

Elements Of Data Science - F2025

Week 10: NLP, Sentiment Analysis and Topic Modeling

11/25/2025

TODOs

- Readings:
 - PML Chapter 11: Working with Unlabeled Data - Clustering Analysis, Sections 11.1 and 11.2
 - [Optional] [PDSH 5.11 k-Means](#)
 - [Optional] [Data Science From Scratch Chap 22: Recommender Systems](#)
- Quiz 9, Due **Mon Dec 2, 11:59pm ET**
- HW3 is Due **TODAY!!**
- HW4 due Dec 9th

Guest Speakers

3 speakers next week

*3 questions (due Monday) + in-person attendance + summary of the talk (1page+ essay) for all guest speakers ==
+5% added to your grade*

Today

- Pipelines
- NLP
- Sentiment Analysis
- Topic Modeling

Questions?

Environment Setup

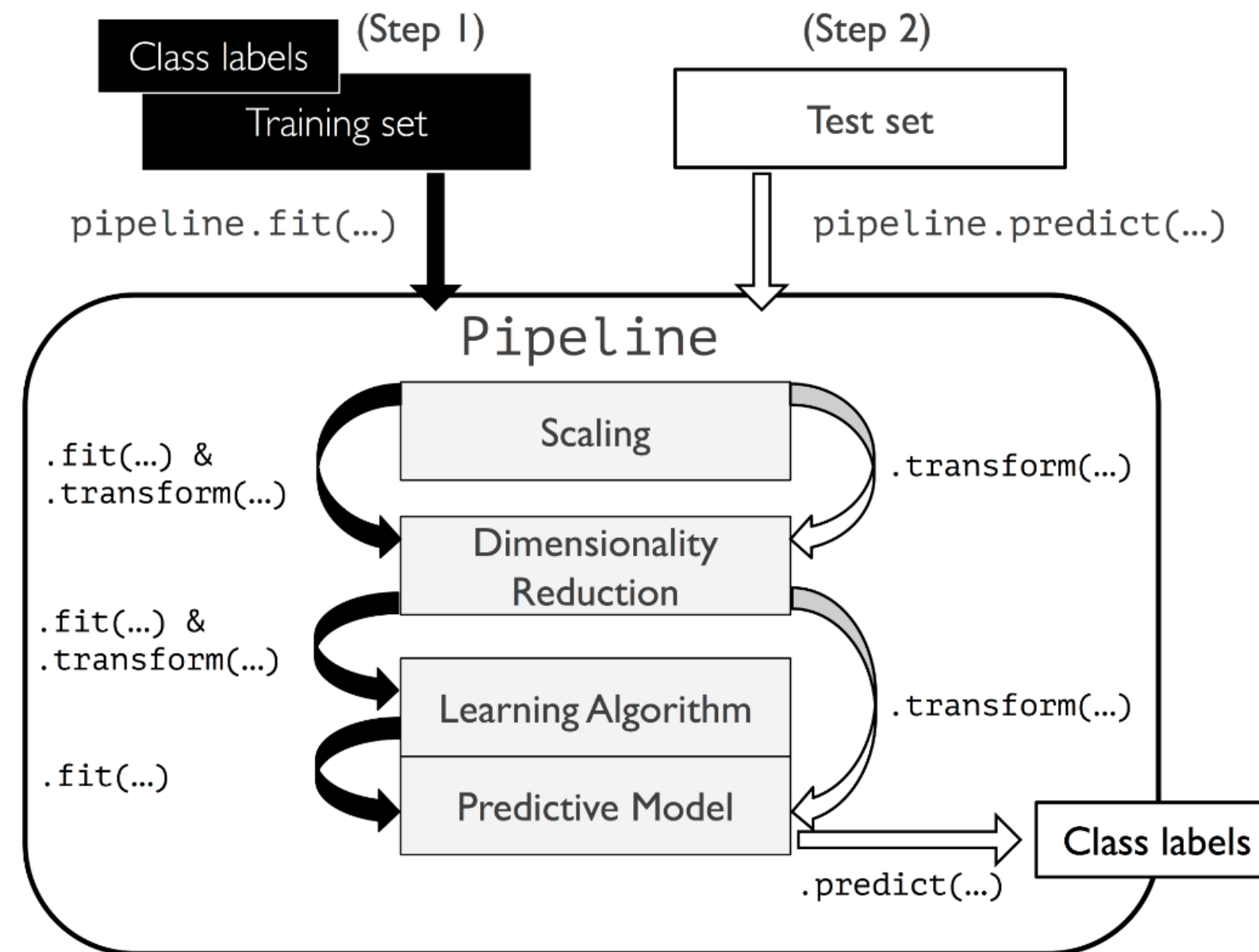
Environment Setup

```
In [1]: 1 import numpy
        2 import numpy as np
        3 import pandas as pd
        4 import matplotlib.pyplot as plt
        5 import seaborn as sns
        6
        7 import warnings
        8 warnings.filterwarnings('ignore')
        9
       10
       11 sns.set_style('darkgrid')
       12 %matplotlib inline
```

Pipelines in sklearn

- Pipelines are wrappers used to string together transformers and estimators
 - sequentially apply a series of transforms, eg, `.fit_transform()` and `.transform()`
 - followed by a prediction, eg. `.fit()` and `.predict()`

Pipelines in sklearn



From PML

Binary Classification With All Numeric Features Setup

Binary Classification With All Numeric Features Setup

```
In [2]: 1 # Example from PML - scaling > feature extraction > classification
2 from sklearn.datasets import load_breast_cancer
3 from sklearn.model_selection import train_test_split
4 bc = load_breast_cancer()
5 X_bc, y_bc = bc['data'], bc['target']
6 X_bc_train, X_bc_test, y_bc_train, y_bc_test = train_test_split(X_bc,
7                                                                 y_bc,
8                                                                 test_size=0.3,
9                                                                 stratify=y_bc,
10                                                                random_state=123)
11
12 # print without scientific notation
13 numpy.set_printoptions(suppress = True)
14
15 print("training set has rows: {} columns: {}".format(*X_bc_train.shape))
16
17 # all real valued features
18 print('Feature names: ', bc.feature_names[:3], ' ...')
19 print('Corresponding Feature values:', X_bc_train[:1, :3][0].round(2), ' ...')
20 print('Target names: ', bc.target_names)
```

```
training set has rows: 398 columns: 30
Feature names: ['mean radius' 'mean texture' 'mean perimeter'] ...
Corresponding Feature values: [12.99 14.23 84.08] ...
Target names: ['malignant' 'benign']
```

Pipelines in sklearn

Pipelines in sklearn

```
In [3]: 1 from sklearn.pipeline import Pipeline
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.decomposition import PCA
4 from sklearn.linear_model import LogisticRegression
5
6 # Pipeline: list of (name,object) pairs
7 pipe1 = Pipeline([('scale',StandardScaler()),           # scale
8                   ('pca',PCA(n_components=15)),          # reduce dimensions
9                   ('lr',LogisticRegression(solver='saga',
10                                           max_iter=1000,
11                                           random_state=12)), # classifier
12                  ])
13
14 pipe1.fit(X_bc_train,y_bc_train)
15
16 print(f'train set accuracy: {pipe1.score(X_bc_train,y_bc_train).round(3)}')
17 print(f'test set accuracy : {pipe1.score(X_bc_test,y_bc_test).round(3)}')
```

```
train set accuracy: 0.987
test set accuracy : 0.982
```

Pipelines in sklearn

```
In [3]: 1 from sklearn.pipeline import Pipeline
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.decomposition import PCA
4 from sklearn.linear_model import LogisticRegression
5
6 # Pipeline: list of (name,object) pairs
7 pipel = Pipeline([('scale',StandardScaler()),           # scale
8                   ('pca',PCA(n_components=15)),         # reduce dimensions
9                   ('lr',LogisticRegression(solver='saga',
10                                           max_iter=1000,
11                                           random_state=12)), # classifier
12                  ])
13
14 pipel.fit(X_bc_train,y_bc_train)
15
16 print(f'train set accuracy: {pipel.score(X_bc_train,y_bc_train).round(3)}')
17 print(f'test set accuracy : {pipel.score(X_bc_test,y_bc_test).round(3)}')
```

train set accuracy: 0.987
test set accuracy : 0.982

```
In [4]: 1 # access pipeline components by name like a dictionary
2 pipel['lr'].coef_.round(2)
```

```
Out[4]: array([[ -2.25,  1.41,  0.43, -0.61,  0.97, -0.08, -0.2 ,  0.7 , -1.47,
                0.06,  0.06, -0.5 ,  0.17,  1.07, -0.57]])
```

Pipelines in sklearn

```
In [3]: 1 from sklearn.pipeline import Pipeline
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.decomposition import PCA
4 from sklearn.linear_model import LogisticRegression
5
6 # Pipeline: list of (name,object) pairs
7 pipel = Pipeline([('scale',StandardScaler()),           # scale
8                   ('pca',PCA(n_components=15)),         # reduce dimensions
9                   ('lr',LogisticRegression(solver='saga',
10                                           max_iter=1000,
11                                           random_state=12)), # classifier
12                ])
13
14 pipel.fit(X_bc_train,y_bc_train)
15
16 print(f'train set accuracy: {pipel.score(X_bc_train,y_bc_train).round(3)}')
17 print(f'test set accuracy : {pipel.score(X_bc_test,y_bc_test).round(3)}')
```

train set accuracy: 0.987
test set accuracy : 0.982

```
In [4]: 1 # access pipeline components by name like a dictionary
2 pipel['lr'].coef_.round(2)
```

```
Out[4]: array([[ -2.25,  1.41,  0.43, -0.61,  0.97, -0.08, -0.2 ,  0.7 , -1.47,
                0.06,  0.06, -0.5 ,  0.17,  1.07, -0.57]])
```

```
In [5]: 1 pipel['pca'].components_[0].round(2)
```

```
Out[5]: array([0.22, 0.08, 0.23, 0.22, 0.14, 0.24, 0.26, 0.26, 0.14, 0.06, 0.21,
                0. , 0.21, 0.2 , 0.01, 0.17, 0.15, 0.18, 0.04, 0.09, 0.23, 0.08,
                0.24, 0.23, 0.13, 0.21, 0.23, 0.25, 0.12, 0.13])
```

Pipelines in sklearn: GridSearch with Pipelines

Pipelines in sklearn: GridSearch with Pipelines

- specify grid points using 'step name' + '___' (double-underscore) + 'argument'

Pipelines in sklearn: GridSearch with Pipelines

- specify grid points using 'step name' + '__' (double-underscore) + 'argument'

```
In [6]: 1 from sklearn.exceptions import ConvergenceWarning # needed to supress warnings
2 from sklearn.utils import parallel_backend # needed to supress warnings
3
4 from sklearn.model_selection import GridSearchCV
5
6 # separate step-names and argument-names with double-underscore '__'
7 params1 = {'pca__n_components':[2,10,15,20],
8            'lr__penalty':['none','l1','l2'],
9            'lr__C':[0,.01,1,10,100]}
10
11 with parallel_backend("multiprocessing"): # needed to supress warnings
12     with warnings.catch_warnings(): # needed to supress warnings
13         warnings.filterwarnings("ignore") # needed to supress warnings
14
15 gscv = GridSearchCV(pipeline, params1, cv=3, n_jobs=-1).fit(X_bc_train,y_bc_train)
16
17 gscv.best_params_
```

```
Out[6]: {'lr__C': 1, 'lr__penalty': 'l1', 'pca__n_components': 15}
```

Pipelines in sklearn: GridSearch with Pipelines

- specify grid points using 'step name' + '__' (double-underscore) + 'argument'

```
In [6]: 1 from sklearn.exceptions import ConvergenceWarning # needed to suppress warnings
        2 from sklearn.utils import parallel_backend        # needed to suppress warnings
        3
        4 from sklearn.model_selection import GridSearchCV
        5
        6 # separate step-names and argument-names with double-underscore '__'
        7 params1 = {'pca__n_components':[2,10,15,20],
        8           'lr__penalty':['none','l1','l2'],
        9           'lr__C':[0,.01,1,10,100]}
        10
        11 with parallel_backend("multiprocessing"):           # needed to suppress warnings
        12     with warnings.catch_warnings():                   # needed to suppress warnings
        13         warnings.filterwarnings("ignore")              # needed to suppress warnings
        14
        15 gscv = GridSearchCV(pipeline, params1, cv=3, n_jobs=-1).fit(X_bc_train,y_bc_train)
        16
        17 gscv.best_params_
```

```
Out[6]: {'lr__C': 1, 'lr__penalty': 'l1', 'pca__n_components': 15}
```

```
In [7]: 1 score = gscv.score(X_bc_test,y_bc_test)
        2 print(f'test set accuracy: {score:0.3f}')
```

