdout = pd.read_csv('/data/drivendata/TestData.csv', index_col=0,␣
 ↪dtype={'Facility_or_Department':'object', 'Text_4':'object'})
```

4 Which feature alignment?

[6]:

```
# holdout = holdout[FEATURES]
FEATURES = holdout.columns.values
```

5 Model metric: multi-multi log loss

[5]:

```
from sklearn.metrics import make_scorer
cci = get_dummies_indices(df[LABELS])
multi_multi_log_loss_scorer = make_scorer(multi_multi_log_loss,␣
 ↪greater_is_better=False, needs_proba=True,
                                          class_column_indices=cci)
```

6 Candidate classifiers

[8]:

```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
# from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
```

7 Baseline pipeline

[7]:

```
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import chi2, SelectKBest
from sklearn.preprocessing import FunctionTransformer, MaxAbsScaler,␣
 ↪PolynomialFeatures
```

[8]:

```
from preprocessing.get_normalized_total import get_normalized_total
get_text_data = FunctionTransformer(combine_text_columns, validate=False,
                                    kw_args = {'to_drop': NUMERIC_FEATURES +␣
 ↪LABELS})
get_numeric_data = FunctionTransformer(get_normalized_total, validate=False,
                                       kw_args = {'reference':'FTE',␣
 ↪'ambiguous':'Total'})
```

[9]:

```
pl = Pipeline([
    ('union', FeatureUnion([
        ('numeric_features', Pipeline([
            ('numeric_selector', get_numeric_data),
            ('imputer', SimpleImputer(strategy = 'constant', fill_value = 0))
        ], verbose=True)),
        ('text_features' , Pipeline([
            ('text_selector', get_text_data),
            ('vectorizer', HashingVectorizer(norm = None, binary = False,␣
 ↪alternate_sign = False, token_pattern='(?u)\\b\\w+\\b', dtype='float32')),
            ('reducer', SelectKBest(score_func = chi2))
        ], verbose=True))
    ])),
    ('interactor', PolynomialFeatures(degree=2, include_bias=False,␣
 ↪interaction_only=True)),
    ('scaler', MaxAbsScaler()),
    ('classifier', OneVsRestClassifier(LogisticRegression(random_state=1,␣
 ↪solver='liblinear')))
], verbose=True)
```

8 Learning and validation curves

[10]:

```
from model_selection.learning_curve import plot_learning_curve
from model_selection.validation_curve import plot_validation_curve
from sklearn.model_selection import ParameterGrid
from sklearn.feature_extraction.text import CountVectorizer

train_sizes = np.linspace(0.2, 1, 5)
```

The learning curves help comparing algorithms for bias-variance behavior, choosing model parameters during design, adjusting optimization to improve convergence and determining the amount of data used for training.

In this pipeline, the amount of data is determined not only by the sample size, but also by: * the feature generation step `HashingVectorizer` with its parameter `ngram_range`. * the feature selection step `SelectKBest(chi2)` with its parameter `k`. * the feature generation step `PolynomialFeatures` with its parameters fixed.

Initially, it's picked main classification algorithms that support specific characteristics of the pipeline: * Sparse data produced by the `HashingVectorizer` * Negative numbers present in the numeric features `Total`and `FTE`.

Because of this characteristics, alghorithms such as Naive Bayes can't be used: * `GaussianNB` requires dense data * `MultinomialNB` requires positive numbers.

Other alghorithms were much slow hence it's limited to `LogisticRegression`. This limitation is expected to be removed in a future version.

The train sizes is a sligthly variation of the default `np.linspace(0.1, 1.0, 5)` to keep evenly spaced intervals.

9 1% data sample

[11]:
```
sampling = multilabel_sample_dataframe(df, y, size=0.01, min_count=7, seed=1)
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
print('Sample size:', sampling.shape[0])                                  #␣
 ↪4002
print('Train sizes:', (sampling.shape[0] * train_sizes * .8).astype('int')) #␣
 ↪[640 1280 1920 2561 3201]
print('Test sizes :', (sampling.shape[0] * train_sizes * .2).astype('int')) #␣
 ↪[160 320 480 640 800]
```

```
Sample size: 4002
Train sizes: [ 640 1280 1920 2561 3201]
Test sizes : [160 320 480 640 800]
```

9.1 Learning curves

[12]:
```
classifiers = [#('Dummy (uniform)',␣
 ↪OneVsRestClassifier(DummyClassifier(strategy='uniform'))),
               ('Logistic regression',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))#,
               #('SVC', OneVsRestClassifier(SVC(probability=True))),
               #('xgboost', OneVsRestClassifier(XGBClassifier())),
               #('GBM', OneVsRestClassifier(GradientBoostingClassifier())),  #␣
 ↪mostly slower/worse thanxgboost
               #('Random Forest',␣
 ↪OneVsRestClassifier(RandomForestClassifier())),
                 #('K-Neighbors', OneVsRestClassifier(KNeighborsClassifier())),
               ]
```

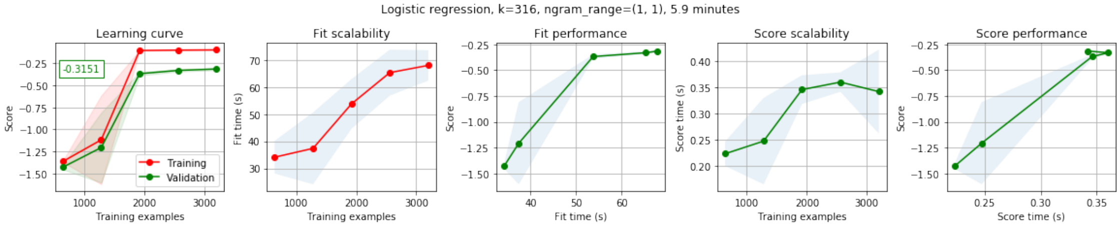
Finding thek parameter space limits for each `ngram_range`

```
[13]: ngram_kmax = [len(CountVectorizer(ngram_range=(1,n),
                                      dtype='uint8').
       ↪fit(combine_text_columns(sampling)).vocabulary_)
                    for n in range(1, 4)]  # [1546, 8703, 19981]
```

9.1.1 1-gram features

```
[18]: ngram1_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[0])), 7).round().
       ↪astype('int')
      ngram1_logspace = np.hstack((ngram1_logspace[ngram1_logspace < ngram_kmax[0]],␣
       ↪ngram_kmax[0]))
      parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
       ↪[(1,1)],
                                   'union__text_features__reducer__k':␣
       ↪ngram1_logspace}]) # [316, 562, 1000, 1546]
      print(datetime.now().isoformat(timespec='minutes'))
      for title, classifier in classifiers:
          pl.set_params(classifier = classifier)
          pl.set_params(classifier__n_jobs = None)
          for parameter in parameters:
              pl.set_params(**parameter)
              plot_learning_curve(pl, ', '.join([title] + [k.
       ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                                  sampling[FEATURES], dummy_labels, cv=5,␣
       ↪scoring=multi_multi_log_loss_scorer,
                                  n_jobs=-1, #verbose=11,
                                  train_sizes=train_sizes)
              plt.show()
```

```
2020-07-01T04:38
```


Logistic regression, k=316, ngram_range=(1, 1), 5.9 minutes

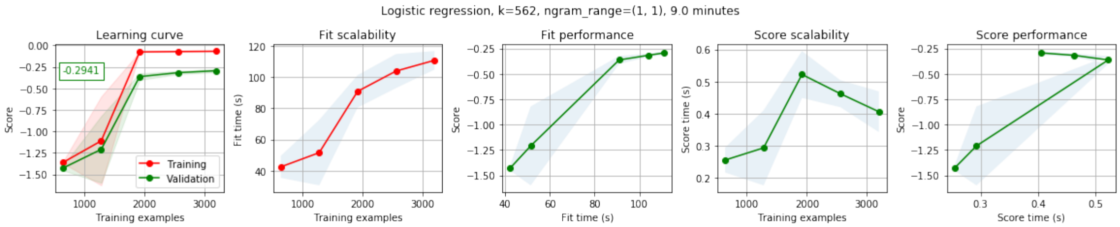
Learning curve

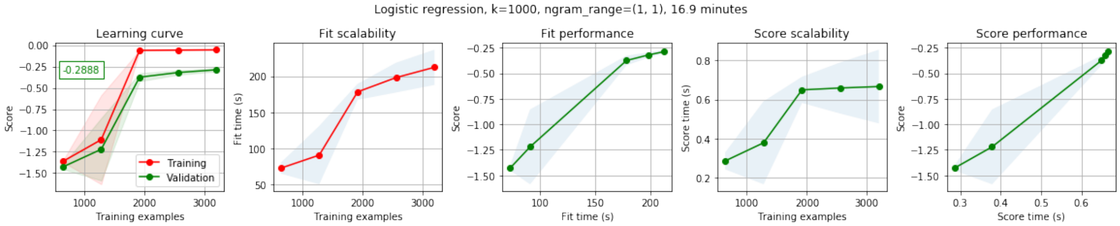
Fit scalability

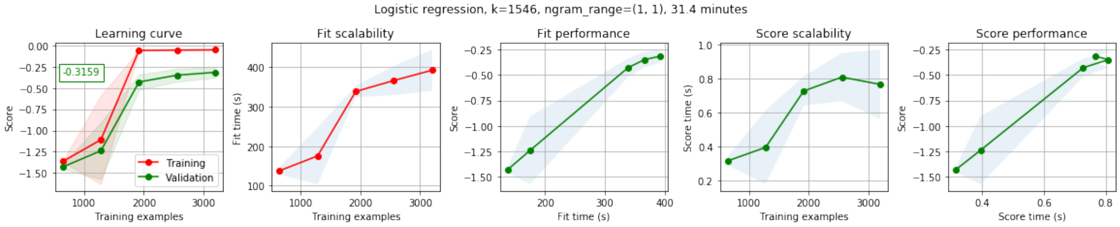
Fit performance

Score scalability

Score performance


Logistic regression, k=562, ngram_range=(1, 1), 9.0 minutes


Logistic regression, k=1000, ngram_range=(1, 1), 16.9 minutes


Logistic regression, k=1546, ngram_range=(1, 1), 31.4 minutes

9.1.2 2-gram features

With 2-gram features it was only possible to run without parallel processing in `plot_learning_curve(n_jobs=None)` but still possible to do some parallel processing in the pipelines's classifier step `OneVsRestClassifier(n_jobs=2)` but not possible to parallelize when `k=5623`. 2-gram features predict worse than 1-gram features.

The intended `k` parameter space:

[19]:
```python
ngram2_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[1])), 7).round().astype('int')
ngram2_logspace = np.hstack((ngram2_logspace[ngram2_logspace < ngram_kmax[1]], ngram_kmax[1]))
parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range': [(1,2)],
                             'union__text_features__reducer__k': [316, 562, 1000, 1778, 3162]}]) # ngram2_logspace}])
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    pl.set_params(classifier__n_jobs = None)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
 ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            sampling[FEATURES], dummy_labels, cv=5,␣
 ↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=2, verbose=11, train_sizes=train_sizes)
        plt.show()
```

```
[19]: array([ 316,  562, 1000, 1778, 3162, 5623, 8703])
```

In order advance to end of the `k` parameter space, the training was run without any parallelism (`n_jobs=None`) for the `plot_learning_curve()` and the pipelines's classifier step `OneVsRestClassifier()`.

