Lecture-5-Model-Evaluation

May 14, 2019

1 Machine Learning

Classification Metrics and Model Evaluation

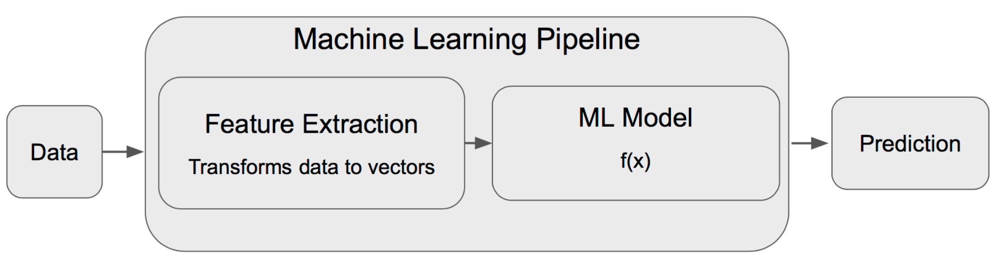
2 Machine Learning Pipelines

3 Model Evaluation

- Machine Learning models are evaluated by comparing the predictions $f(x) = \hat{y}$ and the target values y
- Useful Metrics
  - Regression (for continuous predictions - covered later)
  - Classification
- As ML models can memorize any training data set, **all metrics must always be computed using cross-validation**

3.1 Classification Metrics

- Accuracy
- Precision
- Recall
- F1



ml-pipeline-2.png

3.2 Accuracy

Defined as
$\frac{\text{number of correct assignments}}{\text{number of data points}}$
Problem: Imbalanced classes

In [53]:
```
from sklearn.metrics import accuracy_score

y_predicted = np.array([1, 0, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [54]:
```
print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
```

3.3 Precision

$$\text{precision} = \frac{|\{\text{relevant instances}\} \cap \{\text{predicted instances}\}|}{|\{\text{predicted instances}\}|}$$

In [55]:
```
from sklearn.metrics import precision_score

y_predicted = np.array([1, 0, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [56]:
```
print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
print(f"Precision: {precision_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
Precision: 0.0
```

In [57]:
```
y_predicted = np.array([1, 1, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [58]:
```
print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
print(f"Precision: {precision_score(y_true, y_predicted)}")
```

```
Accuracy: 0.8
Precision: 0.5
```

3.4 Recall

$$\text{recall} = \frac{|\{\text{relevant instances}\} \cap \{\text{predicted instances}\}|}{|\{\text{relevant instances}\}|}$$

In [59]:
```
from sklearn.metrics import recall_score

y_predicted = np.array([1, 0, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [61]:
```python
print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
print(f"Precision: {precision_score(y_true, y_predicted)}")
print(f"Recall: {recall_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
Precision: 0.0
Recall: 0.0
```

In [62]:
```python
y_predicted = np.array([1, 1, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [63]:
```python
print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
print(f"Precision: {precision_score(y_true, y_predicted)}")
print(f"Recall: {recall_score(y_true, y_predicted)}")
```

```
Accuracy: 0.8
Precision: 0.5
Recall: 1.0
```

3.5 Precision and Recall

Source Wikipedia page on Precision and Recall

3.6 F1 Score

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

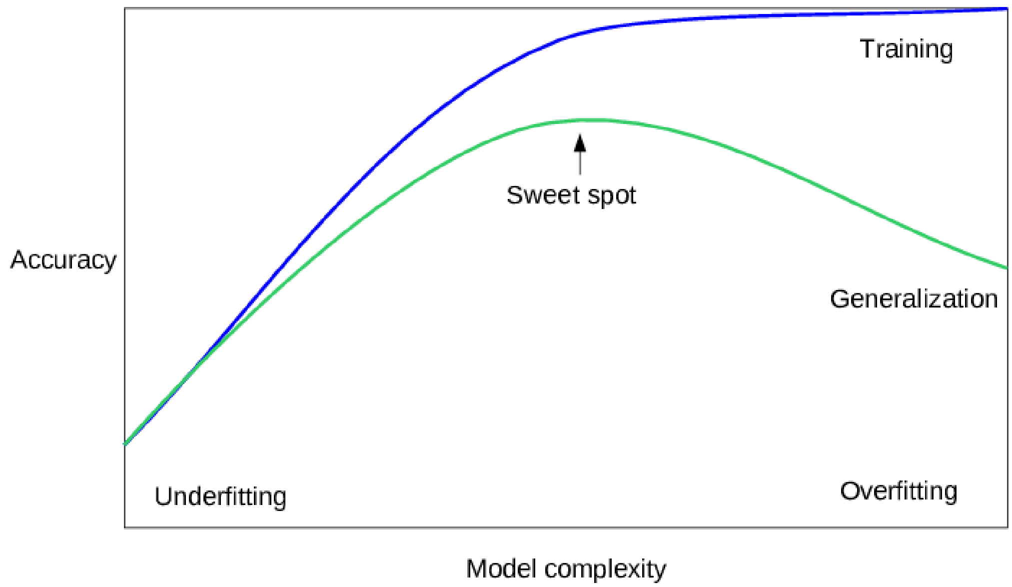
In [71]:
```python
from sklearn.metrics import f1_score