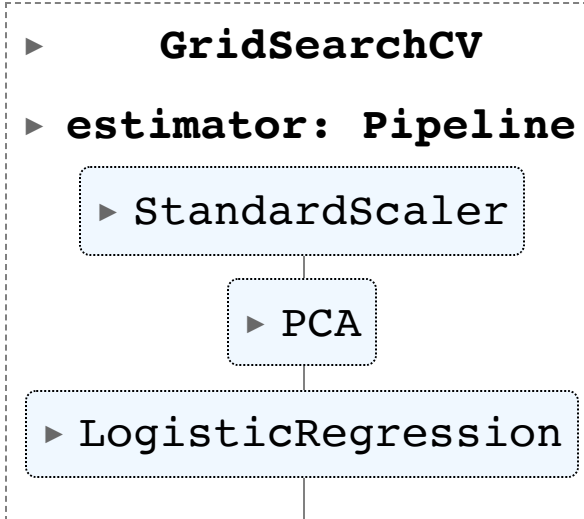
```
test set accuracy: 0.977
```

Displaying Pipelines

Displaying Pipelines

In [8]: 1 gscv

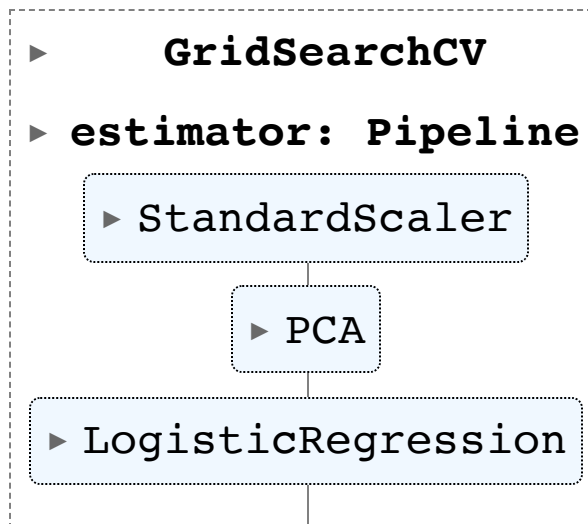
Out[8]:



Displaying Pipelines

```
In [8]: 1 gscv
```

Out[8]:



```
In [9]: 1 print(gscv)
```

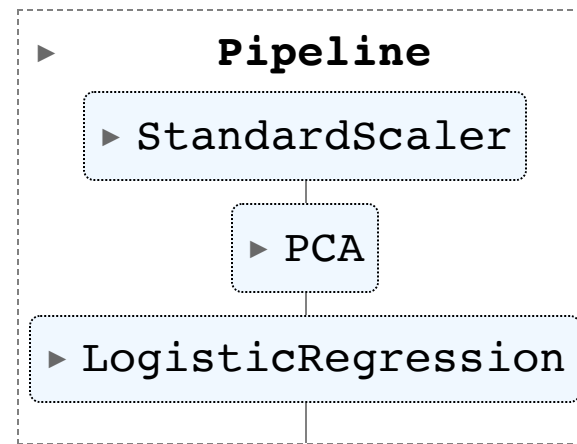
```
GridSearchCV(cv=3,  
            estimator=Pipeline(steps=[('scale', StandardScaler()),  
                                      ('pca', PCA(n_components=15)),  
                                      ('lr',  
                                       LogisticRegression(max_iter=1000,  
                                                           random_state=12,  
                                                           solver='saga')))],  
            n_jobs=-1,  
            param_grid={'lr__C': [0, 0.01, 1, 10, 100],  
                       'lr__penalty': ['none', 'l1', 'l2'],  
                       'pca__n_components': [2, 10, 15, 20]})
```

Displaying Pipelines Cont.

Displaying Pipelines Cont.

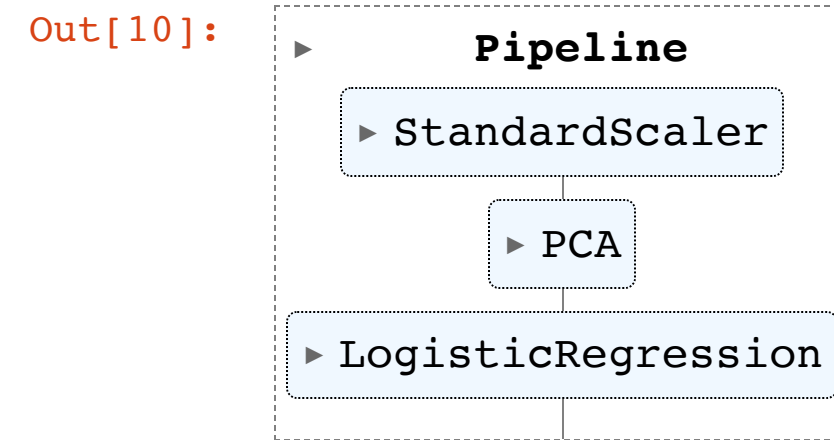
```
In [10]: 1 gscv.best_estimator_
```

Out[10]:



Displaying Pipelines Cont.

```
In [10]: 1 gscv.best_estimator_
```



```
In [11]: 1 print(gscv.best_estimator_)
```

[illegible]

Pipelines in sklearn with `make_pipeline`

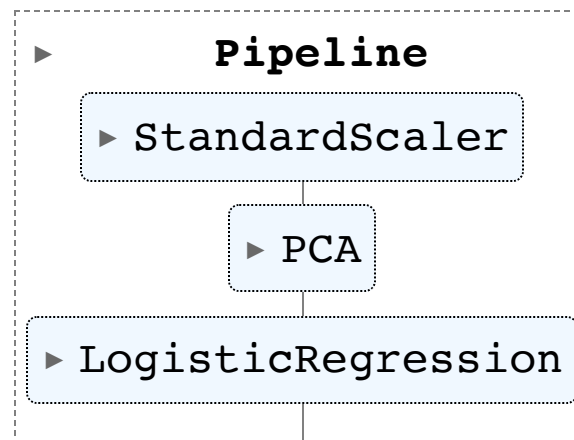
- shorthand for Pipeline
- step names are lowercase of class names

Pipelines in sklearn with `make_pipeline`

- shorthand for Pipeline
- step names are lowercase of class names

```
In [12]: 1 from sklearn.pipeline import make_pipeline
2
3 # make_pipeline: arguments in order of how they should be applied
4 pipe2 = make_pipeline(StandardScaler(),           # center and scale data
5                       PCA(n_components=2),         # extract 2 dimensions
6                       LogisticRegression(random_state=123) # classify using logistic regression
7                       )
8 pipe2.fit(X_bc_train, y_bc_train)
9
10 pipe2
```

Out[12]:

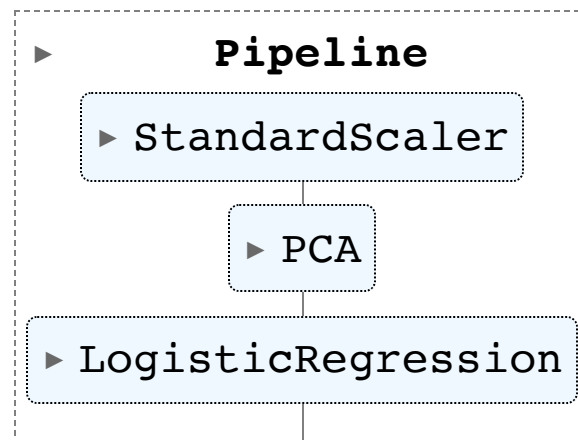


Pipelines in sklearn with `make_pipeline`

- shorthand for Pipeline
- step names are lowercase of class names

```
In [12]: 1 from sklearn.pipeline import make_pipeline
          2
          3 # make_pipeline: arguments in order of how they should be applied
          4 pipe2 = make_pipeline(StandardScaler(),           # center and scale data
          5                     PCA(n_components=2),         # extract 2 dimensions
          6                     LogisticRegression(random_state=123) # classify using logistic regression
          7                     )
          8 pipe2.fit(X_bc_train, y_bc_train)
          9
         10 pipe2
```

Out[12]:



```
In [13]: 1 pipe2['logisticregression'].coef_.round(2)
```

Out[13]: array([[-1.91, 1.04]])

ColumnTransformer

- Transform sets of columns differently as part of a pipeline
- For example: makes it possible to transform categorical and numeric differently

Binary Classification With Mixed Features, Missing Data

Binary Classification With Mixed Features, Missing Data

```
In [14]: 1 # from https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html#sphx-glr-auto-examples-co
2 titanic_url = ('https://raw.githubusercontent.com/amueller/'
3              'scipy-2017-sklearn/091d371/notebooks/datasets/titanic3.csv')
4 df_titanic = pd.read_csv(titanic_url)[['age', 'fare', 'embarked', 'sex', 'pclass', 'survived']]
5 # Numeric Features:
6 # - age: float.
7 # - fare: float.
8 # Categorical Features:
9 # - embarked: categories encoded as strings {'C', 'S', 'Q'}.
10 # - sex: categories encoded as strings {'female', 'male'}.
11 # - pclass: ordinal integers {1, 2, 3}.
12 df_titanic.head(1)
```

Out[14]:

	age	fare	embarked	sex	pclass	survived
0	29.0	211.3375	S	female	1	1

Binary Classification With Mixed Features, Missing Data

```
In [14]: 1 # from https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html#sphx-glr-auto-examples-co
2 titanic_url = ('https://raw.githubusercontent.com/amueller/'
3               'scipy-2017-sklearn/091d371/notebooks/datasets/titanic3.csv')
4 df_titanic = pd.read_csv(titanic_url)[['age', 'fare', 'embarked', 'sex', 'pclass', 'survived']]
5 # Numeric Features:
6 # - age: float.
7 # - fare: float.
8 # Categorical Features:
9 # - embarked: categories encoded as strings {'C', 'S', 'Q'}.
10 # - sex: categories encoded as strings {'female', 'male'}.
11 # - pclass: ordinal integers {1, 2, 3}.
12 df_titanic.head(1)
```

Out[14]:

	age	fare	embarked	sex	pclass	survived
0	29.0	211.3375	S	female	1	1

```
In [15]: 1 df_titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   age         1046 non-null   float64
1   fare        1308 non-null   float64
2   embarked    1307 non-null   object
3   sex         1309 non-null   object
4   pclass      1309 non-null   int64
5   survived    1309 non-null   int64
dtypes: float64(2), int64(2), object(2)
memory usage: 61.5+ KB
```

ColumnTransformer Cont.

ColumnTransformer Cont.

```
In [16]: 1 from sklearn.compose import ColumnTransformer
2 from sklearn.impute import SimpleImputer
3 from sklearn.preprocessing import OneHotEncoder
4
5 # specify columns subset
6 numeric_features = ['age', 'fare']
7 # specify pipeline to apply to those columns
8 numeric_transformer = Pipeline(steps=[
9     ('imputer', SimpleImputer(strategy='median')), # fill missing values with median
10    ('scaler', StandardScaler())])                # scale features
```

ColumnTransformer Cont.

```
In [16]: 1 from sklearn.compose import ColumnTransformer
2 from sklearn.impute import SimpleImputer
3 from sklearn.preprocessing import OneHotEncoder
4
5 # specify columns subset
6 numeric_features = ['age', 'fare']
7 # specify pipeline to apply to those columns
8 numeric_transformer = Pipeline(steps=[
9     ('imputer', SimpleImputer(strategy='median')), # fill missing values with median
10    ('scaler', StandardScaler())])                # scale features
```

```
In [17]: 1 categorical_features = ['embarked', 'sex', 'pclass']
2 categorical_transformer = Pipeline(steps=[
3     ('imputer', SimpleImputer(strategy='constant', fill_value='missing')), # fill missing value with 'missing'
4     ('onehot', OneHotEncoder(handle_unknown='ignore'))])                # one hot encode
```

ColumnTransformer Cont.

```
In [16]: 1 from sklearn.compose import ColumnTransformer
2 from sklearn.impute import SimpleImputer
3 from sklearn.preprocessing import OneHotEncoder
4
5 # specify columns subset
6 numeric_features = ['age', 'fare']
7 # specify pipeline to apply to those columns
8 numeric_transformer = Pipeline(steps=[
9     ('imputer', SimpleImputer(strategy='median')), # fill missing values with median
10    ('scaler', StandardScaler())])                # scale features
```

```
In [17]: 1 categorical_features = ['embarked', 'sex', 'pclass']
2 categorical_transformer = Pipeline(steps=[
3     ('imputer', SimpleImputer(strategy='constant', fill_value='missing')), # fill missing value with 'missing'
4     ('onehot', OneHotEncoder(handle_unknown='ignore'))])                # one hot encode
```

```
In [18]: 1 # combine column pipelines
2 preprocessor = ColumnTransformer(
3     transformers=[('num', numeric_transformer, numeric_features),
4                  ('cat', categorical_transformer, categorical_features)
5                  ])
```

