Applied Machine Learning
Lecture 3: Pre-processing steps and hyperparameters tuning

Selpi (selpi@chalmers.se)

The slides are further development of Richard Johansson's slides

January 28, 2020

Overview

the first step: mapping features to numerical vectors

- scikit-learn's learning methods works with features as **numbers**, not strings
- they can't directly use the feature dicts we have stored in `X`
- converting from string to numbers is the purpose of these lines:

```
vec = DictVectorizer()
Xe = vec.fit_transform(X)
```

X

```
{ 'bloodtype':'B', ... }
{ 'bloodtype':'O', ... }
{ 'bloodtype':'AB', ... }
         ...
         ...
         ...
{ 'bloodtype':'O', ... }
```

Xe

```
1 0 0 0 ...
0 1 0 0 ...
0 0 1 0 ...
      ...
      ...
      ...
0 1 0 0 ...
```

types of vectorizers

- a `DictVectorizer` converts from attribute–value dicts:

$$
\begin{array}{c}
\texttt{X} \\
\left[\begin{array}{l}
\texttt{\{ 'bloodtype':'B', ... \}} \\
\texttt{\{ 'bloodtype':'O', ... \}} \\
\texttt{\{ 'bloodtype':'AB', ... \}} \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\texttt{\{ 'bloodtype':'O', ... \}}
\end{array}\right]
\end{array}
\longrightarrow
\begin{array}{c}
\texttt{Xe} \\
\left[\begin{array}{l}
\texttt{1 0 0 0 ...} \\
\texttt{0 1 0 0 ...} \\
\texttt{0 0 1 0 ...} \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\texttt{0 1 0 0 ...}
\end{array}\right]
\end{array}
$$

- a `CountVectorizer` converts from texts (after splitting into words) or lists:

$$
\begin{array}{c}
\texttt{X} \\
\left[\begin{array}{l}
\texttt{"this is a text"} \\
\texttt{"here is another text"} \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\texttt{"a cat on a mat"}
\end{array}\right]
\end{array}
\longrightarrow
\begin{array}{c}
\texttt{Xe} \\
\left[\begin{array}{l}
\texttt{1 1 1 1 0 0 ...} \\
\texttt{0 1 0 1 1 1 ...} \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\hfil\texttt{...}\hfil \\
\texttt{0 0 2 0 0 0 ...}
\end{array}\right]
\end{array}
$$

- a `TfidfVectorizer` is like a `CountVectorizer`, but also uses TF*IDF (downweighting common words)

what goes on in a DictVectorizer?

- each feature corresponds to one or more columns in the output matrix
- easy case: boolean and numerical features:

$$\underset{X}{\begin{bmatrix} \texttt{\{ 'f1':False, 'f2':7 \}} \\ \texttt{\{ 'f1':True, 'f2':2 \}} \\ \texttt{\{ 'f1':False, 'f2':9 \}} \end{bmatrix}} \longrightarrow \begin{array}{c} Xe \\ \begin{bmatrix} 0 & 7 \\ 1 & 2 \\ 0 & 9 \end{bmatrix} \\ \begin{array}{cc} \texttt{f1} & \texttt{f2} \end{array} \end{array}$$

what goes on in a DictVectorizer?

- each feature corresponds to one or more columns in the output matrix
- easy case: boolean and numerical features:

$$\begin{array}{c} \text{X} \\ \begin{bmatrix} \texttt{\{ 'f1':False, 'f2':7 \}} \\ \texttt{\{ 'f1':True,\ \ 'f2':2 \}} \\ \texttt{\{ 'f1':False, 'f2':9 \}} \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \text{Xe} \\ \begin{array}{c} \begin{bmatrix} 0 & 7 \\ 1 & 2 \\ 0 & 9 \end{bmatrix} \\ \begin{array}{cc} \uparrow & \uparrow \\ \texttt{f1} & \texttt{f2} \end{array} \end{array} \end{array}$$

- for string features, we reserve one column for each possible value: **one-hot encoding**
  - that is, we convert to booleans

$$\begin{array}{c} \text{X} \\ \begin{bmatrix} \texttt{\{ 'bt':'B', 'gen':'F' \}} \\ \texttt{\{ 'bt':'B', 'gen':'M' \}} \\ \texttt{\{ 'bt':'O', 'gen':'F' \}} \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \text{Xe} \\ \begin{array}{c} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \\ \begin{array}{cccc} \uparrow & \uparrow & \uparrow & \uparrow \\ \texttt{bt:B} & \texttt{bt:O} & \texttt{gen:F} & \texttt{gen:M} \end{array} \end{array} \end{array}$$