```
[ ]: parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
 ↪[(1,2)],
                              'union__text_features__reducer__k': [5623]}]) #␣
 ↪ngram2_logspace}])
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    pl.set_params(classifier__n_jobs = None)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
 ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            sampling[FEATURES], dummy_labels, cv=5,␣
 ↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=None, verbose=11, train_sizes=train_sizes)
        plt.show()
```

Even without the parallelism, the limit of the `k` parameter space (8703) wasn't reached because the training ran out of memory.

```
[ ]: parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
 ↪[(1,2)],
                              'union__text_features__reducer__k': [8703]}]) #␣
 ↪ngram2_logspace}])
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    pl.set_params(classifier__n_jobs = None)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
  ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            sampling[FEATURES], dummy_labels, cv=5,␣
  ↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=None, verbose=11, train_sizes=train_sizes)
        plt.show()
```

9.1.3 3-gram features

Because of the results of 2-gram features compared to 1-gram features, it wasn't expected better results with 3-gram features which also suffers from lack of memory.

[ ]:

```
ngram3_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[2])), 11).
  ↪round().astype('int')
ngram3_logspace = np.hstack((ngram3_logspace[ngram3_logspace < ngram_kmax[2]],␣
  ↪ngram_kmax[2]))
parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
  ↪[(1,3)],
                             'union__text_features__reducer__k':␣
  ↪ngram3_logspace}])  #
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    pl.set_params(classifier__n_jobs = None)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
  ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            sampling[FEATURES], dummy_labels, cv=5,␣
  ↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=None, verbose=11, train_sizes=train_sizes)
        plt.show()
```

9.1.4 Metrics summary

sample	min. classes	jobs	ngram	k	interactions	logloss	time	mem. peak
0.01	7	4	(1,1)	316	51681	0.3151	5.9 min	2.2 GiB
0.01	7	4	(1,1)	562	159330	0.2941	9.1 min	2.2 GiB
0.01	7	4	**(1,1)**	**1000**	**505515**	**0.2888**	16.9 min	2.3 GiB
0.01	7	4	(1,1)	1546	1203576	0.3159	31.1 min	2.3 GiB
0.01	7	4	(1,2)	316	51681	0.3880	6.4 min	2.3 GiB
0.01	7	4	(1,2)	562	159330	0.3438	14.7 min	2.2 GiB
0.01	7	4	(1,2)	1000	505515	0.3210	33.6 min	2.2 GiB
0.01	7	4	(1,2)	1778	1590436	0.3139	102.4 min	2.3 GiB
0.01	7	4	(1,2)	3162	5016528	0.3098	228.6 min	4.6 GiB

sample	min. classes	jobs	ngram	k	interactions	logloss	time	mem. peak
0.01	7	1	(1,2)	5623	15840006	0.3250	589.6 min	14 GiB
0.01	7	1	(1,2)	8703	37918986			run out
0.01	7	1	(1,3)	316	51681			run out

The best parameter set is `k=1000` and `ngram_range=(1,1)`

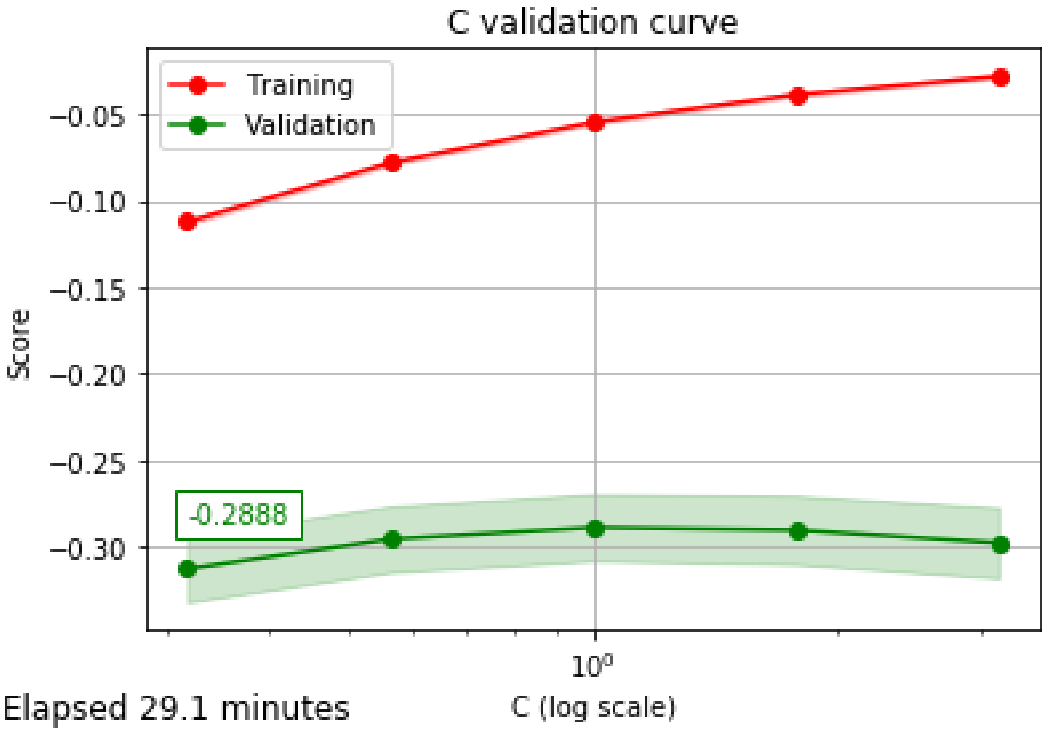
9.2 Regularization validation curve

C is the inverse of regularization strength, therefore, greater values specify weaker regularization.

[20]:

```
pl.set_params(union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1000)
param_name='classifier__estimator__C'
param_range = np.logspace(-0.5, 0.5, 5)  # [0.31622777, 0.56234133, 1, 1.
 ↪77827941, 3.16227766]
param_label = 'C'
plot_validation_curve(pl, sampling[FEATURES], dummy_labels, param_name,␣
 ↪param_range, param_label,
                      cv=5, scoring=multi_multi_log_loss_scorer, n_jobs=-1,␣
 ↪#verbose=11,
                      xscale='log')
plt.show()
```

```
Started 2020-07-01T11:44
```



Having `ngram_range=(1,1)` and `k=1000`, the best parameter is `C=1`, scoring `0.2888`, elapsed `29 minutes`

9.3 Fit model and predict probabilities on holdout set

```
[14]: pl.set_params(classifier =␣
        ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
                     classifier__n_jobs = -1,
                     classifier__estimator__C = 1,
                     union__text_features__vectorizer__ngram_range = (1,1),
                     union__text_features__reducer__k = 1000)

      model_name = '0.01-k1000-logistic-regression-C1'
      pl = fit_cache(pl, sampling[FEATURES], dummy_labels, model_dir, model_name)
      to_csv_to_zip(prediction_dir, model_name, pl.predict_proba(holdout),
                     holdout.index, dummy_labels.columns)
```

```
Fitting started on 2020-07-01T14:55
[Pipeline] .. (step 1 of 2) Processing numeric_selector, total=   0.0s
[Pipeline] … (step 2 of 2) Processing imputer, total=   0.0s
[Pipeline] … (step 1 of 3) Processing text_selector, total=   0.1s
[Pipeline] … (step 2 of 3) Processing vectorizer, total=   0.1s
[Pipeline] … (step 3 of 3) Processing reducer, total=  18.4s
```

```
[Pipeline] … (step 1 of 4) Processing union, total=  18.6s
[Pipeline] … (step 2 of 4) Processing interactor, total=   0.1s
[Pipeline] … (step 3 of 4) Processing scaler, total=   0.1s
[Pipeline] … (step 4 of 4) Processing classifier, total= 1.1min
Done: 1.4 minutes
Saving cache 0.01-k1000-logistic-regression-C1 … Done: 0.0 minutes
Saving CSV…Done: 0.2 minutes
Zipping…Done: 0.2 minutes
```

`ngram_range=(1,1)`, `k=1000` and `C=1`, elapsed `1.4 minutes`, DrivenData score: `0.5598`

10 10% data sample

[11]:

```
sampling = multilabel_sample_dataframe(df, y, size=0.1, min_count=2, seed=1)
print('Sample size:', sampling.shape[0])                                      #␣
 ↪40027
print('Train sizes:', (sampling.shape[0] * train_sizes * .8).astype('int'))   #␣
 ↪[6404 12808 19212 25617 32021]
print('Test sizes :', (sampling.shape[0] * train_sizes * .2).astype('int'))   #␣
 ↪[1601 3202 4803 6404 8005]
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
```

```
Sample size: 40027
Train sizes: [ 6404 12808 19212 25617 32021]
Test sizes : [1601 3202 4803 6404 8005]
```

10.1 Learning curves for 1-gram features

[12]:

```
classifiers = [
    ('Logistic regression (liblinear)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear'))),
    #('Logistic regression (lbfgs)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='lbfgs'))), # max_iter=200
    #('Logistic regression (sag)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='sag'))), # max_iter=4000
    #('Logistic regression (saga)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='saga'))), # max_iter=3200
    #('Logistic regression (newton-cg)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='newton-cg')))
]
```

Finding thek parameter space limits for each `ngram_range`

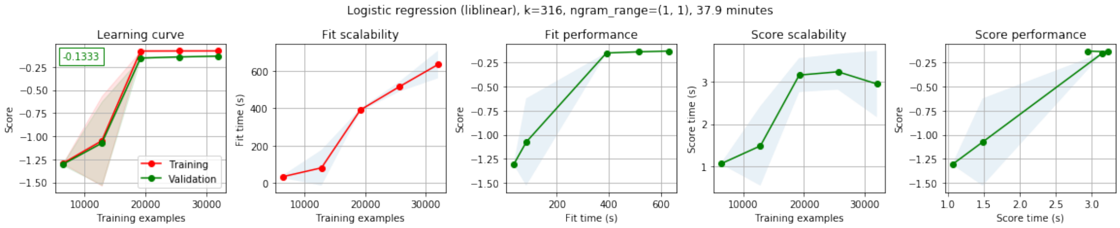
[13]:

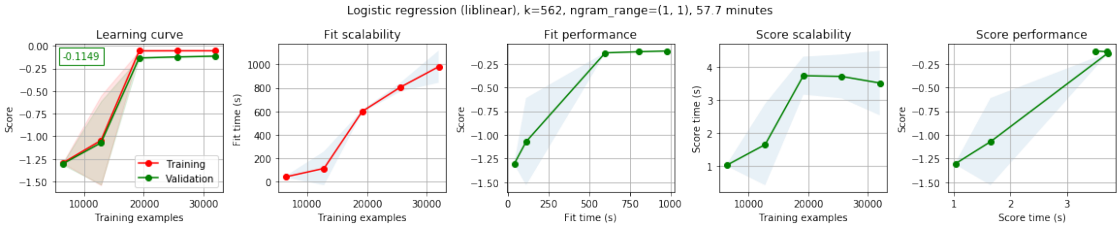
```
ngram_kmax = [len(CountVectorizer(ngram_range=(1,n),
                                  dtype='uint8').
 ↪fit(combine_text_columns(sampling)).vocabulary_)
              for n in range(1, 4)] # [2579, 18282, 46898]
```

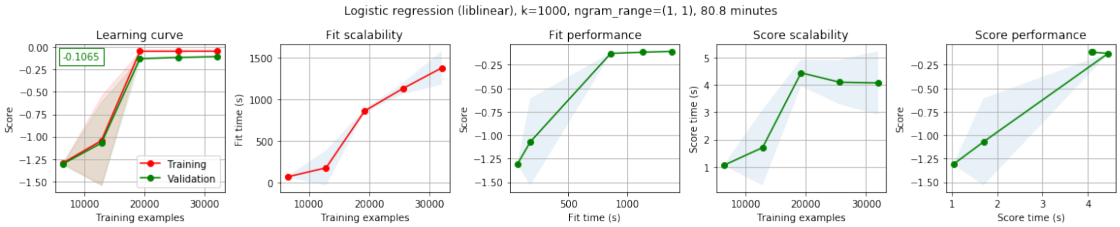
The learning curves:

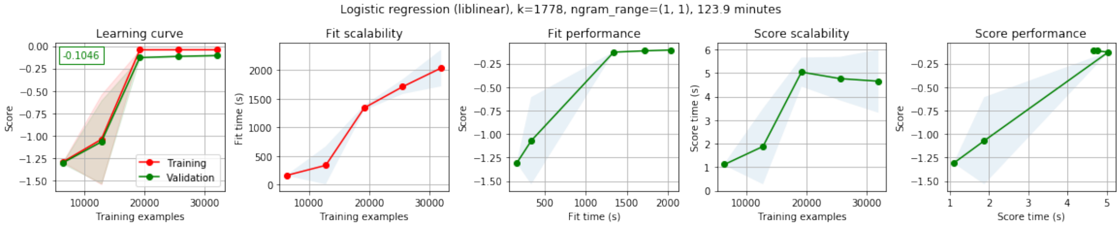
```
[14]: ngram1_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[0])), 7).round().
       ↪astype('int')
      ngram1_logspace = np.hstack((ngram1_logspace[ngram1_logspace < ngram_kmax[0]],␣
       ↪ngram_kmax[0]))
      parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
       ↪[(1,1)],
                                   'union__text_features__reducer__k':␣
       ↪ngram1_logspace}]) # [316, 562, 1000, 1778, 2579]
      print(datetime.now().isoformat(timespec='minutes'))
      for title, classifier in classifiers:
          pl.set_params(classifier = classifier)
          pl.set_params(classifier__n_jobs = None)
          for parameter in parameters:
              pl.set_params(**parameter)
              plot_learning_curve(pl, ', '.join([title] + [k.
       ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                                  sampling[FEATURES], dummy_labels, cv=5,␣
       ↪scoring=multi_multi_log_loss_scorer,
                                  n_jobs=-1, #verbose=11,
                                  train_sizes=train_sizes)
              plt.show()
```

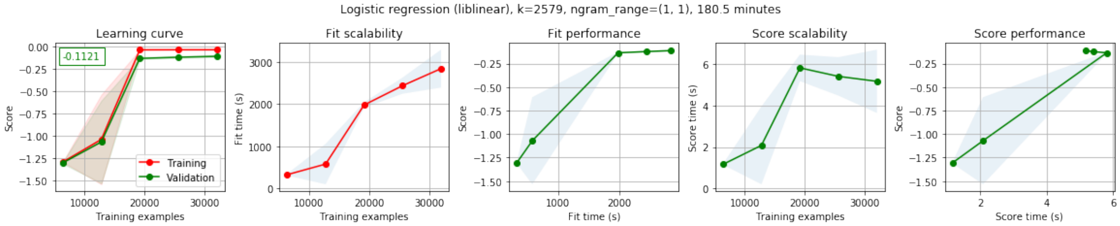
```
2020-07-05T06:56
```


Logistic regression (liblinear), k=316, ngram_range=(1, 1), 37.9 minutes


Logistic regression (liblinear), k=562, ngram_range=(1, 1), 57.7 minutes


Logistic regression (liblinear), k=1000, ngram_range=(1, 1), 80.8 minutes


Logistic regression (liblinear), k=1778, ngram_range=(1, 1), 123.9 minutes


Logistic regression (liblinear), k=2579, ngram_range=(1, 1), 180.5 minutes

Based on 2-gram's 1% sample training results, 2-gram range won't be further used.

10.2 Metrics summary

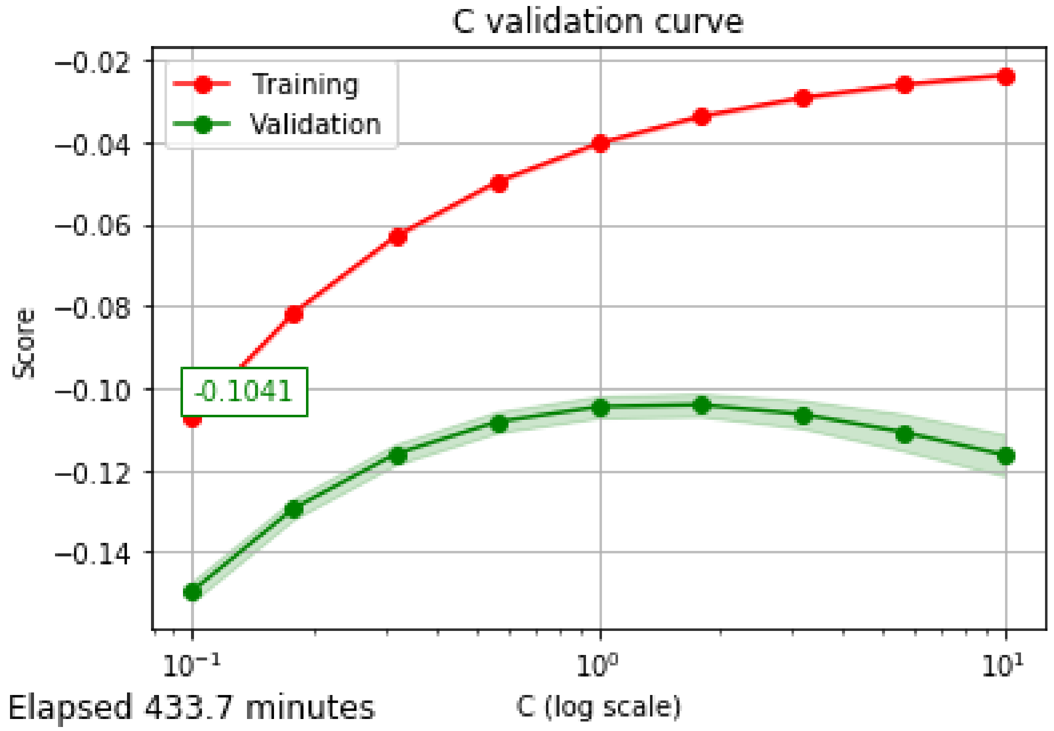
sample	min. classes	jobs	k	interactions	logloss	time	mem. peak
0.1	2	4	316	51681	0.1333	38.6 min	2.2 GiB
0.1	2	4	562	161028	0.1149	58.5 min	2.2 GiB
0.1	2	4	1000	505515	0.1065	85.9 min	2.2 GiB
0.1	2	4	**1778**	1590436	**0.1046**	125.5 min	3.1 GiB
0.1	2	4	2579	3339820	0.1121	180.8 min	3.1 GiB

The best parameter is `k=1778`

10.3 Regularization validation curve