ColumnTransformer Cont.

```
In [16]: 1 from sklearn.compose import ColumnTransformer
2 from sklearn.impute import SimpleImputer
3 from sklearn.preprocessing import OneHotEncoder
4
5 # specify columns subset
6 numeric_features = ['age', 'fare']
7 # specify pipeline to apply to those columns
8 numeric_transformer = Pipeline(steps=[
9     ('imputer', SimpleImputer(strategy='median')), # fill missing values with median
10    ('scaler', StandardScaler())]) # scale features
```

```
In [17]: 1 categorical_features = ['embarked', 'sex', 'pclass']
2 categorical_transformer = Pipeline(steps=[
3     ('imputer', SimpleImputer(strategy='constant', fill_value='missing')), # fill missing value with 'missing'
4     ('onehot', OneHotEncoder(handle_unknown='ignore'))]) # one hot encode
```

```
In [18]: 1 # combine column pipelines
2 preprocessor = ColumnTransformer(
3     transformers=[('num', numeric_transformer, numeric_features),
4                  ('cat', categorical_transformer, categorical_features)
5                 ])
```

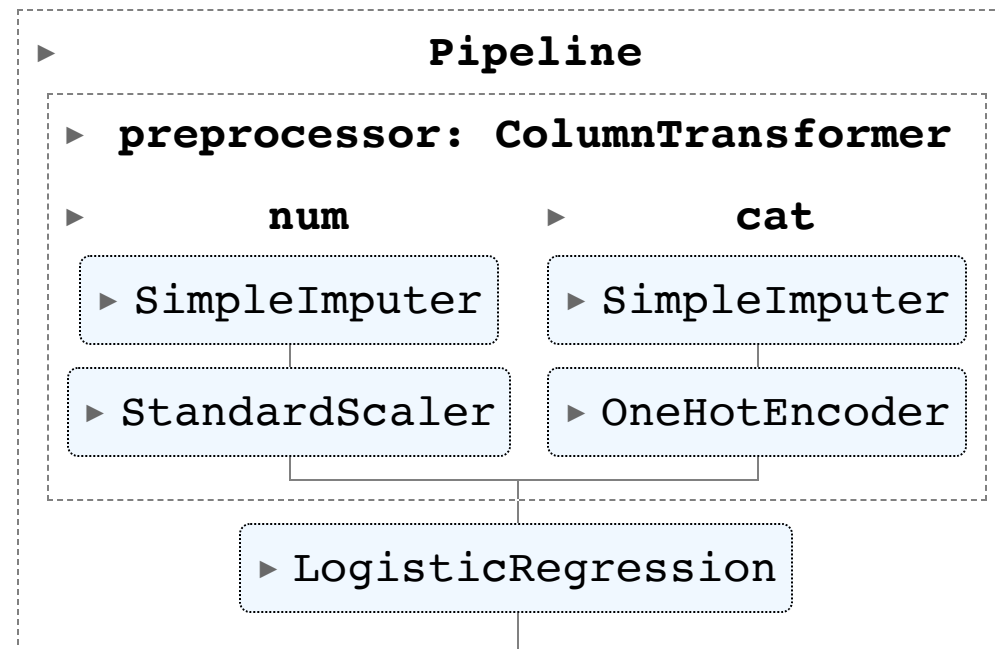
```
In [19]: 1 # add a final prediction step
2 pipe3 = Pipeline(steps=[('preprocessor', preprocessor),
3                          ('classifier', LogisticRegression(solver='lbfgs', random_state=42))
4                          ])
```

ColumnTransformer Cont.

ColumnTransformer Cont.

In [20]: 1 pipe3

Out[20]:



ColumnTransformer Cont.

ColumnTransformer Cont.

```
In [21]: 1 X_titanic = df_titanic.drop('survived', axis=1)
          2 y_titanic = df_titanic['survived']
          3
          4 X_titanic_train, X_titanic_test, y_titanic_train, y_titanic_test = train_test_split(X_titanic,
          5                                                                                   y_titanic,
          6                                                                                   test_size=0.2,
          7                                                                                   random_state=142)
          8 pipe3.fit(X_titanic_train, y_titanic_train)
          9 print(f"train set score: {pipe3.score(X_titanic_train, y_titanic_train).round(3)}")
         10 print(f"test set score : {pipe3.score(X_titanic_test, y_titanic_test).round(3)}")
```

train set score: 0.796

test set score : 0.756

ColumnTransformer Cont.

```
In [21]: 1 X_titanic = df_titanic.drop('survived', axis=1)
2 y_titanic = df_titanic['survived']
3
4 X_titanic_train, X_titanic_test, y_titanic_train, y_titanic_test = train_test_split(X_titanic,
5                                                                                      y_titanic,
6                                                                                      test_size=0.2,
7                                                                                      random_state=142)
8 pipe3.fit(X_titanic_train, y_titanic_train)
9 print(f"train set score: {pipe3.score(X_titanic_train, y_titanic_train).round(3)}")
10 print(f"test set score : {pipe3.score(X_titanic_test, y_titanic_test).round(3)}")
```

```
train set score: 0.796
test set score : 0.756
```

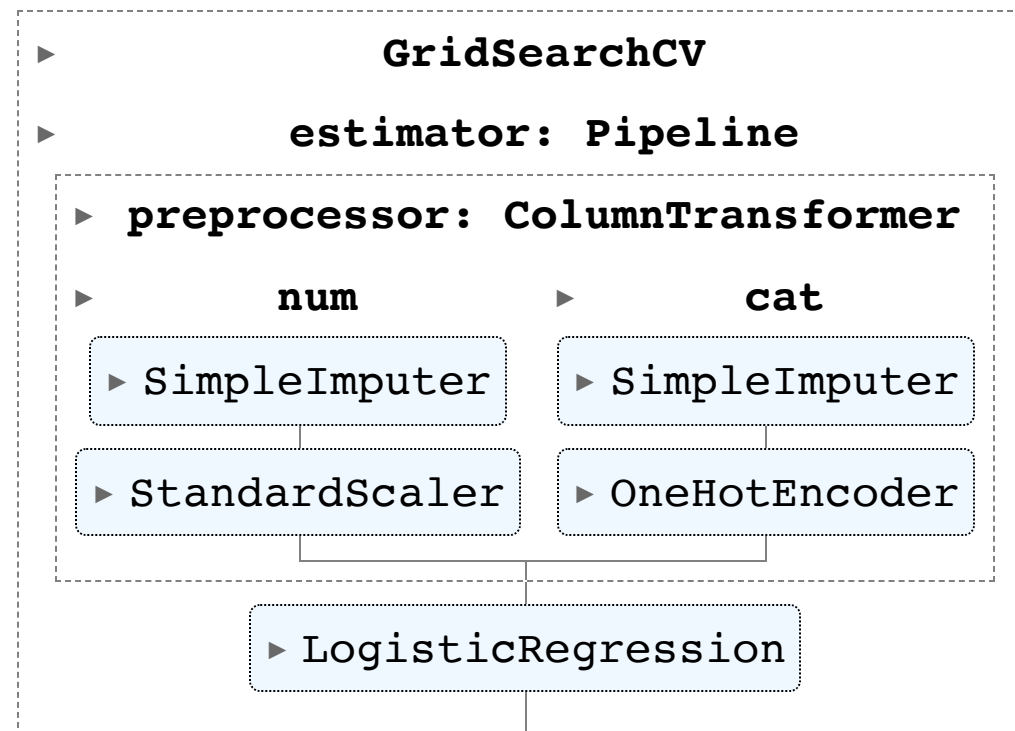
```
In [99]: 1 from sklearn.model_selection import GridSearchCV
2
3 # grid search deep inside the pipeline
4 param_grid = {
5     'preprocessor__num__imputer__strategy': ['mean', 'median'],
6     'classifier__C': [0, 0.1, 1.0, 10, 100],
7 }
8
9 gs_pipeline = GridSearchCV(pipe3, param_grid, cv=10)
10 gs_pipeline.fit(X_titanic_train, y_titanic_train)
11 print(f"best test set score from grid search: {gs_pipeline.score(X_titanic_test, y_titanic_test).round(3)}")
12 print(f"best parameter settings: {gs_pipeline.best_params_}")
```

```
best test set score from grid search: 0.752
best parameter settings: {'classifier__C': 0.1, 'preprocessor__num__imputer__strategy': 'mean'}
```

ColumnTransformer Cont.

In [23]: 1 gs_pipeline

Out[23]:



Questions re Pipelines?

Natural Language Processing (NLP)

- Analyzing and interacting with natural language
- Python Libraries
 - sklearn
 - nltk
 - spaCy
 - gensim
 - ...

Natural Language Processing (NLP)

- Many NLP Tasks
 - **sentiment analysis**
 - **topic modeling**
 - entity detection
 - machine translation
 - natural language generation
 - question answering
 - relationship extraction
 - automatic summarization
 - ...

Recall: Python Builtin String Functions

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```


Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```

```
In [26]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[26]: (['D.S.', 'is', 'fun!'], ['D', 'S', ' is fun!'])
```

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```

```
In [26]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[26]: (['D.S.', 'is', 'fun!'], ['D', 'S', ' is fun!'])
```

```
In [27]: 1 ' | '.join(['ab','c','d'])    # join items in a list together
```

```
Out[27]: 'ab | c | d'
```

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```

```
In [26]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[26]: (['D.S.', 'is', 'fun!'], ['D', 'S', ' is fun!'])
```

```
In [27]: 1 ' | '.join(['ab','c','d'])    # join items in a list together
```

```
Out[27]: 'ab | c | d'
```

```
In [28]: 1 ' | '.join(doc[:5])          # a string itself is treated like a list of characters
```

```
Out[28]: 'D|. |S|. | '
```

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```

```
In [26]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[26]: (['D.S.', 'is', 'fun!'], ['D', 'S', ' is fun!'])
```

```
In [27]: 1 ' | '.join(['ab','c','d'])    # join items in a list together
```

```
Out[27]: 'ab | c | d'
```

```
In [28]: 1 '|'.join(doc[:5])             # a string itself is treated like a list of characters
```

```
Out[28]: 'D|. |S|. | '
```

```
In [29]: 1 ' tes t '.strip()             # remove whitespace from the beginning and end of a string
```

```
Out[29]: 'tes t'
```

Recall: Python Builtin String Functions

```
In [24]: 1 doc = "D.S. is fun!"  
        2 doc
```

```
Out[24]: 'D.S. is fun!'
```

```
In [25]: 1 doc.lower(),doc.upper()      # change capitalization
```

```
Out[25]: ('d.s. is fun!', 'D.S. IS FUN!')
```

```
In [26]: 1 doc.split() , doc.split('.')  # split a string into parts (default is whitespace)
```

```
Out[26]: (['D.S.', 'is', 'fun!'], ['D', 'S', ' is fun!'])
```

```
In [27]: 1 ' | '.join(['ab','c','d'])    # join items in a list together
```

```
Out[27]: 'ab | c | d'
```

```
In [28]: 1 '|'.join(doc[:5])             # a string itself is treated like a list of characters
```

```
Out[28]: 'D|. |S|. | '
```

```
In [29]: 1 ' tes t '.strip()             # remove whitespace from the beginning and end of a string
```

```
Out[29]: 'tes t'
```

- and many more, see <https://docs.python.org/3.10/library/string.html>

NLP: The Corpus

- **corpus:** collection of documents
 - books
 - articles
 - reviews
 - tweets
 - resumes
 - sentences?
 - ...

NLP: Doc Representation

- Documents usually represented as strings
 - string: a sequence (list) of unicode characters

NLP: Doc Representation

- Documents usually represented as strings
 - string: a sequence (list) of unicode characters

```
In [30]: 1 sample_doc = "D.S. is fun!\nIt's  true."  
        2 print(sample_doc)
```

```
D.S. is fun!  
It's  true.
```


NLP: Doc Representation

- Documents usually represented as strings
 - string: a sequence (list) of unicode characters

```
In [30]: 1 sample_doc = "D.S. is fun!\nIt's  true."  
        2 print(sample_doc)
```

```
D.S. is fun!  
It's  true.
```

```
In [31]: 1 '|'.join(sample_doc)
```

```
Out[31]: "D|. |S|. | |i|s| |f|u|n|!|\n|I|t|'|s| | |t|r|u|e|. |"
```

NLP: Doc Representation

- Documents usually represented as strings
 - string: a sequence (list) of unicode characters

```
In [30]: 1 sample_doc = "D.S. is fun!\nIt's  true."  
        2 print(sample_doc)
```

```
D.S. is fun!  
It's  true.
```

```
In [31]: 1 '|'.join(sample_doc)
```

```
Out[31]: "D|. |S|. | |i|s| |f|u|n|!|\n|I|t|'|s| | |t|r|u|e|. |"
```

- Need to split this up into parts (**tokens**)
- Good job for **Regular Expressions**

Aside: Regular Expressions

- Strings that define search patterns over text
- Useful for finding/replacing/grouping
- python `re` library (others available)

Aside: Regular Expressions

- Strings that define search patterns over text
- Useful for finding/replacing/grouping
- python `re` library (others available)

```
In [32]: 1 print(sample_doc)
```

```
D.S. is fun!  
It's true.
```

Aside: Regular Expressions

- Strings that define search patterns over text
- Useful for finding/replacing/grouping
- python `re` library (others available)

```
In [32]: 1 print(sample_doc)
```

```
D.S. is fun!  
It's  true.
```

```
In [33]: 1 import re  
2 # Find all of the whitespaces in doc  
3 # '\s+' means "one or more whitespace characters"  
4 re.findall(r'\s+',sample_doc)
```

```
Out[33]: [' ', ' ', '\n', ' ']
```

Aside: Regular Expressions

Just some of the special character definitions:

- `.` : any single character except newline (r'.' matches 'x')
- `*` : match 0 or more repetitions (r'x*' matches 'x','xx','')
- `+` : match 1 or more repetitions (r'x+' matches 'x','xx')
- `?` : match 0 or 1 repetitions (r'x?' matches 'x' or '')
- `^` : beginning of string (r'^D' matches 'D.S!')
- `$` : end of string (r'fun!\$' matches 'DS is fun!'')