y_predicted = np.array([1, 0, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [72]:
```python
print(f"Precision: {precision_score(y_true, y_predicted)}")
print(f"Recall: {recall_score(y_true, y_predicted)}")
print(f"F1: {f1_score(y_true, y_predicted)}")
```

```
Precision: 0.0
Recall: 0.0
F1: 0.0
```

In [66]:
```python
y_predicted = np.array([1, 1, 0, 0, 0])
y_true = np.array([0, 1, 0, 0, 0])
```

In [70]:
```python
print(f"Precision: {precision_score(y_true, y_predicted)}")
print(f"Recall: {recall_score(y_true, y_predicted)}")
print(f"F1: {f1_score(y_true, y_predicted)}")
```



model_complexity

```
Precision: 0.5
Recall: 1.0
F1: 0.6666666666666666
```

4 Model Evaluation

4.1 Cross-Validation

- ML models can memorize any data set (**overfitting**)
- But we want our models to perform well on new data (**generalization of learned rules**)
- Cross-validation emulates the setting of new unseen data:
  - Split data in training and test
  - Train model on training set
  - Test model on test set

4.2 Simplest Cross-Validation for Model Evaluation

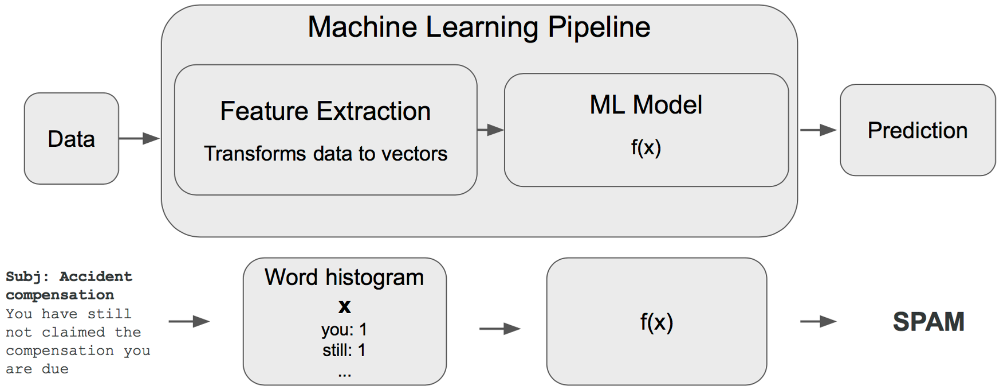
Split data into **two folds** and train / test **just once**

Fold 1	Test
Fold 2	Train

4.3 Example 3-Fold Cross-Validation for Model Evaluation:

Split data into three folds:

Fold 1
Fold 2



ml-pipeline-2.png

Fold 1	Train
Fold 2	Test
Fold 3	Train

4.3.3 Fold 3 is Test Data

Fold 1	Train
Fold 2	Train
Fold 3	Test

5 Recap: Cross-Validation for Model Evaluation

- Split dataset in training and test set
- Requires **at least two** different folds / partitions / data sets (train and test)
- Fast Cross-validation:
  - Make two partitions, train and test
- Cross-validation with best generalization performance estimates:
  - k-fold

6 Example Application: Classifying Parliament Speeches

- Task: Political bias prediction from text data
- Data: parliament speeches in German Bundestag