```
[15]: pl.set_params(classifier =␣
       ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
                    union__text_features__vectorizer__ngram_range = (1,1),
                    union__text_features__reducer__k = 1778)
      param_name='classifier__estimator__C'
      param_range = np.logspace(-1, 1, 9)  # [0.1, 0.17782794100389223, 0.
       ↪31622776601683794, 0.5623413251903491, 1,
                                           # 1.7782794100389228, 3.1622776601683795,␣
       ↪5.623413251903491, 10]
      param_label = 'C'
      plot_validation_curve(pl, sampling[FEATURES], dummy_labels, param_name,␣
       ↪param_range, param_label,
                            cv=5, scoring=multi_multi_log_loss_scorer, n_jobs=-1,␣
       ↪#verbose=11,
                            xscale='log')
      plt.show()
```

```
Started 2020-07-05T14:56
```



Elapsed 433.7 minutes

Having `ngram_range=(1,1)` and `k=1778`, the best parameter is `1.7782794100389228` (decreasing regularization), scoring `0.1041`, elapsed 434 minutes

10.4 Fit model and predict probabilities on holdout set

[ ]:

```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              classifier__n_jobs = -1,
              classifier__estimator__C = 1.7782794100389228,
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)

model_name = '0.1-k1778-logistic-regression-C1.7782794100389228'
pl = fit_cache(pl, sampling[FEATURES], dummy_labels, model_dir, model_name)
to_csv_to_zip(prediction_dir, model_name, pl.predict_proba(holdout),
              holdout.index, dummy_labels.columns)
```

`ngram_range=(1,1)`, `k=1778` and `C=1.7782794100389228`, elapsed `10.3 minutes`, DrivenData score: `0.4990`

11 20% data sample

[11]:

```
sampling = multilabel_sample_dataframe(df, y, size = 0.2, min_count = 2, seed =␣
 ↪1)
print('Sample size:', sampling.shape[0])                                      #␣
 ↪80055
print('Train sizes:', (sampling.shape[0] * train_sizes * .8).astype('int'))  #␣
 ↪[12808 25617 38426 51235 64044]
print('Test sizes :', (sampling.shape[0] * train_sizes * .2).astype('int'))  #␣
 ↪[3202  6404  9606 12808 16011]
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
```

```
Sample size: 80055
Train sizes: [12808 25617 38426 51235 64044]
Test sizes : [ 3202  6404  9606 12808 16011]
```

11.1 Learning curves for 1-gram features

[12]:

```
classifiers = [
    ('Logistic regression (liblinear)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear'))),
    #('Logistic regression (lbfgs)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='lbfgs'))), # max_iter=200
    #('Logistic regression (sag)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='sag'))), # max_iter=4000
    #('Logistic regression (saga)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='saga'))), # max_iter=3200
    #('Logistic regression (newton-cg)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='newton-cg')))
]
```

Finding thek parameter space limits for each `ngram_range`

```
[13]: ngram_kmax = [len(CountVectorizer(ngram_range=(1,n),
                                        dtype='uint8').
        ↪fit(combine_text_columns(sampling)).vocabulary_)
                     for n in range(1, 4)] # [2959, 22304, 58861]
```

```
[14]: ngram1_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[0])), 7).round().
        ↪astype('int')
      ngram1_logspace = np.hstack((ngram1_logspace[ngram1_logspace < ngram_kmax[0]],␣
        ↪ngram_kmax[0]))
      ngram1_logspace # [ 316,  562, 1000, 1778, 2959]
```

```
[14]: array([ 316,  562, 1000, 1778, 2959])
```

```
[ ]: parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
        ↪[(1,1)],
                                  'union__text_features__reducer__k':␣
        ↪ngram1_logspace}])
      print(datetime.now().isoformat(timespec='minutes'))
      for title, classifier in classifiers:
          pl.set_params(classifier = classifier)
          pl.set_params(classifier__n_jobs = None)
          for parameter in parameters:
              pl.set_params(**parameter)
              plot_learning_curve(pl, ', '.join([title] + [k.
        ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                                  sampling[FEATURES], dummy_labels, cv=5,␣
        ↪scoring=multi_multi_log_loss_scorer,
                                  n_jobs=-1, verbose=11, train_sizes=train_sizes)
              plt.show()
```

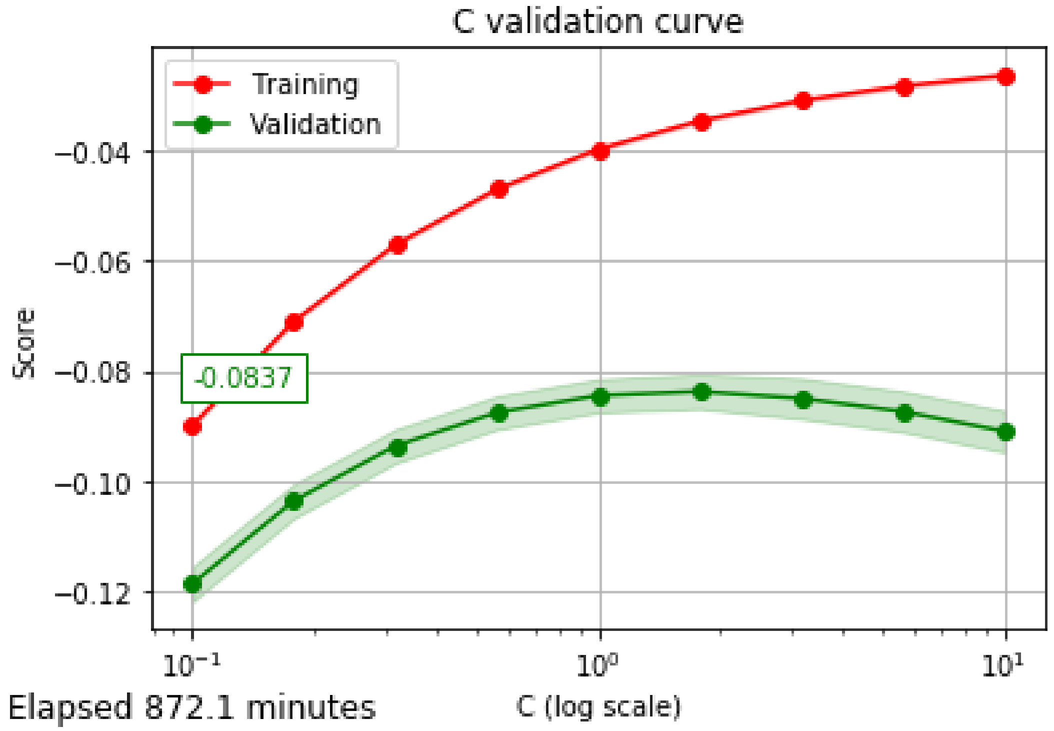
11.2 Metrics summary

	min.						
sample	classes	jobs	k	interactions	logloss	time	mem. peak
0.2	2	4	316	51681	0.1132	94.0 min	2.3 GiB
0.2	2	4	562	161028	0.0946	142.6 min	2.3 GiB
0.2	2	4	1000	505515	0.0866	194.4 min	2.3 GiB
0.2	2	4	**1778**	1590436	**0.0844**	268.3 min	3.0 GiB
0.2	2	4	2959	4394130	0.0868	408.2 min	4.2 GiB

11.3 Regularization validation curve

[15]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)
param_name='classifier__estimator__C'
param_range = np.logspace(-1, 1, 9)  # [0.1,0.1778279410038923, 0.
 ↪31622776601683794, 0.5623413251903491, 1,
                                     # 1.7782794100389228, 3.1622776601683795,␣
 ↪5.623413251903491, 10]
param_label = 'C'
plot_validation_curve(pl, sampling[FEATURES], dummy_labels, param_name,␣
 ↪param_range, param_label,
                      cv=5, scoring=multi_multi_log_loss_scorer, n_jobs=-1,␣
 ↪#verbose=11,
                      xscale='log')
plt.show()
```

```
Started 2020-07-06T00:26
```



Having `ngram_range=(1,1)` and `k=1778`, the best parameter is `C=1.7782794100389228`, scoring `0.0837`, elapsed `872 minutes`

11.4 Fit model and predict probabilities on holdout set

[13]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              classifier__n_jobs = -1,
              classifier__estimator__C = 0.1778279410038923,
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)

model_name = '0.2-k1778-logistic-regression-C0.1778279410038923'
pl = fit_cache(pl, sampling[FEATURES], dummy_labels, model_dir, model_name)
to_csv_to_zip(prediction_dir, model_name, pl.predict_proba(holdout),
              holdout.index, dummy_labels.columns)
```

```
Fitting started on 2020-07-15T07:00
[Pipeline] .. (step 1 of 2) Processing numeric_selector, total=   0.1s
[Pipeline] … (step 2 of 2) Processing imputer, total=   0.0s
[Pipeline] … (step 1 of 3) Processing text_selector, total=   1.0s
[Pipeline] … (step 2 of 3) Processing vectorizer, total=   1.8s
[Pipeline] … (step 3 of 3) Processing reducer, total=  19.9s
[Pipeline] … (step 1 of 4) Processing union, total=  23.0s
[Pipeline] … (step 2 of 4) Processing interactor, total=   1.0s
[Pipeline] … (step 3 of 4) Processing scaler, total=   2.1s
[Pipeline] … (step 4 of 4) Processing classifier, total=16.7min
Done: 17.1 minutes
Saving cache 0.2-k1778-logistic-regression-C0.1778279410038923 … Done: 0.0
minutes
Saving CSV…Done: 0.2 minutes
Zipping…Done: 0.3 minutes
```

sample	ngram	k	C	minutes	DrivenData's logloss
0.2	(1,1)	1778	0.1778279410038923	17	0.4619
0.2	(1,1)	1778	**0.31622776601683794**	19	**0.4609**
0.2	(1,1)	1778	0.5623413251903491	21	0.4670
0.2	(1,1)	1778	1	22	0.4800
0.2	(1,1)	1778	1.7782794100389228	27	0.4995

12 30% data sample

[16]:
```
sampling = multilabel_sample_dataframe(df, y, size = 0.3, min_count = 0, seed =␣
 ↪1)
print('Sample size:', sampling.shape[0])                                    #␣
 ↪120083
print('Train sizes:', (sampling.shape[0] * train_sizes * .8).astype('int'))  #␣
 ↪[19213 38426 57639 76853 96066]
```