Aside: Regular Expression Cont.

- `[]` : a set of characters (^ as first element = not)
- `\s` : whitespace character (Ex: `[\t\n\r\f\v]`)
- `\S` : non-whitespace character (Ex: `[^ \t\n\r\f\v]`)
- `\w` : word character (Ex: `[a-zA-Z0-9_]`)
- `\W` : non-word character
- `\b` : boundary between `\w` and `\W`
- and many more!
- See regex101.com for examples and testing

Aside: Regex Python Functions

Aside: Regex Python Functions

```
In [34]: 1 r'\w*u\w*' # a string of word characters containing the letter 'u'
```

```
Out[34]: '\\w*u\\w*'
```

Aside: Regex Python Functions

```
In [34]: 1 r'\w*u\w*' # a string of word characters containing the letter 'u'
```

```
Out[34]: '\\w*u\\w*'
```

```
In [35]: 1 re.findall(r'\w*u\w*',sample_doc) # return all substrings that match a pattern
```

```
Out[35]: ['fun', 'true']
```

Aside: Regex Python Functions

```
In [34]: 1 r'\w*u\w*' # a string of word characters containing the letter 'u'
```

```
Out[34]: '\\w*u\\w*'
```

```
In [35]: 1 re.findall(r'\w*u\w*',sample_doc) # return all substrings that match a pattern
```

```
Out[35]: ['fun', 'true']
```

```
In [36]: 1 re.sub(r'\w*u\w*','XXXX',sample_doc) # substitute all substrings that match a pattern
```

```
Out[36]: "D.S. is XXXX!\nIt's  XXXX."
```

Aside: Regex Python Functions

```
In [34]: 1 r'\w*u\w*' # a string of word characters containing the letter 'u'
```

```
Out[34]: '\\w*u\\w*'
```

```
In [35]: 1 re.findall(r'\w*u\w*',sample_doc) # return all substrings that match a pattern
```

```
Out[35]: ['fun', 'true']
```

```
In [36]: 1 re.sub(r'\w*u\w*','XXXX',sample_doc) # substitute all substrings that match a pattern
```

```
Out[36]: "D.S. is XXXX!\nIt's  XXXX."
```

```
In [37]: 1 re.split(r'\w*u\w*',sample_doc) # split substrings on a pattern
```

```
Out[37]: ['D.S. is ', '!\\nIt's ', '.']
```

NLP: Tokenization

- **tokens:** strings that make up a document ('the', 'cat',...)
- **tokenization:** convert a document into tokens
- **vocabulary:** set of unique tokens (terms) in corpus

NLP: Tokenization

- **tokens:** strings that make up a document ('the', 'cat',...)
- **tokenization:** convert a document into tokens
- **vocabulary:** set of unique tokens (terms) in corpus

```
In [38]: 1 # split on whitespace  
        2 re.split(r'\s+', sample_doc)
```

```
Out[38]: ['D.S.', 'is', 'fun!', "It's", 'true.']
```

NLP: Tokenization

- **tokens:** strings that make up a document ('the', 'cat',...)
- **tokenization:** convert a document into tokens
- **vocabulary:** set of unique tokens (terms) in corpus

```
In [38]: 1 # split on whitespace  
        2 re.split(r'\s+', sample_doc)
```

```
Out[38]: ['D.S.', 'is', 'fun!', "It's", 'true.']
```

```
In [39]: 1 # find tokens of length 2+ word characters  
        2 re.findall(r'\b\w\w+\b', sample_doc)
```

```
Out[39]: ['is', 'fun', 'It', 'true']
```

NLP: Tokenization

- **tokens:** strings that make up a document ('the', 'cat',...)
- **tokenization:** convert a document into tokens
- **vocabulary:** set of unique tokens (terms) in corpus

```
In [38]: 1 # split on whitespace  
2 re.split(r'\s+', sample_doc)
```

```
Out[38]: ['D.S.', 'is', 'fun!', "It's", 'true.']
```

```
In [39]: 1 # find tokens of length 2+ word characters  
2 re.findall(r'\b\w\w+\b', sample_doc)
```

```
Out[39]: ['is', 'fun', 'It', 'true']
```

```
In [40]: 1 # find tokens of length 2+ non-space characters  
2 re.findall(r'\b\S\S+\b', sample_doc)
```

```
Out[40]: ['D.S', 'is', 'fun', "It's", 'true']
```


NLP: Tokenization

- **tokens:** strings that make up a document ('the', 'cat',...)
- **tokenization:** convert a document into tokens
- **vocabulary:** set of unique tokens (terms) in corpus

```
In [38]: 1 # split on whitespace  
2 re.split(r'\s+', sample_doc)
```

```
Out[38]: ['D.S.', 'is', 'fun!', "It's", 'true.']
```

```
In [39]: 1 # find tokens of length 2+ word characters  
2 re.findall(r'\b\w\w+\b', sample_doc)
```

```
Out[39]: ['is', 'fun', 'It', 'true']
```

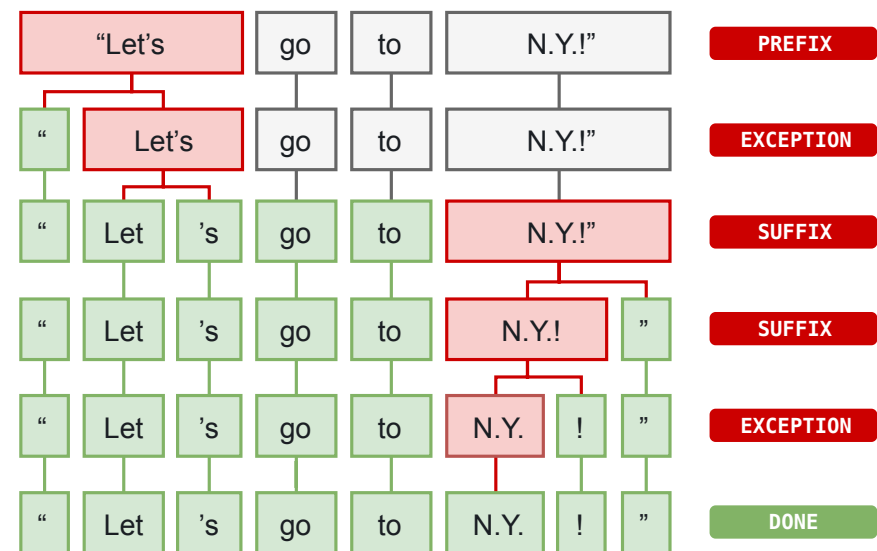
```
In [40]: 1 # find tokens of length 2+ non-space characters  
2 re.findall(r"\b\S\S+\b", sample_doc)
```

```
Out[40]: ['D.S', 'is', 'fun', "It's", 'true']
```

```
In [41]: 1 # example vocabulary  
2 set(re.findall(r"\b\S\S+\b", sample_doc))
```

```
Out[41]: {'D.S', "It's", 'fun', 'is', 'true'}
```

NLP: Tokenization in spaCy



From <https://spacy.io/usage/linguistic-features>

First, the raw text is split on whitespace characters, similar to `text.split(' ')`. Then, the tokenizer processes the text from left to right. On each substring, it performs two checks:

- Does the substring match a tokenizer exception rule? For example, "don't" does not contain whitespace, but should be split into two tokens, "do" and "n't", while "U.K." should always remain one token.
- Can a prefix, suffix or infix be split off? For example punctuation like commas, periods, hyphens or

NLP: Other Options for Preprocessing

- lowercase
- remove special characters
- add `<START>`, `<END>` tags
- **lemmatization:** perform morphological analysis
 - 'studies' becomes 'study'
 - 'studying' becomes 'study'

NLP: Bag of Words

- Bag of Words (BOW) representation: ignore token order

```
In [42]: 1 sample_doc
```

```
Out[42]: "D.S. is fun!\nIt's  true."
```

NLP: Bag of Words

- **Bag of Words (BOW)** representation: ignore token order

```
In [42]: 1 sample_doc
```

```
Out[42]: "D.S. is fun!\nIt's  true."
```

```
In [43]: 1 sample_doc.lower()
```

```
Out[43]: "d.s. is fun!\nit's  true."
```

NLP: Bag of Words

- Bag of Words (BOW) representation: ignore token order

```
In [42]: 1 sample_doc
```

```
Out[42]: "D.S. is fun!\nIt's  true."
```

```
In [43]: 1 sample_doc.lower()
```

```
Out[43]: "d.s. is fun!\nit's  true."
```

```
In [44]: 1 sorted(re.findall(r'\b\S\S+\b', sample_doc.lower()))
```

```
Out[44]: ['d.s', 'fun', 'is', "it's", 'true']
```

NLP: n-Grams

- **Unigram:** single token
- **Bigram:** combination of two ordered tokens
- **n-Gram:** combination of n ordered tokens
- The larger n is, the larger the vocabulary

NLP: n-Grams

- **Unigram:** single token
- **Bigram:** combination of two ordered tokens
- **n-Gram:** combination of n ordered tokens
- The larger n is, the larger the vocabulary

```
In [45]: 1 # Bigram example:
          2 tokens = '<start> ds is fun ds is great <end>'.split()
          3 print("bigrams      : ", [tokens[i]+'_'+tokens[i+1] for i in range(len(tokens)-1)])
          4 print("bigram vocab: ", set([tokens[i]+'_'+tokens[i+1] for i in range(len(tokens)-1)]))

bigrams      : ['<start>_ds', 'ds_is', 'is_fun', 'fun_ds', 'ds_is', 'is_great', 'great_<end>']
bigram vocab: {'is_fun', 'fun_ds', '<start>_ds', 'ds_is', 'great_<end>', 'is_great'}
```


NLP: n-Grams

- **Unigram:** single token
- **Bigram:** combination of two ordered tokens
- **n-Gram:** combination of n ordered tokens
- The larger n is, the larger the vocabulary

```
In [45]: 1 # Bigram example:
          2 tokens = '<start> ds is fun ds is great <end>'.split()
          3 print("bigrams      : ", [tokens[i]+'_'+tokens[i+1] for i in range(len(tokens)-1)])
          4 print("bigram vocab: ", set([tokens[i]+'_'+tokens[i+1] for i in range(len(tokens)-1)]))

bigrams      : ['<start>_ds', 'ds_is', 'is_fun', 'fun_ds', 'ds_is', 'is_great', 'great_<end>']
bigram vocab: {'is_fun', 'fun_ds', '<start>_ds', 'ds_is', 'great_<end>', 'is_great'}
```

```
In [46]: 1 # Trigrams example:
          2 tokens = '<start> ds is fun ds is great <end>'.split()
          3 ['_'.join(tokens[i:i+3]) for i in range(len(tokens)-2)]
```

```
Out[46]: ['<start>_ds_is',
          'ds_is_fun',
          'is_fun_ds',
          'fun_ds_is',
          'ds_is_great',
          'is_great_<end>']
```

NLP: TF and DF

- **Term Frequency:** number of times a term is seen per document
- $\text{tf}(t, d)$ = count of term t in document d

NLP: TF and DF

- **Term Frequency:** number of times a term is seen per document
- $\text{tf}(t, d)$ = count of term t in document d

```
In [47]: 1 example_corpus = ['red green blue', 'red blue blue']  
2  
3 #Vocabulary  
4 example_vocab = sorted(set(' '.join(example_corpus).split()))  
5 example_vocab
```

```
Out[47]: ['blue', 'green', 'red']
```

