DAT300_CA1_16

October 17, 2019

1 Compulsory assignment 1 - Using Dask-ml on large data

1.1 Group #16

1.1.1 Thomas Moen and Jørgen Kongsro

1.2 Import libraries

[1]:
```
%matplotlib notebook

import numpy as np
import pandas as pd
import os
import dask
import dask.dataframe as dd
import scipy
import matplotlib.pyplot as plt
import skimage.io
import dask.array as da
import joblib

from dask.diagnostics import ProgressBar
from dask_ml.linear_model import LogisticRegression
from dask_ml.model_selection import train_test_split
from dask_ml.datasets import make_classification
from dask_ml.model_selection import train_test_split
from dask_ml.linear_model import LogisticRegression
from dask_ml.metrics import accuracy_score
from dask_ml.model_selection import IncrementalSearchCV
from dask.distributed import Client

from sklearn.linear_model import SGDClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

print (dask.__version__)
```

```
2.1.0
```

1.3 Install Kaggle API and download Kaggle data

[ ]:
```python
# Install Kaggle API
# How to setup: https://github.com/Kaggle/kaggle-api
# or visit: https://adityashrm21.github.io/Setting-Up-Kaggle/

#!pip install kaggle
```

[ ]:
```python
# Download Kaggle data using Kaggle API
#!kaggle competitions download -c dat300-ca1-autumn-2019
```

[ ]:
```python
# Unzip Kaggle data
#!unzip "dat300-ca1-autumn-2019.zip" -d "/tmp/whatever"
```

1.4 List files in directory (adjust for different operating systems)

[2]:
```python
# Adjust for different os

if os.name == 'nt':
    workdir = 'C://Users//thomoe//Documents//myDAT300//dat300-ca1-autumn-2019//'
elif os.name == 'posix':
    workdir = '/Users/jorgenkongsro/Downloads/dat300-ca1-autumn-2019/'

os.listdir(workdir)
```

[2]:
```
['y_test_sampleSubmission.csv', 'X_train.csv', 'y_train.csv', 'X_test.csv']
```

1.5 Import data

[131]:
```python
def import_data(csv_file):
    """ Import data from csv file

    :param data: a .csv separated dataset
    :return: a pandas data array, df

    """



    df = dd.read_csv(csv_file)
    return df

x_train_df = import_data(workdir + 'X_train.csv')
x_test_df = import_data(workdir + 'X_test.csv')
y_train_df = import_data(workdir + 'y_train.csv')
```

```
x_train_df
y_train_df
```

[131]:
```
Dask DataFrame Structure:
                  Target
npartitions=1
                 float64
                     ...
Dask Name: from-delayed, 3 tasks
```

1.5.1 View data and study the structure

[10]:
```
x_train_df
```

[10]:
```
Dask DataFrame Structure:
                        f1        f2        f3        f4        f5        f6        f7
f8        f9       f10       f11       f12       f13       f14       f15       f16
f17       f18       f19       f20       f21       f22       f23       f24       f25
f26       f27       f28       f29       f30       f31       f32       f33       f34
f35       f36       f37       f38       f39       f40       f41       f42       f43
f44       f45       f46       f47       f48       f49       f50       f51       f52
f53       f54       f55       f56       f57       f58       f59       f60       f61
f62       f63       f64       f65       f66       f67       f68       f69       f70
f71       f72       f73       f74       f75       f76       f77       f78       f79
f80       f81       f82       f83       f84       f85       f86       f87       f88
f89       f90       f91       f92       f93       f94       f95       f96       f97
f98       f99       f100      f101      f102      f103      f104      f105      f106
f107      f108      f109      f110      f111      f112      f113      f114      f115
f116      f117      f118      f119      f120      f121      f122      f123      f124
f125      f126      f127      f128      f129      f130      f131      f132      f133
f134      f135      f136      f137      f138      f139      f140      f141      f142
f143      f144      f145      f146      f147      f148      f149      f150      f151
f152      f153      f154      f155      f156      f157      f158      f159      f160
f161      f162
npartitions=83
                  float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
float64   float64   float64   float64   float64   float64   float64   float64   float64
```

```
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64  float64  float64  float64  float64  float64  float64  float64
float64  float64
                    ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...
...                 ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...
                    ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
```

```
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...
                    ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...      ...      ...      ...      ...      ...      ...      ...
...      ...
Dask Name: from-delayed, 249 tasks
```

[11]:
```
y_train_df
```

[11]:
```
Dask DataFrame Structure:
                Target
npartitions=1
               float64
                   ...
Dask Name: from-delayed, 3 tasks
```