code example (DictVectorizer)

```
from sklearn.feature_extraction import DictVectorizer
X = [{'f1':'B', 'f2':'F', 'f3':False, 'f4':7},
     {'f1':'B', 'f2':'M', 'f3':True,  'f4':2},
     {'f1':'O', 'f2':'F', 'f3':False, 'f4':9}]
vec = DictVectorizer()
Xe = vec.fit_transform(X)
print(Xe.toarray())

print(vec.get_feature_names())
```

code example (DictVectorizer)

```
from sklearn.feature_extraction import DictVectorizer
X = [{'f1':'B', 'f2':'F', 'f3':False, 'f4':7},
     {'f1':'B', 'f2':'M', 'f3':True,  'f4':2},
     {'f1':'O', 'f2':'F', 'f3':False, 'f4':9}]
vec = DictVectorizer()
Xe = vec.fit_transform(X)
print(Xe.toarray())

print(vec.get_feature_names())
```

the result:

```
 [[ 1.  0.  1.  0.  0.  7.]
  [ 1.  0.  0.  1.  1.  2.]
  [ 0.  1.  1.  0.  0.  9.]]


 ['f1=B', 'f1=O', 'f2=F', 'f2=M', 'f3', 'f4']
```

Overview

Dealing with missing values

- Remove rows/columns that contain missing values (the easiest, but ...)
- Replace the missing values with some values; the process is called imputation (more strategic)

Imputation of missing values

- Derived data from the same feature (univariate feature imputation)
  - (e.g., mean, median, most-frequent)
- Derived data from several features

- See scikit-learn `Imputation of missing values`

Overview

Challenges with large number of features

- Imagine working with two data sets, one with 500 features, the other 10 features. Assume number of samples are moderate and the same in both data sets.
- What are the potential challenges with more features?

Challenges with large number of features

- Imagine working with two data sets, one with 500 features, the other 10 features. Assume number of samples are moderate and the same in both data sets.
- What are the potential challenges with more features?

- Finding key information/feature(s) becomes more difficult
- Higher complexity $\Rightarrow$ bad for generalization
- Training takes longer time

Challenges with large number of features

- Imagine working with two data sets, one with 500 features, the other 10 features. Assume number of samples are moderate and the same in both data sets.
- What are the potential challenges with more features?

- Finding key information/feature(s) becomes more difficult
- Higher complexity $\Rightarrow$ bad for generalization
- Training takes longer time

- To solve these issues: try **feature selection** method

Selecting Features: Brute Force

for every possible set of features S:

    train and evaluate the model based on S

return the set S_{max} that gave the best result

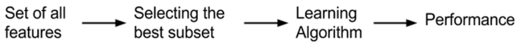
Feature Selection: Overall Ideas

- **filter** methods:

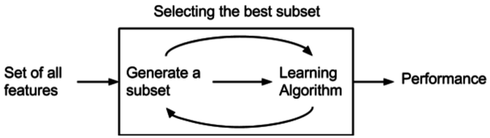
  Set of all features → Selecting the best subset → Learning Algorithm → Performance

Feature Selection: Overall Ideas

- **filter** methods:

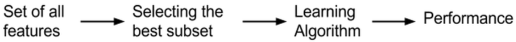


- **wrapper** methods:

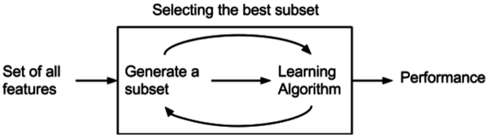


Feature Selection: Overall Ideas

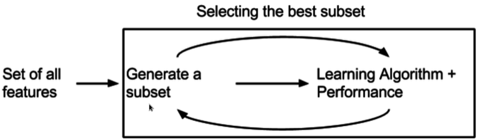
- **filter** methods:



- **wrapper** methods:



- **embedded** methods:



[Source: Wikipedia]

Association between a feature and the output (idea)

		output	
		0	1
feature	F	3	28
	T	25	2

		output	
		0	1
feature	F	14	11
	T	19	22

Feature Ranking Example (document categories)

UK

london	0.1925
uk	0.0755
british	0.0596
stg	0.0555
britain	0.0469
plc	0.0357
england	0.0238
pence	0.0212
pounds	0.0149
english	0.0126