NLP: TF and DF

- **Term Frequency:** number of times a term is seen per document
- $tf(t, d)$ = count of term t in document d

```
In [47]: 1 example_corpus = ['red green blue', 'red blue blue']
          2
          3 #Vocabulary
          4 example_vocab = sorted(set(' '.join(example_corpus).split()))
          5 example_vocab
```

```
Out[47]: ['blue', 'green', 'red']
```

```
In [48]: 1 #TF
          2 from collections import Counter
          3 example_tf = np.zeros((len(example_corpus), len(example_vocab)))
          4 for i, doc in enumerate(example_corpus):
          5     for j, term in enumerate(example_vocab):
          6         example_tf[i, j] = Counter(doc.split())[term]
          7 example_tf = pd.DataFrame(example_tf, index=['doc1', 'doc2'], columns=example_vocab)
          8 example_tf
```

```
Out[48]:
```

	blue	green	red
doc1	1.0	1.0	1.0
doc2	2.0	0.0	1.0

```
In [49]: 1 doc.split()
```

```
Out[49]: ['red', 'blue', 'blue']
```

```
In [50]: 1 Counter(doc.split())
```

NLP: TF and DF

- **Document Frequency:** number of documents containing each term
 $df(t)$ = count of documents containing term t

NLP: TF and DF

- **Document Frequency:** number of documents containing each term
 $df(t)$ = count of documents containing term t

```
In [51]: 1 example_tf
```

```
Out[51]:
```

	blue	green	red
doc1	1.0	1.0	1.0
doc2	2.0	0.0	1.0

NLP: TF and DF

- **Document Frequency:** number of documents containing each term

$df(t)$ = count of documents containing term t

```
In [51]: 1 example_tf
```

```
Out[51]:
```

	blue	green	red
doc1	1.0	1.0	1.0
doc2	2.0	0.0	1.0

```
In [52]: 1 #DF
2 example_df = example_tf.astype(bool).sum(axis=0) # how many documents contain each term (column)
3 example_df
```

```
Out[52]: blue      2
green     1
red       2
dtype: int64
```

NLP: Stopwords

- terms that have high (or very low) DF and aren't informative
 - common english terms (ex: a, the, in,...)
 - domain specific (ex, in class slides: 'data_science')
 - often removed prior to analysis
 - in sklearn
 - `min_df` : integer > 0 : keep terms that occur in at at least n documents
 - `max_df` : float in (0,1] : keep terms that occur in less than max_df% of total documents

NLP: CountVectorizer in sklearn

NLP: CountVectorizer in sklearn

```
In [53]: 1 example_corpus = ['blue green red', 'blue green green']
          2
          3 from sklearn.feature_extraction.text import CountVectorizer
          4 cvect = CountVectorizer(lowercase=True,      # default, transform all docs to lowercase
          5                        ngram_range=(1,1),  # default, only unigrams
          6                        min_df=1,             # default, keep all terms
          7                        max_df=1.0,           # default, keep all terms
          8                        )
          9 X_cv = cvect.fit_transform(example_corpus)
         10 X_cv.shape
```

```
Out[53]: (2, 3)
```

NLP: CountVectorizer in sklearn

```
In [53]: 1 example_corpus = ['blue green red', 'blue green green']
          2
          3 from sklearn.feature_extraction.text import CountVectorizer
          4 cvect = CountVectorizer(lowercase=True,      # default, transform all docs to lowercase
          5                        ngram_range=(1,1),  # default, only unigrams
          6                        min_df=1,             # default, keep all terms
          7                        max_df=1.0,           # default, keep all terms
          8                        )
          9 X_cv = cvect.fit_transform(example_corpus)
         10 X_cv.shape
```

Out[53]: (2, 3)

```
In [54]: 1 cvect.vocabulary_ # learned vocabulary, term:index pairs
```

Out[54]: {'blue': 0, 'green': 1, 'red': 2}

NLP: CountVectorizer in sklearn

```
In [53]: 1 example_corpus = ['blue green red', 'blue green green']
          2
          3 from sklearn.feature_extraction.text import CountVectorizer
          4 cvect = CountVectorizer(lowercase=True,      # default, transform all docs to lowercase
          5                        ngram_range=(1,1),  # default, only unigrams
          6                        min_df=1,             # default, keep all terms
          7                        max_df=1.0,           # default, keep all terms
          8                        )
          9 X_cv = cvect.fit_transform(example_corpus)
         10 X_cv.shape
```

Out[53]: (2, 3)

```
In [54]: 1 cvect.vocabulary_ # learned vocabulary, term:index pairs
```

Out[54]: {'blue': 0, 'green': 1, 'red': 2}

```
In [55]: 1 cvect.get_feature_names() # vocabulary, sorted by indexs
```

Out[55]: ['blue', 'green', 'red']

NLP: CountVectorizer in sklearn

```
In [53]: 1 example_corpus = ['blue green red', 'blue green green']
          2
          3 from sklearn.feature_extraction.text import CountVectorizer
          4 cvect = CountVectorizer(lowercase=True,      # default, transform all docs to lowercase
          5                        ngram_range=(1,1),  # default, only unigrams
          6                        min_df=1,             # default, keep all terms
          7                        max_df=1.0,           # default, keep all terms
          8                        )
          9 X_cv = cvect.fit_transform(example_corpus)
         10 X_cv.shape
```

Out[53]: (2, 3)

```
In [54]: 1 cvect.vocabulary_ # learned vocabulary, term:index pairs
```

Out[54]: {'blue': 0, 'green': 1, 'red': 2}

```
In [55]: 1 cvect.get_feature_names() # vocabulary, sorted by indexs
```

Out[55]: ['blue', 'green', 'red']

```
In [56]: 1 X_cv.todense() # term frequencies
```

Out[56]: matrix([[1, 1, 1],
 [1, 2, 0]])

NLP: CountVectorizer in sklearn

```
In [53]: 1 example_corpus = ['blue green red', 'blue green green']
          2
          3 from sklearn.feature_extraction.text import CountVectorizer
          4 cvect = CountVectorizer(lowercase=True,      # default, transform all docs to lowercase
          5                        ngram_range=(1,1),  # default, only unigrams
          6                        min_df=1,             # default, keep all terms
          7                        max_df=1.0,          # default, keep all terms
          8                        )
          9 X_cv = cvect.fit_transform(example_corpus)
         10 X_cv.shape
```

Out[53]: (2, 3)

```
In [54]: 1 cvect.vocabulary_ # learned vocabulary, term:index pairs
```

Out[54]: {'blue': 0, 'green': 1, 'red': 2}

```
In [55]: 1 cvect.get_feature_names() # vocabulary, sorted by indexs
```

Out[55]: ['blue', 'green', 'red']

```
In [56]: 1 X_cv.todense() # term frequencies
```

Out[56]: matrix([[1, 1, 1],
 [1, 2, 0]])

```
In [57]: 1 cvect.inverse_transform(X_cv) # mapping back to terms via vocabulary mapping
```

Out[57]: [array(['blue', 'green', 'red'], dtype='<U5'),
 array(['blue', 'green'], dtype='<U5')]

NLP: Tf-Idf

- What if some terms are still uninformative?
- Can we downweight terms that occur in many documents?
- **Term Frequency * Inverse Document Frequency (tf-idf)**
 - $\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$
 - $\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

NLP: Tf-Idf

- What if some terms are still uninformative?
- Can we downweight terms that occur in many documents?
- **Term Frequency * Inverse Document Frequency (tf-idf)**
 - $\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$
 - $\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

```
In [58]: 1 from sklearn.feature_extraction.text import TfidfVectorizer
          2
          3 tfidfvect = TfidfVectorizer() # by default, also doing l2 normalization
          4
          5 x_tfidf = tfidfvect.fit_transform(example_corpus)
          6 sorted(tfidfvect.vocabulary_.items(),key=lambda x: x[1])
```

```
Out[58]: [('blue', 0), ('green', 1), ('red', 2)]
```


NLP: Tf-Idf

- What if some terms are still uninformative?
- Can we downweight terms that occur in many documents?
- **Term Frequency * Inverse Document Frequency (tf-idf)**
 - $\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$
 - $\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

```
In [58]: 1 from sklearn.feature_extraction.text import TfidfVectorizer
          2
          3 tfidfvect = TfidfVectorizer() # by default, also doing l2 normalization
          4
          5 x_tfidf = tfidfvect.fit_transform(example_corpus)
          6 sorted(tfidfvect.vocabulary_.items(),key=lambda x: x[1])
```

```
Out[58]: [('blue', 0), ('green', 1), ('red', 2)]
```

```
In [59]: 1 x_tfidf.todense().round(2)
```

```
Out[59]: array([[0.5 , 0.5 , 0.7 ],
                [0.45, 0.89, 0.  ]])
```

NLP: Tf-Idf

- What if some terms are still uninformative?
- Can we downweight terms that occur in many documents?
- **Term Frequency * Inverse Document Frequency (tf-idf)**
 - $\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$
 - $\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

```
In [58]: 1 from sklearn.feature_extraction.text import TfidfVectorizer
          2
          3 tfidfvect = TfidfVectorizer() # by default, also doing l2 normalization
          4
          5 x_tfidf = tfidfvect.fit_transform(example_corpus)
          6 sorted(tfidfvect.vocabulary_.items(),key=lambda x: x[1])
```

```
Out[58]: [('blue', 0), ('green', 1), ('red', 2)]
```

```
In [59]: 1 x_tfidf.todense().round(2)
```

```
Out[59]: array([[0.5 , 0.5 , 0.7 ],
                [0.45, 0.89, 0.  ]])
```

```
In [60]: 1 # can also use to get term frequencies by setting use_idf to False and norm to none
          2 TfidfVectorizer(use_idf=False, norm=None).fit_transform(example_corpus).todense()
```

```
Out[60]: matrix([[1., 1., 1.],
                 [1., 2., 0.]])
```

NLP: Classification Example

NLP: Classification Example

```
In [61]: 1 from sklearn.datasets import fetch_20newsgroups
2
3 ngs = fetch_20newsgroups(categories=['rec.sport.baseball', 'rec.sport.hockey']) # dataset has 20 categories, only get two
4
5 docs_ngs = ngs['data'] # get documents (emails)
6 y_ngs = ngs['target'] # get targets ([0,1])
7 target_names_ngs = ngs['target_names'] # get target names (['rec.sport.baseball', 'rec.sport.hockey'])
8
9 print(y_ngs[1], target_names_ngs[y_ngs[1]]) # print target int and target name
10 print('-'*50) # print a string of 50 dashes
11 print(docs_ngs[0].strip()[:600]) # print beginning characters of first doc, after stripping whitespace
```

```
1 rec.sport.hockey
```

```
-----
From: dougb@comm.mot.com (Doug Bank)
Subject: Re: Info needed for Cleveland tickets
Reply-To: dougb@ecs.comm.mot.com
Organization: Motorola Land Mobile Products Sector
Distribution: usa
Nntp-Posting-Host: 145.1.146.35
Lines: 17
```

```
In article <1993Apr1.234031.4950@leland.Stanford.EDU>, bohnert@leland.Stanford.EDU (matthew bohnert) writes:
```

```
|> I'm going to be in Cleveland Thursday, April 15 to Sunday, April 18.
|> Does anybody know if the Tribe will be in town on those dates, and
|> if so, who're they playing and if tickets are available?
```

```
The tribe will be in town from April 16 to the 19th.
There
```

NLP Example: Transform Docs

NLP Example: Transform Docs

```
In [62]: 1 from sklearn.model_selection import train_test_split
2 docs_ngs_train, docs_ngs_test, y_ngs_train, y_ngs_test = train_test_split(docs_ngs, y_ngs, random_state = 123)
3
4 vect = TfidfVectorizer(lowercase=True,
5                         min_df=5,           # occur in at least 5 documents
6                         max_df=0.8,        # occur in at most 80% of documents
7                         token_pattern=r'\b\S\S+\b', # tokens of at least 2 non-space characters
8                         ngram_range=(1,1), # only unigrams
9                         use_idf=False,     # term frequency counts instead of tf-idf
10                        norm=None          # do not normalize
11                        )
12 X_ngs_train = vect.fit_transform(docs_ngs_train)
13 X_ngs_train.shape
```

```
Out[62]: (897, 3660)
```

NLP Example: Transform Docs

```
In [62]: 1 from sklearn.model_selection import train_test_split
2 docs_ngs_train, docs_ngs_test, y_ngs_train, y_ngs_test = train_test_split(docs_ngs, y_ngs, random_state = 123)
3
4 vect = TfidfVectorizer(lowercase=True,
5                         min_df=5,           # occur in at least 5 documents
6                         max_df=0.8,        # occur in at most 80% of documents
7                         token_pattern=r'\b\S\S+\b', # tokens of at least 2 non-space characters
8                         ngram_range=(1,1),  # only unigrams
9                         use_idf=False,      # term frequency counts instead of tf-idf
10                        norm=None           # do not normalize
11                        )
12 X_ngs_train = vect.fit_transform(docs_ngs_train)
13 X_ngs_train.shape
```

Out[62]: (897, 3660)

```
In [63]: 1 # first few terms in learned vocabulary
2 list(vect.vocabulary_.items())[:5]
```

Out[63]: [('john', 1781),
('hunter', 1637),
('white', 3550),
('sox', 3059),
('mailing', 2029)]