1.6 Check for missing data

[132]:
```
def percent_missing(dataframe):
    """ Check for percent missing values in dataframe
    :param data: dataframe
    :return: dataframe

    """
    missing_values = dataframe.isnull().sum()

    with ProgressBar():
        percent_missing = ((missing_values / dataframe.index.size) * 100).
 ↪compute()

    return percent_missing


print(percent_missing(x_train_df))

"""
note: the results indicate that the features come in "tripets", e.g. f1 to f3␣
 ↪have quite similar missing%.
We could impute some values very precisely by insert the mean of the other␣
 ↪values within the triplet,
if only one or two values are missing
"""
```

```
[########################################] | 100% Completed | 35.0s
f1       3.628002
f2       3.754723
f3       3.836795
f4       4.133142
f5       4.299061
f6       4.377457
f7       9.171993
f8       9.276405
f9       9.410288
f10      3.628002
f11      3.754723
f12      3.836795
f13      4.133142
f14      4.299061
f15      4.377457
f16      9.171993
f17      9.276405
f18      9.410288
f19      3.628002
```

```
f20     3.754723
f21     3.836795
f22     4.133142
f23     4.299061
f24     4.377457
f25     9.171993
f26     9.276405
f27     9.410288
f28     3.628002
f29     3.754723
f30     3.836795
          ...
f133    9.171993
f134    9.276405
f135    9.410288
f136    3.628002
f137    3.754723
f138    3.836795
f139    4.133142
f140    4.299061
f141    4.377457
f142    9.171993
f143    9.276405
f144    9.410288
f145    3.628002
f146    3.754723
f147    3.836795
f148    4.133142
f149    4.299061
f150    4.377457
f151    9.171993
f152    9.276405
f153    9.410288
f154    3.628002
f155    3.754723
f156    3.836795
f157    4.133142
f158    4.299061
f159    4.377457
f160    9.171993
f161    9.276405
f162    9.410288
Length: 162, dtype: float64
```

[132]:
```
'\nnote: the results indicate that the features come in "tripets", e.g. f1 to f3
have quite similar missing%. \nWe could impute some values very precisely by
insert the mean of the other values within the triplet, \nif only one or two
```

```
values are missing\n'
```

1.7 Drop rows due to missing values in target

[14]:
```python
# Delete rows in x_train_df and y_train_df where
# 1) y is NaN and/or
# 2) x has more than N missing features
# Delete the same rows in x_train_df and y_train_df

#First inspect the distribution of missing values in rows of x_train_df, in␣
↪order to decide on a threshold for culling rows:
if dir().count('nacounts_rows') == 0:
    nacounts_rows = x_train_df.isna().sum(axis=1).compute()
nacounts_rows.plot.hist(bins=20, grid=True)

#Based on the histogram we chose to remove rows of x_train_df which had more␣
↪than 150 missing features:
nacount_thresh = 150

#Make boolean list where True means the number of NaN's exceeds nacount_thresh␣
↪for that row (of x_train_df):
rows_to_drop_x = list(nacounts_rows > nacount_thresh)

#Make boolean list where True means thaty_train_df is NaN for that row:
rows_to_drop_y = list(y_train_df['Target'].isna().compute())

#Combine rows_to_drop_x and rows_to_drop_y and negate the list so that it␣
↪becomes a boolean list of rows to keep:
rows_to_keep = [not(rows_to_drop_x[a] or rows_to_drop_y[a]) for a in␣
↪range(len(rows_to_drop_y))]

#Delete rows from x_train_df and y_train_df (we did not manage to do this␣
↪without pandas dataframes from the dask dataframes):
x_train_droprows = x_train_df.compute()[rows_to_keep]
y_train_droprows = y_train_df.compute()[rows_to_keep]

#...therefore we had to revert back to dask dataframe afterwards:
x_train_df_droprows=dd.from_pandas(x_train_droprows, npartitions=249) #␣
↪npartitions based on viewing data structure in previous section
y_train_df_droprows=dd.from_pandas(y_train_droprows, npartitions=3) #␣
↪npartitions based on viewing data structure in previous section

# Delete obsolete variables
del x_train_df, y_train_df
```

[21]:
```python
type(x_train_droprows)
```

[21]: pandas.core.frame.DataFrame

1.8 Impute missing data

[15]:
```
#choose here which imputation methods to use:
imputation_methods = ['correlated_columns', 'col_means']

"""

We decided to use col_means as imputation method since we managed to get this␣
↪to work. There is a lot of possible solutions here and hopefully better␣
↪outcomes,
but we chose to focus on what was actually working.

"""
```