```
print('Test sizes :', (sampling.shape[0] * train_sizes * .2).astype('int'))  #␣
 ↪[ 4803  9606 14409 19213 24016]
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
```

```
Sample size: 120083
Train sizes: [19213 38426 57639 76853 96066]
Test sizes : [ 4803  9606 14409 19213 24016]
```

12.1 Learning curves for 1-gram features

[17]:
```
classifiers = [
    ('Logistic regression (liblinear)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear'))),
    #('Logistic regression (lbfgs)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='lbfgs'))), # max_iter=200
    #('Logistic regression (sag)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='sag'))), # max_iter=4000
    #('Logistic regression (saga)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='saga'))), # max_iter=3200
    #('Logistic regression (newton-cg)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='newton-cg')))
]
```

Finding thek parameter space limits for each `ngram_range`

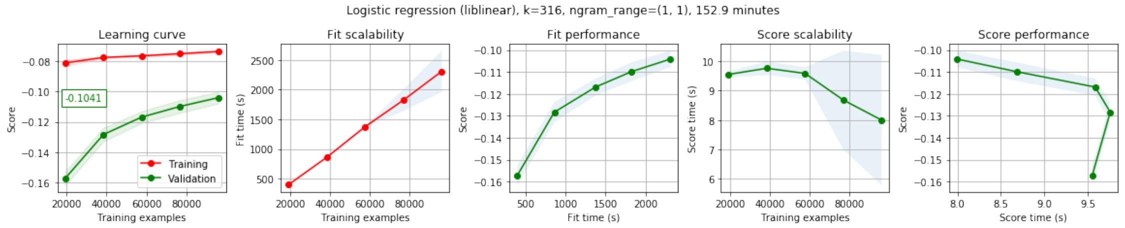
[13]:
```
ngram_kmax = [len(CountVectorizer(ngram_range=(1,n),
                                  dtype='uint8').
 ↪fit(combine_text_columns(sampling)).vocabulary_)
              for n in range(1, 4)] # [3122, 24721, 66367]
```

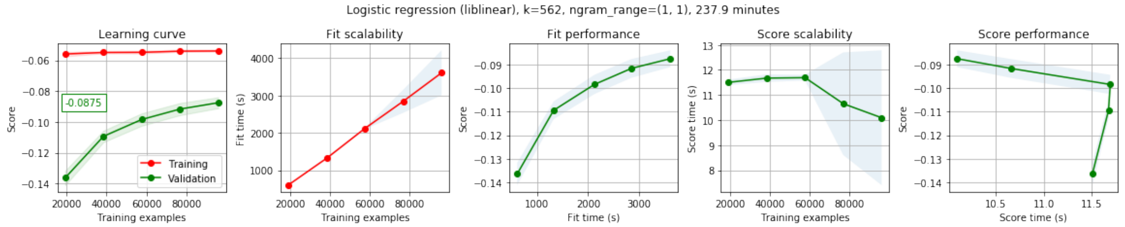
[14]:
```
ngram1_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[0])), 7).round().
 ↪astype('int')
ngram1_logspace = np.hstack((ngram1_logspace[ngram1_logspace < ngram_kmax[0]],␣
 ↪ngram_kmax[0]))
ngram1_logspace # [ 316,  562, 1000, 1778, 3122]
```

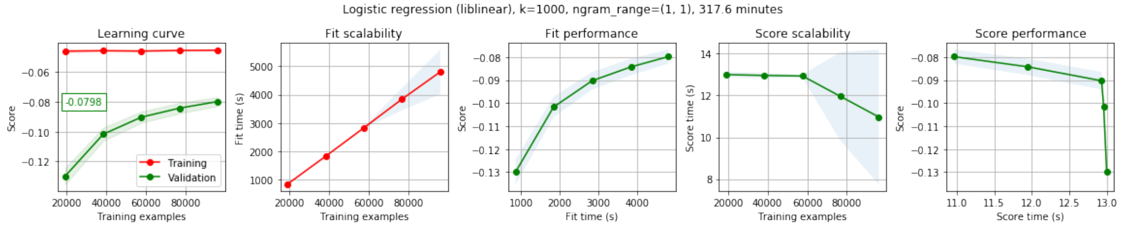
[14]:
```
array([ 316,  562, 1000, 1778, 3122])
```

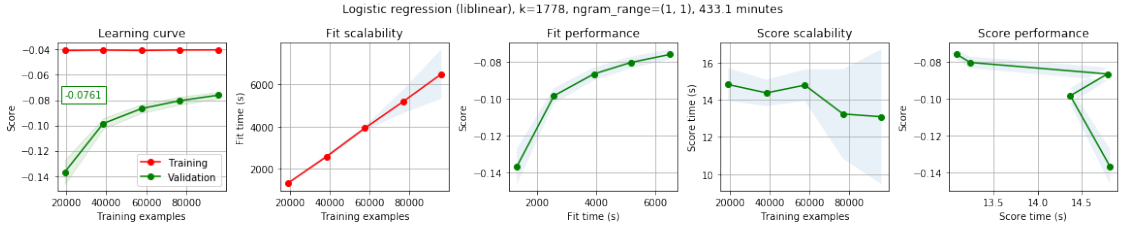
[15]:
```
parameters = ParameterGrid([{'union__text_features__vectorizer__ngram_range':␣
 ↪[(1,1)],
                             'union__text_features__reducer__k':␣
 ↪ngram1_logspace}])
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    pl.set_params(classifier__n_jobs = None)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            sampling[FEATURES], dummy_labels, cv=5,␣
↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=-1, #verbose=11,
                            train_sizes=train_sizes)
        plt.show()
```

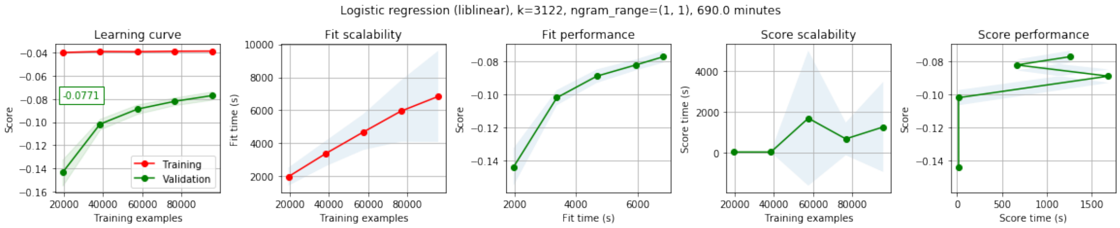
2020-07-06T20:22


Logistic regression (liblinear), k=316, ngram_range=(1, 1), 152.9 minutes


Logistic regression (liblinear), k=562, ngram_range=(1, 1), 237.9 minutes


Logistic regression (liblinear), k=1000, ngram_range=(1, 1), 317.6 minutes


Logistic regression (liblinear), k=1778, ngram_range=(1, 1), 433.1 minutes



12.2 Metrics summary

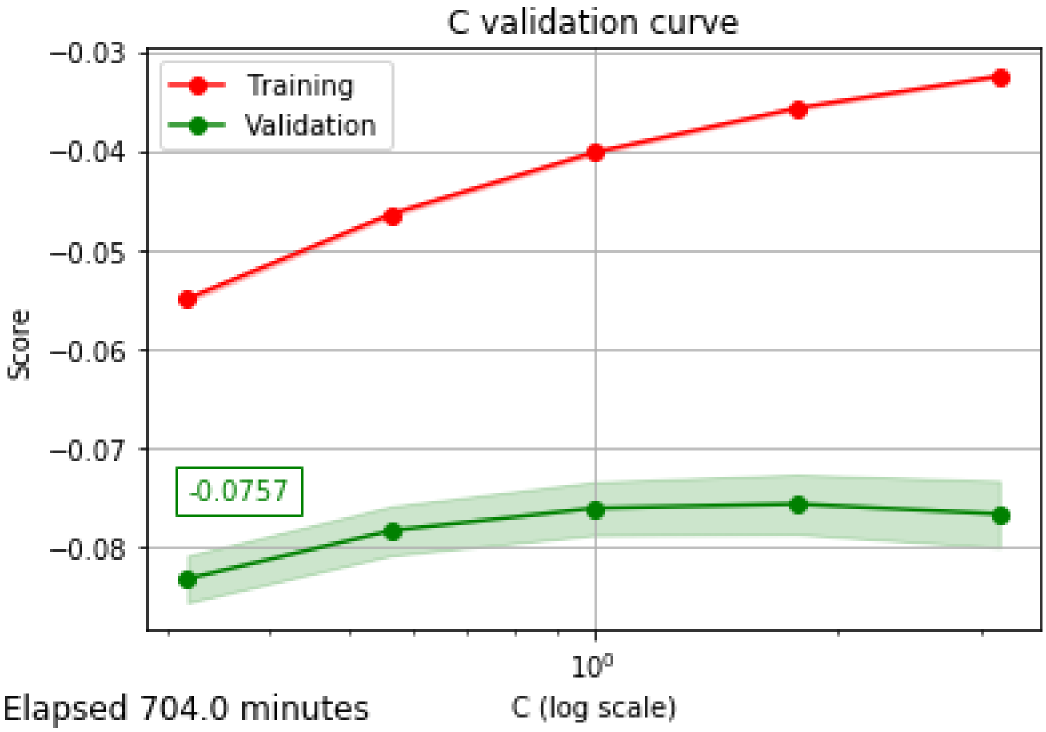
sample	min. classes	jobs	k	interactions	logloss	time	mem. peak
0.3	0	4	316	51681	0.1041	153 min	GiB
0.3	0	4	562	161028	0.0875	238 min	GiB
0.3	0	4	1000	505515	0.0798	318 min	GiB
0.3	0	4	**1778**	1590436	**0.0761**	433 min	GiB
0.3	0	4	3122	4890628	0.0771	690 min	GiB

12.3 Regularization validation curve

[13]:

```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)
param_name='classifier__estimator__C'
param_range = np.logspace(-0.5, 0.5, 5) # [0.31622777, 0.56234133, 1.        ,␣
 ↪1.77827941, 3.16227766]
param_label = 'C'
plot_validation_curve(pl, sampling[FEATURES], dummy_labels, param_name,␣
 ↪param_range, param_label,
                      cv=5, scoring=multi_multi_log_loss_scorer, n_jobs=-1,␣
 ↪#verbose=11,
                      xscale='log')
plt.show()
```

```
Started 2020-07-08T04:18
```



Having `ngram_range=(1,1)` and `k=1778`, the best parameter is `C=1.7782794100389228`, scoring `0.0757`, elapsed `704 minutes`.

12.4 Fit model and predict probabilities on holdout set

[18]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              classifier__n_jobs = -1,
              classifier__estimator__C = 0.5623413251903491,
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)

model_name = '0.3-k1778-logistic-regression-C0.5623413251903491'
pl = fit_cache(pl, sampling[FEATURES], dummy_labels, model_dir, model_name)
to_csv_to_zip(prediction_dir, model_name, pl.predict_proba(holdout),
              holdout.index, dummy_labels.columns)
```

```
Fitting started on 2020-07-13T12:33
[Pipeline] .. (step 1 of 2) Processing numeric_selector, total=   0.1s
[Pipeline] … (step 2 of 2) Processing imputer, total=   0.0s
[Pipeline] … (step 1 of 3) Processing text_selector, total=   1.5s
[Pipeline] … (step 2 of 3) Processing vectorizer, total=   3.7s
[Pipeline] … (step 3 of 3) Processing reducer, total=  20.9s
```

```
[Pipeline] … (step 1 of 4) Processing union, total=  26.2s
[Pipeline] … (step 2 of 4) Processing interactor, total=   1.4s
[Pipeline] … (step 3 of 4) Processing scaler, total=   3.0s
[Pipeline] … (step 4 of 4) Processing classifier, total=31.2min
Done: 31.8 minutes
Saving cache 0.3-k1778-logistic-regression-C0.5623413251903491 … Done: 0.0
minutes
Saving CSV…Done: 0.3 minutes
Zipping…Done: 0.2 minutes
```

sample	ngram	k	C	minutes	DrivenData's logloss
0.3	(1,1)	1778	**0.5623413251903491**	32	**0.4890**
0.3	(1,1)	1778	1	36	0.5095
0.3	(1,1)	1778	1.7782794100389228	40	0.5372

13 100% data

[11]:

```
print('Sample size:', df.shape[0])                                    # 400277
print('Train sizes:', (df.shape[0] * train_sizes * .8).astype('int'))  # [64044␣
 ↪128088 192132 256177 320221]
print('Test sizes :', (df.shape[0] * train_sizes * .2).astype('int'))  # [16011␣
 ↪32022 48033 64044 80055]
```

```
Sample size: 400277
Train sizes: [ 64044 128088 192132 256177 320221]
Test sizes : [16011 32022 48033 64044 80055]
```

13.1 Learning curves for 1-gram features

[12]:

```
classifiers = [
    ('Logistic regression (liblinear)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear'))) #,
    #('Logistic regression (lbfgs)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='lbfgs'))), # max_iter=200
    #('Logistic regression (sag)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='sag'))), # max_iter=4000
    #('Logistic regression (saga)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='saga'))), # max_iter=3200
    #('Logistic regression (newton-cg)',␣
 ↪OneVsRestClassifier(LogisticRegression(solver='newton-cg')))
]
```

Finding thek parameter space limits for each `ngram_range`

[14]:

```
ngram_kmax = [len(CountVectorizer(ngram_range=(1,n),
                                              dtype='uint8').fit(combine_text_columns(df)).
 ↪vocabulary_)
               for n in range(1, 4)]  # [3728, 32572, 91308]
```

[15]:
```
ngram1_logspace = np.logspace(2.5, np.ceil(np.log10(ngram_kmax[0])), 7).round().
 ↪astype('int')
ngram1_logspace = np.hstack((ngram1_logspace[ngram1_logspace < ngram_kmax[0]],␣
 ↪ngram_kmax[0]))
ngram1_logspace # array([ 316,  562, 1000, 1778, 3162, 3728])
```

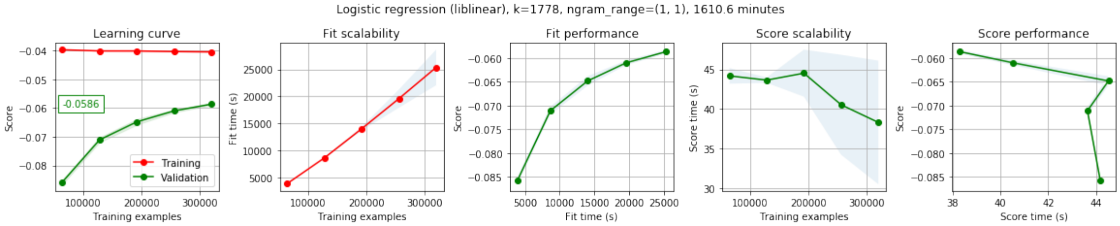
[15]:
```
array([ 316,  562, 1000, 1778, 3162, 3728])
```

[16]:
```
parameters = ParameterGrid({'union__text_features__reducer__k': [1778], #␣
 ↪ngram1_logspace,
                                      'union__text_features__vectorizer__ngram_range':␣
 ↪[(1,1)]})
print(datetime.now().isoformat(timespec='minutes'))
for title, classifier in classifiers:
    pl.set_params(classifier = classifier)
    for parameter in parameters:
        pl.set_params(**parameter)
        plot_learning_curve(pl, ', '.join([title] + [k.
 ↪split('__')[-1]+'='+str(v) for k,v in parameter.items()]),
                            df[FEATURES], y, cv=5,␣
 ↪scoring=multi_multi_log_loss_scorer,
                            n_jobs=-1, verbose=11, train_sizes=train_sizes)
        plt.show()
```

```
2020-07-09T13:51
[learning_curve] Training set sizes: [ 64044 128088 192132 256176 320221]

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  25 | elapsed: 148.8min remaining:
1711.3min
[Parallel(n_jobs=-1)]: Done   5 out of  25 | elapsed: 354.8min remaining:
1419.3min
[Parallel(n_jobs=-1)]: Done   8 out of  25 | elapsed: 527.7min remaining:
1121.4min
[Parallel(n_jobs=-1)]: Done  11 out of  25 | elapsed: 695.0min remaining:
884.5min
[Parallel(n_jobs=-1)]: Done  14 out of  25 | elapsed: 838.0min remaining:
658.4min
[Parallel(n_jobs=-1)]: Done  17 out of  25 | elapsed: 1070.7min remaining:
503.9min
[Parallel(n_jobs=-1)]: Done  20 out of  25 | elapsed: 1282.3min remaining:
320.6min
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed: 1461.7min remaining:
```

```
127.1min
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed: 1610.6min finished
```



13.2 Metrics summary

sample	jobs	k	interactions	logloss	time	mem. peak
1.0	4	316	51681	0.0834	667.9 min	2.8-3.1 GiB
1.0	4	562	161028	0.0686	998.1 min	2.8-3.1 GiB
1.0	4	1000	505515	0.0623	1291.2 min	2.8-3.1 GiB
1.0	4	**1778**	1590436	**0.0586**	1610.6 min	2.8-3.1 GiB

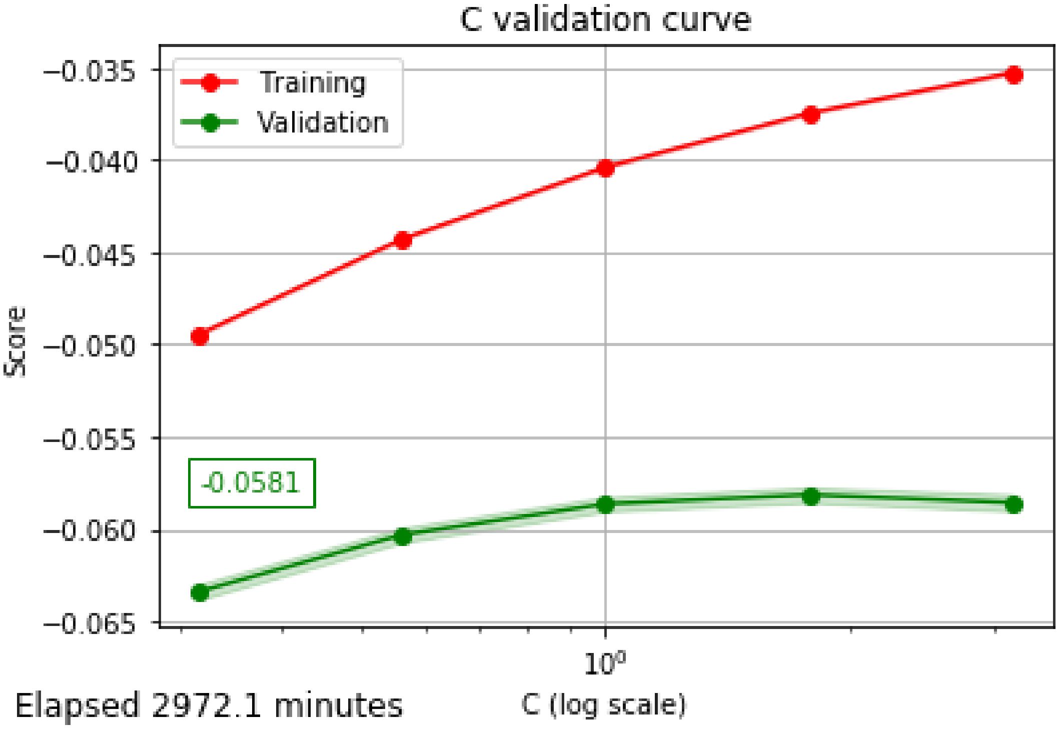
13.3 Regularization validation curve

[11]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
              union__text_features__vectorizer__ngram_range = (1,1),
              union__text_features__reducer__k = 1778)
param_name='classifier__estimator__C'
param_range = np.logspace(-0.5, 0.5, 5) # [0.31622777, 0.56234133, 1.        ,␣
 ↪1.77827941, 3.16227766]
param_label = 'C'
plot_validation_curve(pl, df[FEATURES], y, param_name, param_range, param_label,
                      cv=5, scoring=multi_multi_log_loss_scorer, n_jobs=-1,␣
 ↪verbose=11, xscale='log')
plt.show()
```