China

china	0.0997
chinese	0.0523
beijing	0.0444
yuan	0.0344
shanghai	0.0292
hong	0.0198
kong	0.0195
xinhua	0.0155
province	0.0117
taiwan	0.0108

poultry

poultry	0.0013
meat	0.0008
chicken	0.0006
agriculture	0.0005
avian	0.0004
broiler	0.0003
veterinary	0.0003
birds	0.0003
inspection	0.0003
pathogenic	0.0003

coffee

coffee	0.0111
bags	0.0042
growers	0.0025
kg	0.0019
colombia	0.0018
brazil	0.0016
export	0.0014
exporters	0.0013
exports	0.0013
crop	0.0012

elections

election	0.0519
elections	0.0342
polls	0.0339
voters	0.0315
party	0.0303
vote	0.0299
poll	0.0225
candidate	0.0202
campaign	0.0202
democratic	0.0198

sports

soccer	0.0681
cup	0.0515
match	0.0441
matches	0.0408
played	0.0388
league	0.0386
beat	0.0301
game	0.0299
games	0.0284
team	0.0264

Figure 13.7: Features with high mutual information scores for six Reuters-RCV1 classes.

[Source: Manning & Schütze, *Introduction to Information Retrieval*]

Filter-based Feature Selection in scikit-learn

http://scikit-learn.org/stable/modules/feature_selection.html

- selectors:
  - `SelectKBest`
  - `SelectPercentile`
- feature scoring functions:
  - `f_classif`
  - `mutual_info_classif`
  - `chi2`

Example of a wrapper method: greedy forward selection

S = empty set
repeat:
  find the feature F that gives the largest
    improvement when added to S
  if there was an improvement add F to S
until there was no improvement

- the **MLXtend** library has an implementation
  - see `SequentialFeatureSelector`

Example of a wrapper method: backward selection

- See notebook
- Recursive Feature Elimination and Cross-Validation (RFECV)

Overview

Hyperparameters vs parameters

- Hyperparameters

- often used to help estimate the model parameters
- specified by the users
- tuned for the problem at hand
- In Random forests: number of trees, number of features considered
- In Neural Networks: learning rate, number of hidden layers

Hyperparameters vs parameters

- Hyperparameters
- often used to help estimate the model parameters
- specified by the users
- tuned for the problem at hand
- In Random forests: number of trees, number of features considered
- In Neural Networks: learning rate, number of hidden layers

- Parameters
- are needed by the model for making predictions
- the values are estimated from data
- usually are not set by the users
- In Random forests: random seed used
- In Neural Networks: weights

How to tune the hyperparameters?

- Trial and error

In scikit-learn:

`http://scikit-learn.org/stable/modules/grid_search.html`

- `GridSearchCV`
- `RandomizedSearchCV`

Overview

“Finding needle in haystack” problems

- Finding rare diseases
- Finding anomalies in travel pattern
- Differentiating crashes vs non-crashes situations
- . . .

[source]

Evaluation of imbalance cases

```
def has_disease(patient):
  return False
```

Evaluation of imbalance cases

```
def has_disease(patient):
  return False
```

- a `DummyClassifier` will have a high **accuracy**
- better to use **precision** and **recall** (more later)
  - or **sensitivity**/**specificity**, or **FPR**/**FNR**, ...

Dealing with class imbalance

- Change parameter(s) of the learning algorithm

Dealing with class imbalance

- Change parameter(s) of the learning algorithm
- Change the data

Dealing with class imbalance

- Change parameter(s) of the learning algorithm
- Change the data
- Use appropriate performance metric(s)

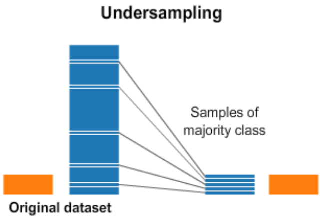
Change the parameter(s) of the learning algorithm

- Example in scikit-learn: `LinearSVC`

class_weight : *{dict, 'balanced'}, optional*

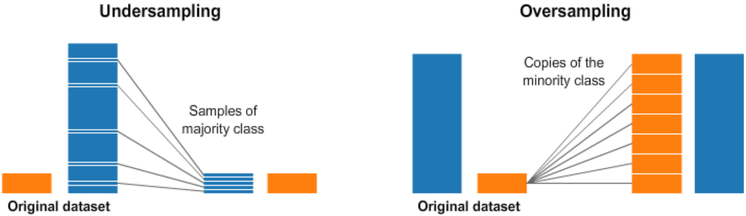
Set the parameter C of class i to `class_weight[i]*C` for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

Changing the data



[source]

Changing the data



[source]

- Or generate synthetic data using the existing data

Library for `imbalanced-learn`

- https://imbalanced-learn.readthedocs.io/
- See for example: BalancedRandomForestClassifier

Overview

Review of pre-processing and hyperparameters tuning

- Explain different ways of selecting features
- Explain pros/cons of GridSearchCV and RandomizedSearchCV
- Explain different ways of dealing with imbalance classes

Next lecture

- Linear classifiers and regression models
- (Methods for) data collection and bias
- Annotating data