NLP Example: Transform Docs

```
In [62]: 1 from sklearn.model_selection import train_test_split
2 docs_ngs_train, docs_ngs_test, y_ngs_train, y_ngs_test = train_test_split(docs_ngs, y_ngs, random_state = 123)
3
4 vect = TfidfVectorizer(lowercase=True,
5                         min_df=5,           # occur in at least 5 documents
6                         max_df=0.8,        # occur in at most 80% of documents
7                         token_pattern=r'\b\S\S+\b', # tokens of at least 2 non-space characters
8                         ngram_range=(1,1), # only unigrams
9                         use_idf=False,      # term frequency counts instead of tf-idf
10                        norm=None           # do not normalize
11                        )
12 X_ngs_train = vect.fit_transform(docs_ngs_train)
13 X_ngs_train.shape
```

Out[62]: (897, 3660)

```
In [63]: 1 # first few terms in learned vocabulary
2 list(vect.vocabulary_.items())[:5]
```

Out[63]: [('john', 1781),
('hunter', 1637),
('white', 3550),
('sox', 3059),
('mailing', 2029)]

```
In [64]: 1 # first few terms in learned stopword list
2 list(vect.stop_words_)[:5]
```

Out[64]: ['downec@crockett1a.its.rpi.edu',
'1003.4',
'965.3',
'princeton.edu',
'galley-bowen']

NLP Example: Train and Evaluate Classifier

NLP Example: Train and Evaluate Classifier

```
In [66]: 1 from sklearn.model_selection import cross_val_score
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.dummy import DummyClassifier
4
5 scores_dummy = cross_val_score(DummyClassifier(strategy='most_frequent'),X_ngs_train,y_ngs_train)
6 scores_lr    = cross_val_score(LogisticRegression(),X_ngs_train,y_ngs_train)
7
8 print(f'dummy cv accuracy: {scores_dummy.mean().round(2):0.2f} +- {scores_dummy.std().round(2):0.2f}')
9 print(f'lr      cv accuracy: {scores_lr.mean().round(2):0.2f} +- {scores_lr.std().round(2):0.2f}')
```

```
dummy cv accuracy: 0.52 +- 0.00
lr      cv accuracy: 0.96 +- 0.01
```

NLP Example: Using Pipeline

NLP Example: Using Pipeline

```
In [67]: 1 from sklearn.pipeline import Pipeline
2
3 # Recall: use Pipeline instead of make_pipeline to add names to the steps
4 # (name,object) tuple pairs for each step
5 pipe_ngs1 = Pipeline([('vect',TfidfVectorizer(lowercase=True,
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

pass in docs, not transformed X

```
pipe_ngs1.fit(docs_ngs_train,y_ngs_train)

score_ngs1 = pipe_ngs1.score(docs_ngs_train,y_ngs_train).round(2)
print(f'lr pipeline accuracy on training set: {score_ngs1:0.3f}')
```

```
lr pipeline accuracy on training set: 1.000
```

NLP Example: Using Pipeline

```
In [67]: 1 from sklearn.pipeline import Pipeline
2
3 # Recall: use Pipeline instead of make_pipeline to add names to the steps
4 # (name,object) tuple pairs for each step
5 pipe_ngs1 = Pipeline([('vect',TfidfVectorizer(lowercase=True,
6                                             min_df=5,
7                                             max_df=0.8,
8                                             token_pattern=r'\b\S\S+\b',
9                                             ngram_range=(1,1),
10                                            use_idf=False,
11                                            norm=None )
12                                ),
13                               ('lr',LogisticRegression())
14                           ])
15
16 pipe_ngs1.fit(docs_ngs_train,y_ngs_train) # pass in docs, not transformed X
17
18 score_ngs1 = pipe_ngs1.score(docs_ngs_train,y_ngs_train).round(2)
19 print(f'lr pipeline accuracy on training set: {score_ngs1:0.3f}')
```

lr pipeline accuracy on training set: 1.000

```
In [68]: 1 scores_ngs1 = cross_val_score(pipe_ngs1,docs_ngs_train,y_ngs_train)
2 print(f'lr pipeline cv accuracy: {scores_ngs1.mean().round(2):0.2f} +- {scores_ngs1.std().round(2):0.2f}')
```

lr pipeline cv accuracy: 0.96 +- 0.01

NLP Example: Using Pipeline

```
In [67]: 1 from sklearn.pipeline import Pipeline
2
3 # Recall: use Pipeline instead of make_pipeline to add names to the steps
4 # (name,object) tuple pairs for each step
5 pipe_ngs1 = Pipeline([('vect',TfidfVectorizer(lowercase=True,
6
7
8
9
10
11
12
13
14
15
16 pipe_ngs1.fit(docs_ngs_train,y_ngs_train) # pass in docs, not transformed X
17
18 score_ngs1 = pipe_ngs1.score(docs_ngs_train,y_ngs_train).round(2)
19 print(f'lr pipeline accuracy on training set: {score_ngs1:0.3f}')
```

```
lr pipeline accuracy on training set: 1.000
```

```
In [68]: 1 scores_ngs1 = cross_val_score(pipe_ngs1,docs_ngs_train,y_ngs_train)
2 print(f'lr pipeline cv accuracy: {scores_ngs1.mean().round(2):0.2f} +- {scores_ngs1.std().round(2):0.2f}')
```

```
lr pipeline cv accuracy: 0.96 +- 0.01
```

```
In [101]: 1 list(pipe_ngs1['vect'].get_feature_names_out())[2100:2110]
```

```
Out[101]: ['mediocre',
            'meet',
            'mel',
            'mellon',
```

NLP Example: Add Feature Selection

NLP Example: Add Feature Selection

```
In [70]: 1 from sklearn.feature_selection import SelectFromModel, SelectPercentile
2
3 pipe_ngs2 = Pipeline([('vect', TfidfVectorizer(lowercase=True,
4                                                min_df=5,
5                                                max_df=0.8,
6                                                token_pattern='\\b\\s\\s+\\b',
7                                                ngram_range=(1,1),
8                                                use_idf=False,
9                                                norm=None )
10
11                                                ),
12                                                ('fs', SelectFromModel(estimator=LogisticRegression(C=1.0,
13                                                                 penalty='l1',
14                                                                 solver='liblinear',
15                                                                 max_iter=1000,
16                                                                 random_state=123
17                                                                 ))),
18                                                ('lr', LogisticRegression(max_iter=10000))
19                                            ])
20 pipe_ngs2.fit(docs_ngs_train, y_ngs_train)
21 print(f'pipeline accuracy on training set: {pipe_ngs2.score(docs_ngs_train, y_ngs_train).round(2):0.2f}')
22
23 scores_ngs2 = cross_val_score(pipe_ngs2, docs_ngs_train, y_ngs_train)
24 print(f'pipeline cv accuracy          : {scores_ngs2.mean().round(2):0.2f} +- {scores_ngs2.std().round(2):0.2f}')
```

```
pipeline accuracy on training set: 1.00
pipeline cv accuracy          : 0.94 +- 0.02
```


NLP Example: Grid Search with Feature Selection

NLP Example: Grid Search with Feature Selection

In [71]:

```
1 %%time
2 # NOTE: this may take a minute or so
3 params_ngs2 = {'vect__use_idf':[True,False],
4               'vect__ngram_range':[(1,1),(2,2)],
5               'fs__estimator__C':[10,1000],
6               'lr__C': [.01,1,100]}
7
8 gscv_ngs = GridSearchCV(pipe_ngs2, params_ngs2, cv=2, n_jobs=-1).fit(docs_ngs_train,y_ngs_train)
9
10 print(f'gscv_ngs best parameters : {gscv_ngs.best_params_}')
11 print(f'gscv_ngs best cv accuracy : {gscv_ngs.best_score_.round(2):0.2f}')
12 print(f'gscv_ngs test set accuracy: {gscv_ngs.score(docs_ngs_test,y_ngs_test).round(2):0.2f}')
```

```
gscv_ngs best parameters : {'fs__estimator__C': 1000, 'lr__C': 0.01, 'vect__ngram_range': (1, 1), 'vect__use_idf': True}
gscv_ngs best cv accuracy : 0.96
gscv_ngs test set accuracy: 0.97
CPU times: user 319 ms, sys: 29.3 ms, total: 348 ms
Wall time: 45.8 s
```

Sentiment Analysis and sklearn

- determine sentiment/opinion from unstructured text
 - usually positive/negative, but is domain specific
 - can be treated as a classification task (with a target, using all of the tools we know)
 - can also be treated as a linguistic task (sentence parsing)
-
- Example: determine sentiment of movie reviews
 - see [sentiment_analysis_example.ipynb](#)

Topic Modeling

- What topics are our documents composed of?
- How much of each topic does each document contain?
- Can we represent documents using topic weights? (dimensionality reduction!)

Topic Modeling

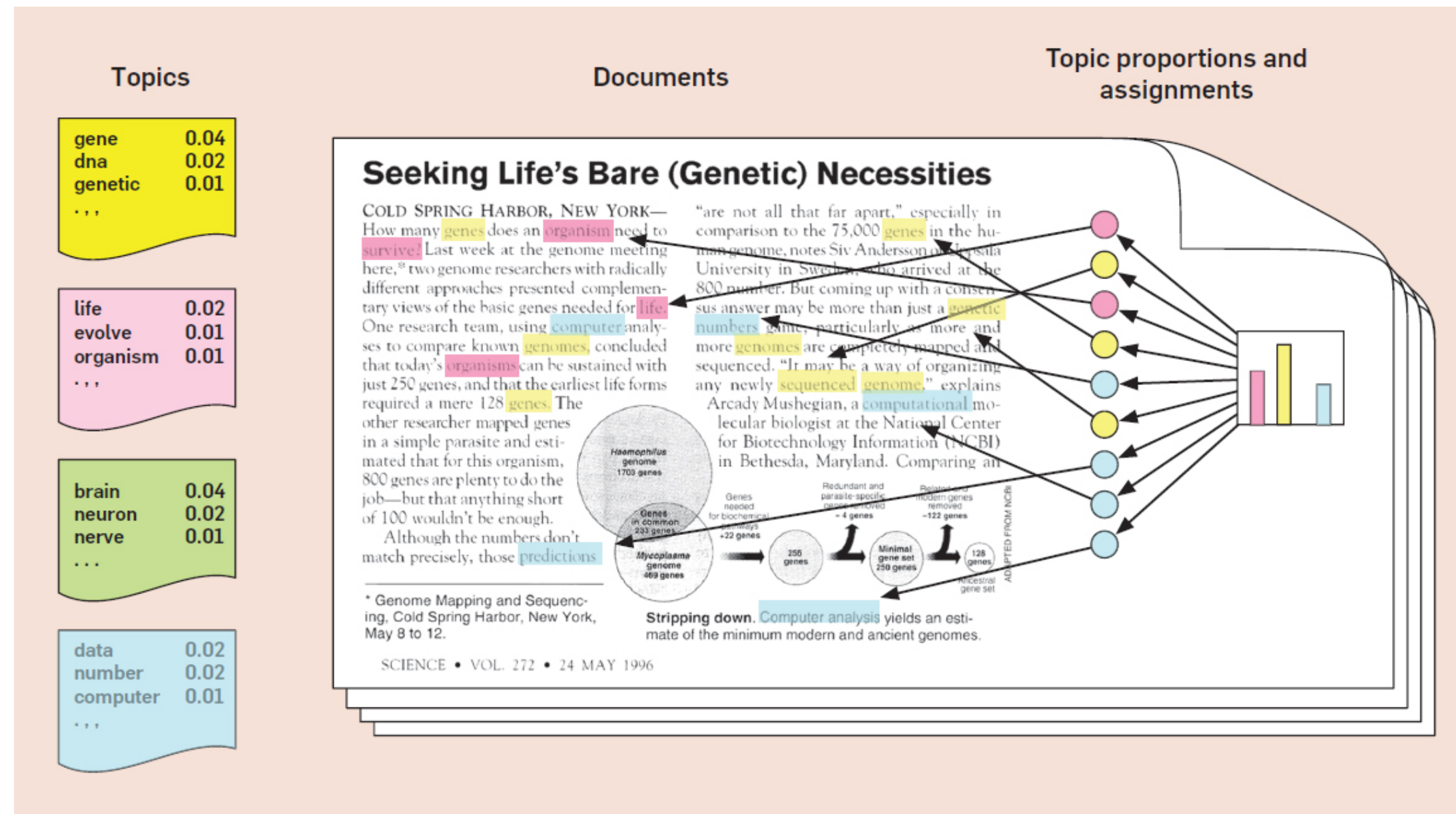
- What topics are our documents composed of?
 - How much of each topic does each document contain?
 - Can we represent documents using topic weights? (dimensionality reduction!)
-
- What is topic modeling?
 - How does **Latent Dirichlet Allocation (LDA)** work?
 - How to train and use LDA with sklearn?

What is Topic Modeling?

- **topic:** a collection of related words
- A document can be composed of several topics
- Given a collection of documents, we can ask:
 - **What terms make up each topic?** (per topic term distribution)
 - **What topics make up each document?** (per document topic distribution)

Topic Modeling with Latent Dirichlet Allocation (LDA)

- Unsupervised method for determining topics and topic assignments



Two Important Matrices Learned by LDA

- the per topic term distributions aka φ (phi)

Two Important Matrices Learned by LDA

- the per topic term distributions aka φ (phi)