[16]:
```
#impute by inserting the mean of the column in question, for all columns:

if 'col_means' in imputation_methods:

    #calculate mean (note that axis needs to be 0 to get columns, which is␣
↪weird)
    miin = x_train_df_droprows.mean(axis = 0).compute() # Fill with mean value

    x_train_df_impute = x_train_df_droprows.fillna(dict(miin))

type(x_train_df_impute)
```

[16]: dask.dataframe.core.DataFrame

[ ]:
```
#impute by inserting value from most closely correlated column:

"""
We have tried to impute by inserting value from closely correlated column.
We can't understand why this one doesn't work. The idea was to fill in NaN's␣
↪from correlated columns.
The problem might be in the last line

"""

#will only correct if the correlation between columns is above this threshold:
lowest_allowed_corr = 0.995

if 'correlated_columns' in imputation_methods:

    #if correlation matrix does not exist, first try to read it from file, if␣
↪that does not work calculate it:
    if dir().count('corrs') == 0:
```

```
        try:
            corrs = pd.read_csv(workdir + 'features_correlation_matrix.txt')
        except:
            corrs = x_train_df.corr('pearson')

    #impute for each feature feat:
    for feat in x_train_df_colnames:

        #order the feature names according to (absolute value of) correlations␣
↪to feat:
        abscorr = [abs(a) for a in list(corrs[feat])]
        order = np.argsort(abscorr)[::-1]
        topfeatures = [x_train_df_colnames[a] for a in order]

        #remove features which are not sufficiently correlated to feat:
        mapper = dict(zip(x_train_df_colnames, abscorr))
        topfeatures = [a for a in topfeatures if mapper[a] >=␣
↪lowest_allowed_corr]

        #correct using each feature in topfeatures, starting with the most␣
↪strongly correlated otherfeature:
        for otherfeat in [a for a in topfeatures if a != feat]:
            print(feat, otherfeat)
            x_train_df_trim[feat] = x_train_df_trim[feat].
↪fillna(x_train_df_trim[otherfeat]).compute()
```

1.9 Build model and train

[82]:
```
# Create dask array with chunks. Delete obsolete variables
X = x_train_df_impute
X = X.to_dask_array(lengths=True)
x_test = x_test_df
x_test = x_test.to_dask_array(lengths=True)
y = y_train_df_droprows
y = y.to_dask_array(lengths=True)
X
#del x_train_df_imean, x_train_df, y_train_df_imean, y_train_df
```

[82]:
```
dask.array<values, shape=(2858793, 162), dtype=float64, chunksize=(11550, 162)>
```

[91]:
```
# Convert blocks in dask array x for new chunks
X = X.rechunk((100000, 162))
y = y.rechunk((100000, 1))
y = y.flatten()
y
```

[91]:
```
dask.array<reshape, shape=(2858793,), dtype=float64, chunksize=(100000,)>
```

```
[93]: X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1)
#del X, y
X_train
```

```
[93]: dask.array<concatenate, shape=(2572913, 162), dtype=float64, chunksize=(90000,
162)>
```

1.9.1 Parallelize Scikit-Learn

```
[ ]: # Model 1: Random forest skikit-learn parallelized

client = Client()  # Connect to a Dask Cluster

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, max_depth=2,random_state=0)
with joblib.parallel_backend('dask'):
    model.fit(X_train, y_train)
    y_true = y_test
    y_pred = model.predict(X_test)


"""

We ran into memory error for the model.predict part of the code

"""
```