```
Started 2020-07-10T18:10

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of  25 | elapsed: 390.0min remaining:
4484.6min
[Parallel(n_jobs=-1)]: Done   5 out of  25 | elapsed: 723.7min remaining:
2895.0min
[Parallel(n_jobs=-1)]: Done   8 out of  25 | elapsed: 985.4min remaining:
2094.0min
[Parallel(n_jobs=-1)]: Done  11 out of  25 | elapsed: 1370.6min remaining:
1744.4min
```

```
[Parallel(n_jobs=-1)]: Done  14 out of  25 | elapsed: 1835.2min remaining:
1442.0min
[Parallel(n_jobs=-1)]: Done  17 out of  25 | elapsed: 2087.5min remaining:
982.4min
[Parallel(n_jobs=-1)]: Done  20 out of  25 | elapsed: 2420.6min remaining:
605.2min
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed: 2778.7min remaining:
241.6min
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed: 2972.1min finished
```



Having `ngram_range=(1,1)` and `k=1778`, the best is `C=1.7782794100389228`, scoring `0.0581`, elapsed `2972 minutes`

13.4 Fit model and predict probabilities on holdout set

```
[ ]: pl.set_params(classifier =␣
       ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
                     classifier__n_jobs = -1,
                     classifier__estimator__C = 0.05623413251903491,
                     union__text_features__vectorizer__ngram_range = (1,1),
                     union__text_features__reducer__k = 1778)
     model_name = '1.0-k1778-logistic-regression-C0.05623413251903491'
     pl = fit_cache(pl, df[FEATURES], y, model_dir, model_name)
```

```
to_csv_to_zip(prediction_dir, model_name, pl.predict_proba(holdout), holdout.
 ↪index, y.columns)
```

sample	ngram	k	C	minutes	DrivenData's logloss
1.0	(1,1)	1778	0.01	80	0.4934
1.0	(1,1)	1778	0.05623413251903491	80	0.4465
1.0	(1,1)	1778	**0.1**	100	**0.4461**
1.0	(1,1)	1778	0.1778279410038923	101	0.4530
1.0	(1,1)	1778	0.31622776601683794	115	0.4675
1.0	(1,1)	1778	0.5623413251903491	136	0.4889
1.0	(1,1)	1778	1	150	0.5173
1.0	(1,1)	1778	1.7782794100389228	177	0.5523

14 Parameter optimizations and predictions

[ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, log_loss
```

14.1 0.8% training, 0.2% testing

[ ]:

```
sampling = multilabel_sample_dataframe(df, y, size = 0.01, min_count = 7, seed␣
 ↪= 1)
print('Sample size:', sampling.shape[0])
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
X_train, X_test, y_train, y_test =␣
 ↪multilabel_train_test_split(sampling[FEATURES], dummy_labels,
                                                               size = 0.2,␣
 ↪min_count = 1, seed=1)
print('Train size`:', y_train.shape[0])
print('Test size  :', y_test.shape[0])
```

[ ]:

```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))
parameters = {'union__text_features__reducer__k' : np.logspace(2, 3, 5).round().
 ↪astype('int'),
              'classifier__estimator__C' : np.logspace(0, 2, 3)} #[1 10 100]
grid = GridSearchCV(estimator = pl,
                    n_jobs = -1,
                    param_grid = parameters,
                    cv = 5,
                    scoring = {'logloss' : multi_multi_log_loss_scorer},
                    refit = 'logloss',
                    verbose=11)
```

[ ]:

```
model_name = '0.008-all-features-gridsearch-logistic-regression'
grid = fit_cache(grid, X_train, y_train, model_dir, model_name)
```

Fitting 5 folds for each of 15 candidates, totalling 75 fits, took ~120 minutes

[ ]:

```
print('Time refitting best model on whole data  : {:.1f} minutes'.format(grid.
 ↪refit_time_ / 60))
results = pd.DataFrame(grid.cv_results_)
results = results.rename(columns={'param_classifier__estimator__C':'C',
                                  'param_union__text_features__reducer__k':'k',
                                  'split0_test_multi_multi_log_loss':
 ↪'split0_logloss',
                                  'split1_test_multi_multi_log_loss':
 ↪'split1_logloss',
                                  'split2_test_multi_multi_log_loss':
 ↪'split2_logloss',
                                  'split3_test_multi_multi_log_loss':
 ↪'split3_logloss',
                                  'split4_test_multi_multi_log_loss':
 ↪'split4_logloss',
                                  'mean_test_multi_multi_log_loss':
 ↪'mean_test_logloss',
                                  'std_test_multi_multi_log_loss':
 ↪'std_test_logloss',
                                  'rank_test_multi_multi_log_loss':
 ↪'rank_test_logloss'
                                 })
results[['C', 'k', 'mean_test_logloss', 'std_test_logloss','rank_test_logloss',␣
 ↪'mean_fit_time', 'std_fit_time', 'mean_score_time',
         'std_score_time']].head(60)
```

[ ]:

```
#from sklearn.model_selection import cross_val_score
#pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')),
#              union__text_features__reducer__k = 562,
#              classifier__estimator__C = 1)
#cross_val_score(pl, X_train, y_train,
#                scoring = multi_multi_log_loss_scorer,
#                cv = 5,
#                n_jobs = -1,
#                verbose=11).mean()
```

[ ]:

```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Training data score        : {}".format(grid.score(X_train, y_train))) ␣
 ↪# Takes some minutes
y_pred = grid.predict(X_train)  # Takes some minutes
```

```
report = pd.DataFrame(classification_report(y_train, y_pred,␣
 ↪target_names=y_train.columns, output_dict=True)).transpose()
report, summary = report[:-4].sort_values('f1-score', ascending=False),␣
 ↪report[-4:]
display(report)
display(summary)
```

[ ]:
```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Testing data score         : {}".format(grid.score(X_test, y_test)))  #␣
 ↪Takes some minutes

y_pred = grid.predict(X_test)  # Takes some minutes
report = pd.DataFrame(classification_report(y_test, y_pred, target_names=y_test.
 ↪columns, output_dict=True)).transpose()
report, summary = report[:-4].sort_values('f1-score', ascending=False),␣
 ↪report[-4:]
display(report)
display(summary)
```

[ ]:
```
# Testing score breakdown checking
y_pred = grid.predict_proba(X_test)  # Takes some minutes
pd.DataFrame(multi_multi_log_loss(y_test, y_pred, class_column_indices=cci,␣
 ↪averaged=False), index=LABELS,
             columns=['multi-multi log loss'])
```

[ ]:
```
t=time()
y_pred = grid.predict_proba(holdout)
print('Elapsed: {:.1f} minutes'.format(np.floor(time()-t)/60))
```

[ ]:
```
to_csv_zip(prediction_dir, model_name, y_pred, holdout.index, y.columns)
```

14.2 8% training, 2% testing

[ ]:
```
sampling = multilabel_sample_dataframe(df, y, size = 0.1, min_count = 2, seed =␣
 ↪1)
print('Sample size:', sampling.shape[0])
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
X_train, X_test, y_train, y_test =␣
 ↪multilabel_train_test_split(sampling[FEATURES], dummy_labels,
                                                               size = 0.2,␣
 ↪min_count = 0, seed=1)
print('Train size`:', y_train.shape[0])
print('Test size  :', y_test.shape[0])
```

[ ]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))
```

```
parameters = {'union__text_features__reducer__k' : np.logspace(2, 3, 5).round().
 ↪astype('int'),
              'classifier__estimator__C' : np.logspace(0, 2, 3)} #[1 10 100]
grid = GridSearchCV(estimator = pl,
                    n_jobs = -1,
                    param_grid = parameters,
                    cv = 5,
                    scoring = {'logloss' : multi_multi_log_loss_scorer},
                    refit = 'logloss',
                    verbose=11)
```

```
[ ]: model_name = '0.08-all-features-gridsearch-logistic-regression'
     grid = fit_cache(grid, X_train, y_train, model_dir, model_name)
```

Fitting 5 folds for each of 15 candidates, totalling 75 fits, took 392 minutes

```
[ ]: print('Time refitting best model on whole data  : {:.0f} minutes'.format(grid.
      ↪refit_time_ / 60))
     results = pd.DataFrame(grid.cv_results_)
     results = results.rename(columns={'param_classifier__estimator__C':'C',
                                       'param_union__text_features__reducer__k':'k'})
     results[['C', 'k', 'mean_test_logloss', 'std_test_logloss','rank_test_logloss',␣
      ↪'mean_fit_time', 'std_fit_time', 'mean_score_time',
             'std_score_time']].head(60)
```

```
[ ]: pl.set_params(union__text_features__reducer__k = 1000,
                   classifier__estimator__C = 0.1)
     cross_val_score(pl, X_train, y_train,
                     scoring = multi_multi_log_loss_scorer,
                     cv = 5,
                     n_jobs = -1,
                     verbose=11).mean()
```

```
[ ]: print('Mean cross-validated score : {}'.format(grid.best_score_))
     print("Training data score        : {}".format(grid.score(X_train, y_train))) ␣
      ↪# Takes some minutes

     y_pred = grid.predict(X_train)  # Takes some minutes
     report = pd.DataFrame(classification_report(y_train, y_pred,␣
      ↪target_names=y_train.columns, output_dict=True)).transpose()
     summary = report[:-4].sort_values('f1-score', ascending=False)
     display(report)
     display(summary)
```