```
In [72]: 1 topics = ['topic1', 'topic2']
          2 vocab = ['cat', 'baseball', 'play']
          3 phi = pd.DataFrame([[0.4, .2, .4], [0.2, .4, .4]], columns=vocab, index=topics)
          4 phi
```

Out[72]:

	cat	baseball	play
topic1	0.4	0.2	0.4
topic2	0.2	0.4	0.4

Two Important Matrices Learned by LDA

- the per topic term distributions aka φ (phi)

```
In [72]: 1 topics = ['topic1', 'topic2']
          2 vocab = ['cat', 'baseball', 'play']
          3 phi = pd.DataFrame([[0.4, .2, .4], [0.2, .4, .4]], columns=vocab, index=topics)
          4 phi
```

Out[72]:

	cat	baseball	play
topic1	0.4	0.2	0.4
topic2	0.2	0.4	0.4

- the per document TOPIC distributions aka θ (theta)

Two Important Matrices Learned by LDA

- the per topic term distributions aka φ (phi)

```
In [72]: 1 topics = ['topic1', 'topic2']
          2 vocab = ['cat', 'baseball', 'play']
          3 phi = pd.DataFrame([[0.4, .2, .4], [0.2, .4, .4]], columns=vocab, index=topics)
          4 phi
```

Out[72]:

	cat	baseball	play
topic1	0.4	0.2	0.4
topic2	0.2	0.4	0.4

- the per document TOPIC distributions aka θ (theta)

```
In [73]: 1 topics = ['topic1', 'topic2']
          2 docs = ['doc1', 'doc2']
          3 theta = pd.DataFrame([[0.1, .9], [.5, .5]], columns=topics, index=docs)
          4 theta
```

Out[73]:

	topic1	topic2
doc1	0.1	0.9
doc2	0.5	0.5

Topic Modeling: Example

- Given the data and the number of topics we want

Topic Modeling: Example

- Given the data and the number of topics we want

```
In [74]: 1 corpus = ['the dog and cat played tennis',  
2             'tennis and baseball are sports',  
3             'a dog or a cat can be a pet']  
4  
5 M = 3 # the number of documents  
6  
7 vocab = ['baseball', 'cat', 'dog', 'pet', 'played', 'tennis']  
8  
9 V = len(vocab) # size of vocabulary  
10  
11 K = 2 # our guess about the number of topics  
12  
13 print(f'{M = :}\n{V = :}\n{K = :}')
```

```
M = 3  
V = 6  
K = 2
```

Topic Modeling: Example

- Guessing some **per topic term distributions** (φ) given the documents and vocab

Topic Modeling: Example

- Guessing some **per topic term distributions** (φ) given the documents and vocab

```
In [75]: 1 print(vocab)
```

```
['baseball', 'cat', 'dog', 'pet', 'played', 'tennis']
```

Topic Modeling: Example

- Guessing some per topic term distributions (φ) given the documents and vocab

In [75]:

```
1 print(vocab)
```

```
['baseball', 'cat', 'dog', 'pet', 'played', 'tennis']
```

In [76]:

```
1 # the probability of each term given topic 1 (high for sports terms)
2 topic_1 = [.33, 0, 0, 0, .33, .33]
3
4 # the probability of each term given topic 2 (high for pet terms)
5 topic_2 = [ 0, .25, .25, .25, .25, 0]
6
7 # per topic term distributions
8 phi = pd.DataFrame([topic_1, topic_2], columns=vocab,
9                     index=['topic_'+str(x) for x in range(1, K+1)])
10
11 phi
```

Out[76]:

	baseball	cat	dog	pet	played	tennis
topic_1	0.33	0.00	0.00	0.00	0.33	0.33
topic_2	0.00	0.25	0.25	0.25	0.25	0.00

Topic Modeling: Example

- Guessing the per document topic distributions θ given the topics

Topic Modeling: Example

- Guessing the per document topic distributions θ given the topics

```
In [77]: 1 # Given our guess about phi
         2 display(phi)
         3 # And the corpus
         4 corpus
```

	baseball	cat	dog	pet	played	tennis
topic_1	0.33	0.00	0.00	0.00	0.33	0.33
topic_2	0.00	0.25	0.25	0.25	0.25	0.00

```
Out[77]: ['the dog and cat played tennis',
          'tennis and baseball are sports',
          'a dog or a cat can be a pet']
```

Topic Modeling: Example

- Guessing the per document topic distributions θ given the topics

```
In [77]: 1 # Given our guess about phi
         2 display(phi)
         3 # And the corpus
         4 corpus
```

	baseball	cat	dog	pet	played	tennis
topic_1	0.33	0.00	0.00	0.00	0.33	0.33
topic_2	0.00	0.25	0.25	0.25	0.25	0.00

```
Out[77]: ['the dog and cat played tennis',
          'tennis and baseball are sports',
          'a dog or a cat can be a pet']
```

```
In [78]: 1 # generate a guess about per document topic distributions
         2 theta = pd.DataFrame([[.50, .50],
         3                        [.99, .01],
         4                        [.01, .99]],
         5                        columns=['topic_'+str(x) for x in range(1,K+1)],
         6                        index=['doc_'+str(x) for x in range(1,M+1)])
         7 theta
```

```
Out[78]:
```

	topic_1	topic_2
doc_1	0.50	0.50
doc_2	0.99	0.01
doc_3	0.01	0.99

Topic Modeling With LDA

- Given
 - a set of documents
 - a number of topics K
- Learn
 - the per topic term distributions φ (phi), size: $K \times V$
 - the per document topic distributions θ (theta), size: $M \times K$
- How to learn φ and θ :
 - Latent Dirichlet Allocation (LDA)
 - generative statistical model
 - Blei, D., Ng, A., Jordan, M. Latent Dirichlet allocation. J. Mach. Learn. Res. 3 (Jan 2003)

Topic Modeling With LDA

- Uses for φ (phi), the per topic term distributions:
 - inferring labels for topics
 - word clouds
- Uses for θ (theta), the per document topic distributions:
 - dimensionality reduction
 - clustering
 - similarity

LDA with sklearn

LDA with sklearn

```
In [79]: 1 # load data from all 20 newsgroups
          2 newsgroups = fetch_20newsgroups()
          3 ngs_all = newsgroups.data
          4 len(ngs_all)
```

```
Out[79]: 11314
```

LDA with sklearn

```
In [79]: 1 # load data from all 20 newsgroups
2 newsgroups = fetch_20newsgroups()
3 ngs_all = newsgroups.data
4 len(ngs_all)
```

Out[79]: 11314

```
In [80]: 1 # transform documents using tf-idf
2 tfidf = TfidfVectorizer(token_pattern=r'\b[a-zA-Z0-9-][a-zA-Z0-9-]+\b',min_df=50, max_df=.2)
3 X_tfidf = tfidf.fit_transform(ngs_all)
4 X_tfidf.shape
5
6 tf_idf_array = X_tfidf.toarray()
```

```
In [81]: 1 tf_idf_array[90:100,-10:]
```

```
Out[81]: array([[0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.17079238, 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.17000043, 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.]])
```


LDA with sklearn

```
In [79]: 1 # load data from all 20 newsgroups
          2 newsgroups = fetch_20newsgroups()
          3 ngs_all = newsgroups.data
          4 len(ngs_all)
```

Out[79]: 11314

```
In [80]: 1 # transform documents using tf-idf
2 tfidf = TfidfVectorizer(token_pattern=r'\b[a-zA-Z0-9-][a-zA-Z0-9-]+\b',min_df=50, max_df=.2)
3 X_tfidf = tfidf.fit_transform(ngs_all)
4 X_tfidf.shape
5
6 tf_idf_array = X_tfidf.toarray()
```

```
In [81]: 1 tf_idf_array[90:100, -10:]
```

```
Out[81]: array([[0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0.17079238],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0.17000043],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ],
                [0., 0., 0., 0., 0., ]])
```

LDA with sklearn Cont.

LDA with sklearn Cont.

```
In [83]: 1 from sklearn.decomposition import LatentDirichletAllocation
2
3 # create model with 20 topics
4 lda = LatentDirichletAllocation(n_components=20, # the number of topics
5                                n_jobs=-1,       # use all cpus
6                                random_state=123) # for reproducibility
7
8 # learn phi (lda.components_) and theta (X_lda)
9 # this will take a while!
10 X_lda = lda.fit_transform(X_tfidf)
```

LDA with sklearn Cont.

```
In [83]: 1 from sklearn.decomposition import LatentDirichletAllocation
          2
          3 # create model with 20 topics
          4 lda = LatentDirichletAllocation(n_components=20, # the number of topics
          5                               n_jobs=-1,         # use all cpus
          6                               random_state=123) # for reproducibility
          7
          8 # learn phi (lda.components_) and theta (X_lda)
          9 # this will take a while!
         10 X_lda = lda.fit_transform(X_tfidf)
```

```
In [84]: 1 ngs_all[100][:1000]
```

```
Out[84]: 'From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software forsale (lots) **\nNntp-Posting-Host: magnusug.ma\n          gnus.acs.ohio-state.edu\nOrganization: The Ohio State University\n          **** This is a post for my friend, You can either call\n          ****\n          **** him J.K Lee (614)791-0748 or Drop me a mail ****\nDistribution: usa\nLines: 39\n1. Software publi\nshing SuperBase 4 windows v.1.3 --->$80\n2. OCR System ReadRight v.3.1 for Windows --->$65\n3. OCR System ReadRight v.2.01 for DOS --->$65\n4. Unregistered Zortech 32 bit C++ Compiler v.3.1 --->$250\n5. with Multiscope windows Debugger,\nWhiteWater Resource Toolkit, Library Source Code\n6. Glockenspiel/Im\nageSoft Commonview 2 Windows\nApplications Framework for Borland C++ --->$70\n7. Spontaneous Assembly L\nibrary With Source Code --->$50\n8. Microsoft Macro Assembly 6.0 --->$50\nft Win'
```

LDA with sklearn Cont.

```
In [83]: 1 from sklearn.decomposition import LatentDirichletAllocation
2
3 # create model with 20 topics
4 lda = LatentDirichletAllocation(n_components=20, # the number of topics
5                                n_jobs=-1,       # use all cpus
6                                random_state=123) # for reproducibility
7
8 # learn phi (lda.components_) and theta (X_lda)
9 # this will take a while!
10 X_lda = lda.fit_transform(X_tfidf)
```

```
In [84]: 1 ngs_all[100][:1000]
```

```
Out[84]: 'From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software forsale (lots) **\nNntp-Posting-Host: magnusug.ma\n          gnus.acs.ohio-state.edu\nOrganization: The Ohio State University\n          **** This is a post for my friend, You can either call\n          ****\n          **** him J.K Lee (614)791-0748 or Drop me a mail ****\nDistribution: usa\nLines: 39\n1. Software publi\nshing SuperBase 4 windows v.1.3 --->$80\n2. OCR System ReadRight v.3.1 for Windows --->$65\n3. OCR System ReadRight v.2.01 for DOS --->$65\n4. Unregistered Zortech 32 bit C++ Compiler v.3.1\n--->$ 250\nwith Multiscope windows Debugger,\nWhiteWater Resource Toolkit, Library Source Code\n5. Glockenspiel/Im\nageSoft Commonview 2 Windows\nApplications Framework for Borland C++ --->$70\n6. Spontaneous Assembly L\nibrary With Source Code --->$50\n7. Microsoft Macro Assembly 6.0 --->$50\n8. Microso\nft Win'
```

```
In [85]: 1 X_lda[100].round(2) # lda representation of document_100
```

```
Out[85]: array([0.01, 0.01, 0.01, 0.01, 0.08, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,\n                0.01, 0.01, 0.01, 0.79, 0.01, 0.01, 0.01, 0.01, 0.01])
```

LDA with sklearn Cont.