7 Machine Learning Pipeline

In [3]:
```
import os, gzip
import pandas as pd
import numpy as np
import urllib.request

import warnings
warnings.filterwarnings('ignore')

DATADIR = "data"

if not os.path.exists(DATADIR):
    os.mkdir(DATADIR)

file_name = os.path.join(DATADIR, 'bundestags_parlamentsprotokolle.csv.gzip')
if not os.path.exists(file_name):
    url_data = 'https://www.dropbox.com/s/1nlbfehnrwwa2zj/bundestags_parlamentsprotoko
    urllib.request.urlretrieve(url_data, file_name)

df = pd.read_csv(gzip.open(file_name), index_col=0).sample(frac=1)

alle_sprecher = df.sprecher.unique()
parteien = df.partei.unique()
partei_farben = {'cducsu':'black', 'linke':'purple', 'spd':'red', 'gruene':'green', 'fo
```

In [2]:
```
df[:5]
```

Out[2]:
```
       sitzung  wahlperiode                 sprecher  \
30289       76           18  Dr.ăKonstantin von Notz
11201      133           17            Peter Wichtel
24223       10           18              Dr.ăEva Högl
34597      127           18     Dr. Joachim Pfeiffer
7715        96           17            Bettina Kudla

                                                   text  partei
30289  Ich kann Ihnen schon sagen, wie das Ihre Unabh...  gruene
11201  Denn damit haben Sie am Ende genau das, was hi...  cducsu
24223  Unser Koalitionsvertrag ist voller guter Ideen...     spd
34597  Frau Präsidentin! Liebe Kolleginnen und Kolleg...  cducsu
7715   Zur Gläubigerbeteiligung - auch dieses Thema w...  cducsu
```

In [31]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import NearestCentroid
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
```

```
In [22]: # Put some data aside for model evaluation
         train_data, test_data, train_labels, test_labels = train_test_split(df['text'], df['pa

         ncc_clf = Pipeline([('vect', TfidfVectorizer(max_features=int(1e8))),
                             ('clf', NearestCentroid())]).fit(train_data, train_labels)

         logreg_clf = Pipeline([('vect', TfidfVectorizer(max_features=int(1e8))),
                                ('clf', SGDClassifier())]).fit(train_data, train_labels)
```

7.1 Evaluating NCC on Training Data

```
In [23]: ncc_predictions = ncc_clf.predict(train_data)
         print(classification_report(ncc_predictions, train_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.49      0.65      0.56      9612
         fdp       0.42      0.21      0.29      5275
      gruene       0.43      0.34      0.38      6323
       linke       0.62      0.37      0.46      8222
         spd       0.28      0.48      0.35      5511

   micro avg       0.44      0.44      0.44     34943
   macro avg       0.45      0.41      0.41     34943
weighted avg       0.47      0.44      0.43     34943
```

7.2 Evaluating NCC on Test Data

```
In [27]: ncc_predictions_test = ncc_clf.predict(test_data)
         print(classification_report(ncc_predictions_test, test_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.48      0.61      0.54      2498
         fdp       0.35      0.18      0.23      1365
      gruene       0.40      0.32      0.35      1569
       linke       0.54      0.34      0.42      1894
         spd       0.26      0.45      0.33      1410

   micro avg       0.41      0.41      0.41      8736
   macro avg       0.41      0.38      0.38      8736
weighted avg       0.42      0.41      0.40      8736
```

7.3 Evaluating Logistic Regression on Training Data

```
In [26]: logreg_predictions = logreg_clf.predict(train_data)
         print(classification_report(logreg_predictions, train_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.96      0.66      0.78     18852
         fdp       0.27      0.98      0.42       738
      gruene       0.69      0.93      0.79      3708
       linke       0.85      0.87      0.86      4768
         spd       0.62      0.85      0.72      6877

   micro avg       0.76      0.76      0.76     34943
   macro avg       0.68      0.86      0.71     34943
weighted avg       0.84      0.76      0.77     34943
```

7.4 Evaluating Logistic Regression on Training Data

```
In [28]: logreg_predictions_test = logreg_clf.predict(test_data)
         print(classification_report(logreg_predictions_test, test_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.91      0.55      0.69      5273
         fdp       0.04      0.62      0.08        50
      gruene       0.33      0.62      0.43       658
       linke       0.63      0.64      0.63      1177
         spd       0.37      0.57      0.45      1578

   micro avg       0.57      0.57      0.57      8736
   macro avg       0.46      0.60      0.46      8736
weighted avg       0.72      0.57      0.61      8736
```

7.5 Summary Comparison NCC and Logistic Regression

Nearest Centroid Classifiers - NCC is a simple model - Overall performance not great - But not a lot of overfitting

Logistic Regression - More powerful model - Better prediction performance - More overfitting