```
[ ]: print('Mean cross-validated score : {}'.format(grid.best_score_))
     print("Testing data score         : {}".format(grid.score(X_test, y_test)))  #␣
      ↪Takes some minutes
```

```
y_pred = grid.predict(X_test)  # Takes some minutes
report = pd.DataFrame(classification_report(y_test, y_pred, target_names=y_test.
 ↪columns, output_dict=True)).transpose()
report, summary = report[:-4].sort_values('f1-score', ascending=False),␣
 ↪report[-4:]
display(report)
display(summary)
```

[ ]:
```
# Testing score checking
y_pred = grid.predict_proba(X_test)  # Takes some minutes
pd.DataFrame(multi_multi_log_loss(y_test, y_pred, class_column_indices=cci,␣
 ↪averaged=False), index=LABELS,
             columns=['multi-multi log loss'])
```

[ ]:
```
t=time()
y_pred = grid.predict_proba(holdout)
print('Elapsed: {:.1f} minutes'.format(np.floor(time()-t)/60))
```

[ ]:
```
to_csv_zip(prediction_dir, model_name, y_pred, holdout.index, y.columns)
```

14.3 16% training, 4% testing (TO RUN)

[ ]:
```
sampling = multilabel_sample_dataframe(df, y, size = 0.2, min_count = 2, seed =␣
 ↪1)
print('Sample size:', sampling.shape[0])
dummy_labels = pd.get_dummies(sampling[LABELS], prefix_sep='__')
X_train, X_test, y_train, y_test =␣
 ↪multilabel_train_test_split(sampling[FEATURES], dummy_labels,
                                                                size = 0.2,␣
 ↪min_count = 0, seed=1)
print('Train size`:', y_train.shape[0])
print('Test size  :', y_test.shape[0])
```

[ ]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))
parameters = {'union__text_features__reducer__k' : np.logspace(2, 3, 5).round().
 ↪astype('int'),
              'classifier__estimator__C' : np.logspace(0, 2, 3)} #[1 10 100]
grid = GridSearchCV(estimator = pl,
                    n_jobs = -1,
                    param_grid = parameters,
                    cv = 5,
                    scoring = {'logloss' : multi_multi_log_loss_scorer},
                    refit = 'logloss',
                    verbose=11)
```

[ ]:
```
model_name = '0.016-all-features-gridsearch-logistic-regression'
grid = fit_cache(grid, X_train, y_train, model_dir, model_name)
```

[ ]:
```
print('Time refitting best model on whole data  : {:.0f} minutes'.format(grid.
 ↪refit_time_ / 60))
results = pd.DataFrame(grid.cv_results_)
results = results.rename(columns={'param_classifier__estimator__C':'C',
                                  'param_union__text_features__reducer__k':'k'})
results[['C', 'k', 'mean_test_logloss', 'std_test_logloss','rank_test_logloss',␣
 ↪'mean_fit_time', 'std_fit_time', 'mean_score_time',
         'std_score_time']].head(60)
```

[ ]:
```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Training data score        : {}".format(grid.score(X_train, y_train))) ␣
 ↪# Takes some minutes

y_pred = grid.predict(X_train)  # Takes some minutes
report = pd.DataFrame(classification_report(y_train, y_pred,␣
 ↪target_names=y_train.columns, output_dict=True)).transpose()
summary = report[:-4].sort_values('f1-score', ascending=False)
display(report)
display(summary)
```

[ ]:
```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Testing data score         : {}".format(grid.score(X_test, y_test)))  #␣
 ↪Takes some minutes

y_pred = grid.predict(X_test)  # Takes some minutes
report = pd.DataFrame(classification_report(y_test, y_pred, target_names=y_test.
 ↪columns, output_dict=True)).transpose()
report, summary = report[:-4].sort_values('f1-score', ascending=False),␣
 ↪report[-4:]
display(report)
display(summary)
```

[ ]:
```
# Testing score checking
y_pred = grid.predict_proba(X_test)  # Takes some minutes
pd.DataFrame(multi_multi_log_loss(y_test, y_pred, class_column_indices=cci,␣
 ↪averaged=False), index=LABELS,
             columns=['multi-multi log loss'])
```

[ ]:
```
t=time()
y_pred = grid.predict_proba(holdout)
print('Elapsed: {:.1f} minutes'.format(np.floor(time()-t)/60))
```

[ ]:
```
to_csv_zip(prediction_dir, model_name, y_pred, holdout.index, y.columns)
```

14.4 80% training, 20% testing (TO RUN)

[ ]:
```
X_train,X_test,y_train,y_test = multilabel_train_test_split(df[FEATURES], y,␣
 ↪size=0.2, min_count=0, seed=1)
print('Train size`:', y_train.shape[0])
print('Test size  :', y_test.shape[0])
```

[ ]:
```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))
parameters = {'union__text_features__reducer__k' : np.logspace(2, 3, 5).round().
 ↪astype('int'),
              'classifier__estimator__C' : np.logspace(0, 2, 3)} #[1 10 100]
grid = GridSearchCV(estimator = pl,
                    n_jobs = -1,
                    param_grid = parameters,
                    cv = 5,
                    scoring = {'logloss' : multi_multi_log_loss_scorer},
                    refit = 'logloss',
                    verbose=11)
```

[ ]:
```
model_name = '0.8-all-features-gridsearch-logistic-regression'
grid = fit_cache(grid, X_train, y_train, model_dir, model_name)
```

[ ]:
```
print('Time refitting best model on whole data  : {:.0f} minutes'.format(grid.
 ↪refit_time_ / 60))
results = pd.DataFrame(grid.cv_results_)
results = results.rename(columns={'param_classifier__estimator__C':'C',
                                  'param_union__text_features__reducer__k':'k'})
results[['C', 'k', 'mean_test_logloss', 'std_test_logloss','rank_test_logloss',␣
 ↪'mean_fit_time', 'std_fit_time', 'mean_score_time',
         'std_score_time']].head(60)
```

[ ]:
```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Training data score        : {}".format(grid.score(X_train, y_train))) ␣
 ↪# Takes some minutes

y_pred = grid.predict(X_train)  # Takes some minutes
report = pd.DataFrame(classification_report(y_train, y_pred,␣
 ↪target_names=y_train.columns, output_dict=True)).transpose()
summary = report[:-4].sort_values('f1-score', ascending=False)
display(report)
display(summary)
```

[ ]:
```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Testing data score         : {}".format(grid.score(X_test, y_test)))  #␣
 ↪Takes some minutes
```

```
y_pred = grid.predict(X_test)  # Takes some minutes
report = pd.DataFrame(classification_report(y_test, y_pred, target_names=y_test.
 ↪columns, output_dict=True)).transpose()
report, summary = report[:-4].sort_values('f1-score', ascending=False),␣
 ↪report[-4:]
display(report)
display(summary)
```

[ ]:

```
# Testing score checking
y_pred = grid.predict_proba(X_test)  # Takes some minutes
pd.DataFrame(multi_multi_log_loss(y_test, y_pred, class_column_indices=cci,␣
 ↪averaged=False), index=LABELS,
             columns=['multi-multi log loss'])
```

[ ]:

```
t=time()
y_pred = grid.predict_proba(holdout)
print('Elapsed: {:.1f} minutes'.format(np.floor(time()-t)/60))
```

[ ]:

```
to_csv_zip(prediction_dir, model_name, y_pred, holdout.index, y.columns)
```

14.5 100% training, 0% testing (TO RUN)

[ ]:

```
df = df.sample(frac=1, random_state=1) # Ensure iid samples because␣
 ↪CVGridSearch/KFold doesn't shuffle folding data
X_train = df[FEATURES]
y_train = pd.get_dummies(df[LABELS], prefix_sep='__')
del, X_test, y_test
print('Train size`:', y_train.shape[0])
```

[ ]:

```
pl.set_params(classifier =␣
 ↪OneVsRestClassifier(LogisticRegression(solver='liblinear')))
parameters = {'union__text_features__reducer__k' : np.logspace(2, 3, 5).round().
 ↪astype('int'),
              'classifier__estimator__C' : np.logspace(0, 2, 3)} #[1 10 100]
grid = GridSearchCV(estimator = pl,
                    n_jobs = -1,
                    param_grid = parameters,
                    cv = 5,
                    scoring = {'logloss' : multi_multi_log_loss_scorer},
                    refit = 'logloss',
                    verbose=11)
```

[ ]:

```
model_name = '1.0-all-features-gridsearch-logistic-regression'
grid = fit_cache(grid, X_train, y_train, model_dir, model_name)
```

[ ]:

```
print('Time refitting best model on whole data  : {:.0f} minutes'.format(grid.
 ↪refit_time_ / 60))
results = pd.DataFrame(grid.cv_results_)
results = results.rename(columns={'param_classifier__estimator__C':'C',
                                 'param_union__text_features__reducer__k':'k'})
results[['C', 'k', 'mean_test_logloss', 'std_test_logloss','rank_test_logloss',␣
 ↪'mean_fit_time', 'std_fit_time', 'mean_score_time',
         'std_score_time']].head(60)
```

[ ]:

```
print('Mean cross-validated score : {}'.format(grid.best_score_))
print("Training data score        : {}".format(grid.score(X_train, y_train))) ␣
 ↪# Takes some minutes

y_pred = grid.predict(X_train)  # Takes some minutes
report = pd.DataFrame(classification_report(y_train, y_pred,␣
 ↪target_names=y_train.columns, output_dict=True)).transpose()
summary = report[:-4].sort_values('f1-score', ascending=False)
display(report)
display(summary)
```

[ ]:

```
# Training score checking
y_pred = grid.predict_proba(X_train)  # Takes some minutes
pd.DataFrame(multi_multi_log_loss(y_train, y_pred, class_column_indices=cci,␣
 ↪averaged=False), index=LABELS,
             columns=['multi-multi log loss'])
```

[ ]:

```
t=time()
y_pred = grid.predict_proba(holdout)
print('Elapsed: {:.1f} minutes'.format(np.floor(time()-t)/60))
```

[ ]:

```
to_csv_zip(prediction_dir, model_name, y_pred, holdout.index, y.columns)
```