1.9.2 Reimplement Scalable Algorithms with Dask Array

```
[122]: # Model 2: Logistic regression Dask style

from dask.diagnostics import ProgressBar

lr = LogisticRegression()
with ProgressBar():
    lr.fit(X_train, y_train)
```

```
[########################################] | 100% Completed | 16.4s
[########################################] | 100% Completed | 58.5s
[########################################] | 100% Completed | 56.3s
[########################################] | 100% Completed | 58.4s
[########################################] | 100% Completed | 55.0s
[########################################] | 100% Completed | 55.7s
[########################################] | 100% Completed | 55.2s
[########################################] | 100% Completed | 53.7s
[########################################] | 100% Completed | 54.0s
[########################################] | 100% Completed | 54.0s
```

```
[########################################] | 100% Completed | 54.4s
[########################################] | 100% Completed | 53.9s
[########################################] | 100% Completed | 55.5s
[########################################] | 100% Completed | 54.0s
[########################################] | 100% Completed | 53.8s
[########################################] | 100% Completed | 53.9s
[########################################] | 100% Completed | 54.0s
[########################################] | 100% Completed | 55.7s
[########################################] | 100% Completed | 54.1s
[########################################] | 100% Completed | 54.1s
[########################################] | 100% Completed | 54.5s
[########################################] | 100% Completed | 54.8s
[########################################] | 100% Completed | 54.6s
[########################################] | 100% Completed | 57.3s
[########################################] | 100% Completed | 55.6s
[########################################] | 100% Completed | 54.9s
[########################################] | 100% Completed | 55.0s
[########################################] | 100% Completed | 55.5s
[########################################] | 100% Completed | 57.5s
[########################################] | 100% Completed | 55.7s
[########################################] | 100% Completed | 55.6s
[########################################] | 100% Completed | 55.5s
[########################################] | 100% Completed | 55.8s
[########################################] | 100% Completed | 56.3s
[########################################] | 100% Completed | 55.7s
[########################################] | 100% Completed | 55.5s
[########################################] | 100% Completed | 55.2s
[########################################] | 100% Completed | 55.4s
[########################################] | 100% Completed | 55.1s
[########################################] | 100% Completed | 55.5s
[########################################] | 100% Completed | 56.9s
[########################################] | 100% Completed | 56.5s
[########################################] | 100% Completed | 55.6s
[########################################] | 100% Completed | 56.8s
[########################################] | 100% Completed | 57.7s
[########################################] | 100% Completed | 54.5s
[########################################] | 100% Completed | 54.1s
[########################################] | 100% Completed | 54.8s
[########################################] | 100% Completed | 53.2s
[########################################] | 100% Completed | 53.9s
[########################################] | 100% Completed | 53.2s
[########################################] | 100% Completed | 54.1s
[########################################] | 100% Completed | 53.1s
[########################################] | 100% Completed | 53.7s
[########################################] | 100% Completed | 51.6s
[########################################] | 100% Completed | 53.3s
[########################################] | 100% Completed | 50.9s
[########################################] | 100% Completed | 52.1s
```

```
[########################################] | 100% Completed | 50.7s
[########################################] | 100% Completed | 50.2s
[########################################] | 100% Completed | 50.3s
[########################################] | 100% Completed | 50.9s
[########################################] | 100% Completed | 49.3s
[########################################] | 100% Completed | 49.6s
[########################################] | 100% Completed | 48.7s
[########################################] | 100% Completed | 48.6s
[########################################] | 100% Completed | 49.0s
[########################################] | 100% Completed | 49.1s
[########################################] | 100% Completed | 50.2s
[########################################] | 100% Completed | 49.1s
[########################################] | 100% Completed | 49.8s
[########################################] | 100% Completed | 48.0s
[########################################] | 100% Completed | 48.1s
[########################################] | 100% Completed | 49.5s
[########################################] | 100% Completed | 46.8s
[########################################] | 100% Completed | 46.9s
[########################################] | 100% Completed | 47.4s
[########################################] | 100% Completed | 46.9s
[########################################] | 100% Completed | 46.6s
[########################################] | 100% Completed | 46.7s
[########################################] | 100% Completed | 48.6s
[########################################] | 100% Completed | 46.6s
[########################################] | 100% Completed | 45.9s
[########################################] | 100% Completed | 46.8s
[########################################] | 100% Completed | 45.7s
[########################################] | 100% Completed | 46.1s
[########################################] | 100% Completed | 47.0s
[########################################] | 100% Completed | 45.7s
[########################################] | 100% Completed | 45.4s
[########################################] | 100% Completed | 46.4s
[########################################] | 100% Completed | 45.4s
[########################################] | 100% Completed | 45.9s
[########################################] | 100% Completed | 45.9s
[########################################] | 100% Completed | 46.8s
[########################################] | 100% Completed | 45.3s
[########################################] | 100% Completed | 45.5s
[########################################] | 100% Completed | 45.4s
[########################################] | 100% Completed | 44.4s
[########################################] | 100% Completed | 44.9s
[########################################] | 100% Completed | 45.8s
[########################################] | 100% Completed | 45.6s
```

[122]:
```
'\n\nTakes a lot of time to compute\n\n'
```

[108]:
```python
# Model 3: Logistic regression Using grid search to tune hyperparameters
```

```
from dask_ml.model_selection import GridSearchCV

parameters = {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10, 100]}

lr = LogisticRegression()
tuned_lr = GridSearchCV(lr, parameters)

with ProgressBar():
    tuned_lr.fit(X_train, y_train)
```