```
In [83]: 1 from sklearn.decomposition import LatentDirichletAllocation
          2
          3 # create model with 20 topics
          4 lda = LatentDirichletAllocation(n_components=20, # the number of topics
          5                                n_jobs=-1,         # use all cpus
          6                                random_state=123) # for reproducibility
          7
          8 # learn phi (lda.components_) and theta (X_lda)
          9 # this will take a while!
         10 X_lda = lda.fit_transform(X_tfidf)
```

```
In [84]: 1 ngs_all[100][:1000]
```

```
Out[84]: 'From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software forsale (lots) **\nNntp-Posting-Host: magnusug.ma
gnus.acs.ohio-state.edu\nOrganization: The Ohio State University\n      **** This is a post for my friend, You can either call
****\n      **** him J.K Lee (614)791-0748 or Drop me a mail ****\nDistribution: usa\nLines: 39\n\n1. Software publi
shing SuperBase 4 windows v.1.3 --->$80\n\n2. OCR System ReadRight v.3.1 for Windows --->$65\n\n3.
OCR System ReadRight v.2.01 for DOS --->$65\n\n4. Unregistered Zortech 32 bit C++ Compiler v.3.1
--->$ 250\n      with Multiscope windows Debugger,\n      WhiteWater Resource Toolkit, Library Source Code\n\n5. Glockenspiel/Im
ageSoft Commonview 2 Windows\n      Applications Framework for Borland C++ --->$70\n\n6. Spontaneous Assembly L
ibrary With Source Code --->$50\n\n7. Microsoft Macro Assembly 6.0 --->$50\n\n8. Microso
ft Win'
```

```
In [85]: 1 X_lda[100].round(2) # lda representation of document_100
```

```
Out[85]: array([0.01, 0.01, 0.01, 0.01, 0.08, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
                0.01, 0.01, 0.01, 0.79, 0.01, 0.01, 0.01, 0.01, 0.01])
```

```
In [86]: 1 # Note: since this is unsupervised, these numbers may change
          2 np.argsort(X_lda[100])[:, -1]#[ :3] # the top topics of document_100
```

```
Out[86]: array([14,  4, 11, 16,  2, 15,  7,  1,  9,  8, 18, 19,  6, 12, 10,  5,  0,
                17, 13,  3])
```

LDA: Per Topic Term Distributions

LDA: Per Topic Term Distributions

```
In [88]: 1 print_top_words(lda,feature_names,20)
```

Topic 0: udel rpi princeton phoenix delaware tie abc coverage record cancer gore terminal devils islanders nsa batf scott terr
orist fire fan

Topic 1: turkish armenian armenians armenia serdar argic turks turkey zuma sera genocide sdpa uucp greek extermination utexas
davidian 1920 soviet 1919

Topic 2: god his jesus because why us say believe israel such those should many our being said even did see these

Topic 3: pitt geb gordon banks cadre cs intellect shameful chastity n3jxp gregg dsl skepticism surrender pittsburgh soon univ
science too computer

Topic 4: cwru cleveland ohio-state magnus acs freenet ohio reserve ins western po uci case usa orion oac sam colostate irvine
state

Topic 5: rit isc rochester ring testing harris tape waterloo donald uwaterloo hello ca frequent edit effects print intend king
incorrect sunos

Topic 6: psu psuvm penn gay men homosexual sexual male holland partners study cramer percent acns percentage sex optilink repo
rt straight engaged

Topic 7: caltech keith sgi livesey cco wpd solntze ibm schneider jon pasadena morality allan objective gap atheists california
moral institute punisher

Topic 8: stratus wpi sw jewish cdt tavares baseball players rocket polytechnic packet come humor special institute company jhu
weren go names

Topic 9: msg gatech prism utexas georgia ccwf cc food sensitivity vt superstition chinese institute technology austin atlanta
purdue andre go internet

Topic 10: fraser sfu portal simon craig cup canada dennis tm army mil wave handling sg silver problems tree indicated baby rand
y

Topic 11: window mit motif lcs x11r5 xterm widget server font expo xpert application umn xlib usc internet enterpoop windows pr
ogram manager

Topic 12: indiana duke lehigh captain ns1 ucs adobe traded sean danny dale ticket silver golden roy vacation andrew isu space c
ar

Topic 13: henry toronto uga ai zoo buffalo spencer georgia max covington mcovingt athens michael programs vnews news-software a
rtificial intelligence amateur vax

Topic 14: windows card drive thanks dos mac pc scsi video file sale please files system monitor help disk anyone use driver

Topic 15: columbia cunibx gerald cc alchemy utoronto dare chem gary iastate denver nyx laurentian golchowy olchowy maynard iowa
ramsey ia objective

Topic 16: ca new cs use anyone nasa gov car year am ve very now his well two space much uk key

Topic 17: alaska ti dseg aurora nsmca gene space moon bull billion adams mary 214 cubs high residents race michael station shut
tle

Topic 18: radar detector su 4th quote utexas bbs sorry phone means serdar having computer never system being say cs russia upen
n

LDA Review

- What did we learn?
 - per document topic distributions
 - per topic term distributions
- What can we use this for?
 - Dimensionality Reduction/Feature Extraction!
 - investigate topics (much like PCA components)

Other NLP Features

- Part of Speech tags
- Dependency Parsing
- Entity Detection
- Word Vectors
- See spaCy!

Using spaCy for NLP

Using spaCy for NLP

```
In [89]: 1 import spacy
          2
          3 # uncomment the line below the first time you run this cell
          4 #%run -m spacy download en_core_web_sm
          5 try:
          6
          7     nlp = spacy.load("en_core_web_sm")
          8
          9 except OSError as e:
          10     print('Need to run the following line in a new cell:')
          11     print('%run -m spacy download en_core_web_sm')
          12     print('or the following line from the commandline with eods-f20 activated:')
          13     print('python -m spacy download en_core_web_sm')
          14
          15 parsed = nlp("N.Y.C. isn't in New Jersey.")
          16 '|'.join([token.text for token in parsed])
```

2025-11-25 18:23:26.993980: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

```
Out[89]: "N.Y.C.|is|n't|in|New|Jersey|."
```

spaCy: Part of Speech Tagging

spaCy: Part of Speech Tagging

```
In [90]: 1 doc = nlp("Apple is looking at buying U.K. startup for $one billion.")
2
3 print(f"{'text':7s} {'lemma':7s} {'pos':5s} {'is_stop'}")
4 print('-'*30)
5 for token in doc:
6     print(f"{'token.text':7s} {'token.lemma_':7s} {'token.pos_':5s} {'token.is_stop'}")
```

text	lemma	pos	is_stop
Apple	Apple	PROPN	False
is	be	AUX	True
looking	look	VERB	False
at	at	ADP	True
buying	buy	VERB	False
U.K.	U.K.	PROPN	False
startup	startup	VERB	False
for	for	ADP	True
\$	\$	SYM	False
one	one	NUM	True
billion	billion	NUM	False
.	.	PUNCT	False

spaCy: Part of Speech Tagging

```
In [90]: 1 doc = nlp("Apple is looking at buying U.K. startup for $one billion.")
2
3 print(f"{'text':7s} {'lemma':7s} {'pos':5s} {'is_stop'}")
4 print('-'*30)
5 for token in doc:
6     print(f'{'token.text':7s} {'token.lemma_':7s} {'token.pos_':5s} {'token.is_stop'}')
```

text	lemma	pos	is_stop
Apple	Apple	PROPN	False
is	be	AUX	True
looking	look	VERB	False
at	at	ADP	True
buying	buy	VERB	False
U.K.	U.K.	PROPN	False
startup	startup	VERB	False
for	for	ADP	True
\$	\$	SYM	False
one	one	NUM	True
billion	billion	NUM	False
.	.	PUNCT	False

```
In [91]: 1 from spacy import displacy
2 displacy.render(doc, style="dep")
```

dep

po

nsubj

spaCy: Part of Speech Tagging

spaCy: Entity Detection

spaCy: Entity Detection

```
In [92]: 1 [(ent.text,ent.label_) for ent in doc.ents]
```

```
Out[92]: [('Apple', 'ORG'), ('U.K.', 'GPE'), ('$one billion', 'MONEY')]
```

spaCy: Entity Detection

```
In [92]: 1 [(ent.text,ent.label_) for ent in doc.ents]
```

```
Out[92]: [('Apple', 'ORG'), ('U.K.', 'GPE'), ('$one billion', 'MONEY')]
```

```
In [93]: 1 displacy.render(doc, style="ent")
```

Apple ORG is looking at buying U.K. GPE startup for \$one billion MONEY .

spaCy: Word Vectors

- word2vec
- shallow neural net
- predict a word given the surrounding context (SkipGram or CBOW)
- words used in similar context should have similar vectors

spaCy: Word Vectors

- word2vec
- shallow neural net
- predict a word given the surrounding context (SkipGram or CBOW)
- words used in similar context should have similar vectors

```
In [94]: 1 # Need either the _md or _lg models to get vector information
          2 # Note: this takes a while!
          3 # %run -m spacy download en_core_web_md
```

spaCy: Word Vectors

- word2vec
- shallow neural net
- predict a word given the surrounding context (SkipGram or CBOW)
- words used in similar context should have similar vectors

```
In [94]: 1 # Need either the _md or _lg models to get vector information
          2 # Note: this takes a while!
          3 # %run -m spacy download en_core_web_md
```

```
In [95]: 1 nlp = spacy.load('en_core_web_md') # _lg has a larger vocabulary
          2
          3 doc = nlp('Baseball is played on a diamond.')
          4 doc[0].text, doc[0].vector.shape, list(doc[0].vector[:3])
```

```
Out[95]: ('Baseball', (300,), [0.55838, 0.42791, -0.11687])
```

spaCy: Multiple Documents

spaCy: Multiple Documents

```
In [96]: 1 # Use nlp.pipe to transform multiple docs at once
2 docs = list(nlp.pipe(['Baseball is played on a diamond.',
3                       'Hockey is played on ice.',
4                       'Diamonds are clear as ice.']))
```

```
In [97]: 1 # using average of token vectors for each document.
2 np.array([[ '{:.2f}'.format(docs[i].similarity(docs[j])) for j in range(3)]
3           for i in range(3)])
```

```
Out[97]: array([[ '1.00', '0.85', '0.76'],
                ['0.85', '1.00', '0.77'],
                ['0.76', '0.77', '1.00']], dtype='<U4')
```

```
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:1113: UserWarning: Setting pen
alty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
  warnings.warn(
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:1113: UserWarning: Setting pen
alty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
  warnings.warn(
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
  warnings.warn(
/Users/andi/miniconda3/envs/sigma2/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_it
er was reached which means the coef_ did not converge
  warnings.warn(
```

Learning Sequences

- Hidden Markov Models
- Conditional Random Fields
- Recurrent Neural Networks
- LSTM
- GPT3
- BERT
- Transformers (MLP 16.4)

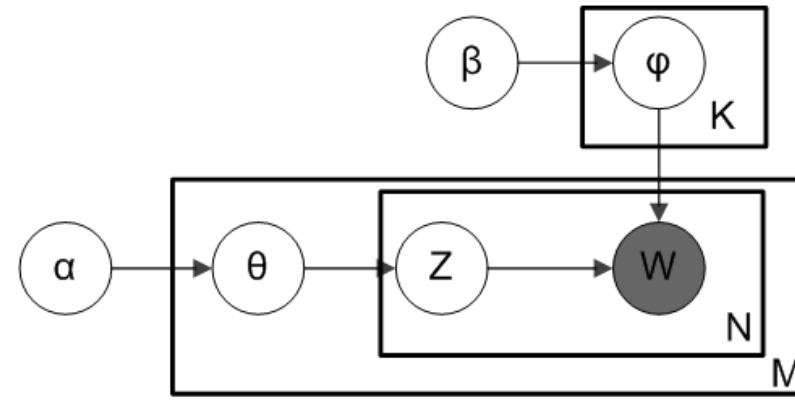
NLP Review

- corpus, tokens, vocabulary, terms, n-grams, stopwords
- tokenization
- term frequency (TF), document frequency (DF)
- TF vs TF-IDF
- sentiment analysis
- topic modeling
- POS
- Dependency Parsing
- Entity Extraction
- Word Vectors

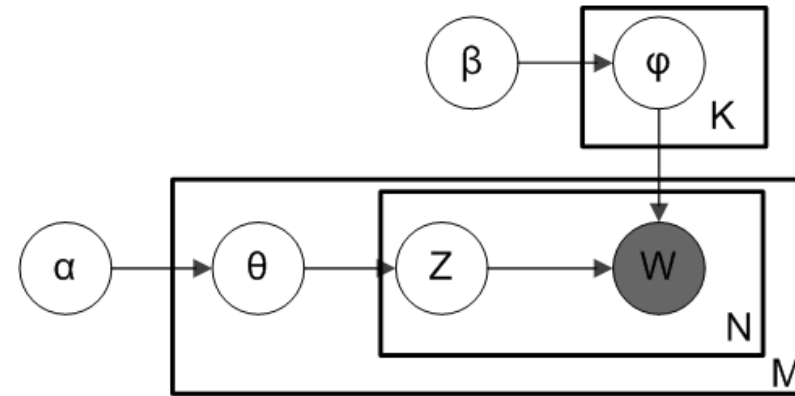
Questions?

Appendix: LDA Plate Diagram

Appendix: LDA Plate Diagram



Appendix: LDA Plate Diagram



K : number of topics

φ : per topic term distributions

β : parameters for word distribution die factory, length = V (size of vocab)

M : number of documents

N : number of words/tokens in each document

θ : per document topic distributions

α : parameters for topic die factory, length = K (number of topics)

z : topic indexes

w : observed tokens