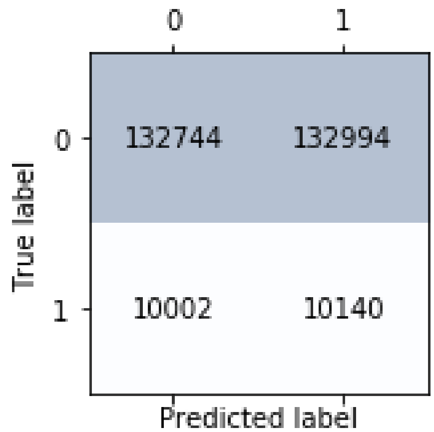
```
[########################################] | 100% Completed | 30min 52.6s
[########################################] | 100% Completed |  1min 53.9s
```

1.9.3 Run diagnostics

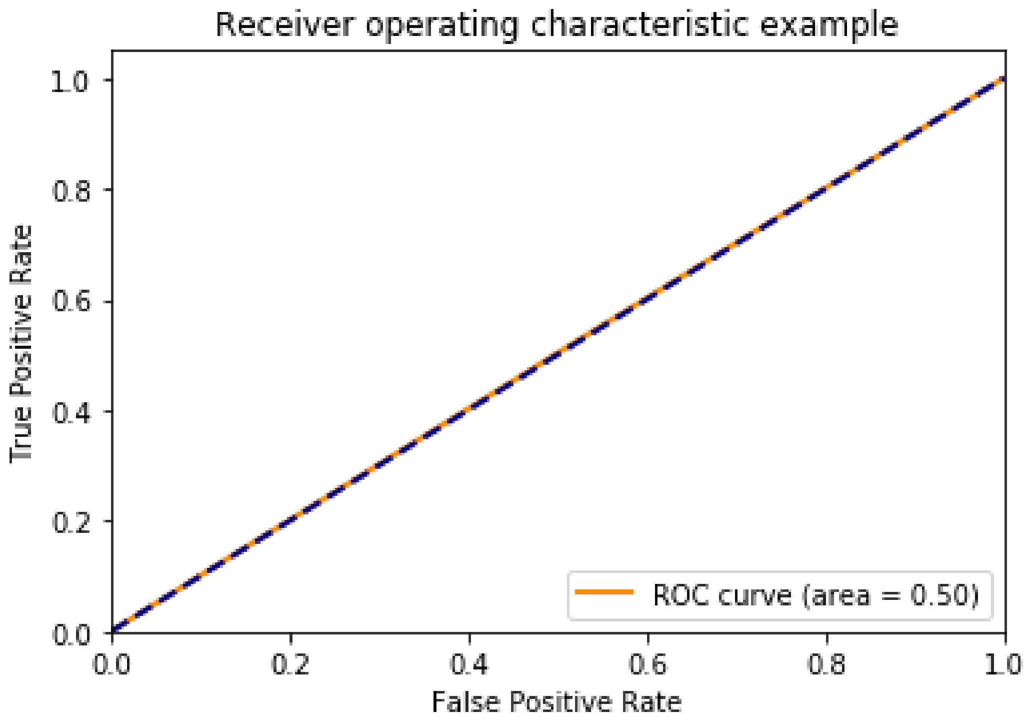
[128]:
```
y_true = y_val
y_pred = tuned_lr.predict(X_val)
accuracy_score(y_true, y_pred)
fpr, tpr, thresholds = metrics.roc_curve(y_true, y_pred)
AUC = metrics.auc(fpr, tpr)
F1 = f1_score(y_true, y_pred)

print (AUC)
print (F1)
```

```
0.5001943914080547
0.1257484708371596
```

[124]:
```
# Confusion matrix of test data
%matplotlib inline
confmat_test = confusion_matrix(y_true, y_pred)
fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat_test, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat_test.shape[0]):
    for j in range(confmat_test.shape[1]):
        ax.text(x=j, y=i, s=confmat_test[i, j], va='center', ha='center')

plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.show()
```



[125]:
```
# Plot the ROC for your best model on the training data
# Compute ROC curve and ROC area for each class
%matplotlib inline
roc_auc=metrics.auc(fpr, tpr)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



1.9.4 tuned_lr using GridSearch gave the best model, and we have submitted that to Kaggle

1.10 Make ready file to be submitted to Kaggle

[127]:
```python
# Predict using model and X_test standardized
y_pred = lr.predict(x_test)
YPRED = y_pred.compute()

np.mean(YPRED)
# Make list of IDs from 0 to 9999
ID = list(range(len(YPRED)))

# Make dataframes from predicted values, IDs and concatenate in result
pred1 = pd.DataFrame(YPRED, index=None, columns=['Target'])
pred2 = pd.DataFrame(ID, index=None, columns=['id'])
result = pd.concat([pred2, pred1], axis=1)
result = result.astype(int) # change type to integers

# Read results to csv for publication on Kaggle
result.to_csv('CA1_16.csv',index=False)
```