Big Data and Automated Content Analysis Part I+II

Week 10 – Wednesday
»Supervised Machine Learning II«

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

17 April 2019

Today

Alternatives to train/test split

Train/validation/test split

Train/validation/test split

- When you compare a lot of different models (or (hyper-)parameters), you might want to evaluate (compare) them using a third dataset
- e.g., make 80/20 split (train/test); then split first part again 80/20 (train/validation)
- only use the test data *at the very end* to get a final estimate of how good your model is.

Train/validation/test split

- When you compare a lot of different models (or (hyper-)parameters), you might want to evaluate (compare) them using a third dataset
- e.g., make 80/20 split (train/test); then split first part again 80/20 (train/validation)
- only use the test data *at the very end* to get a final estimate of how good your model is.

In short: Validation data to *select* the best approach; test data to get the accuracy of the approach you chose.

Alternatives to train/test split

Cross-validation

Cross-validation

```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()   # the classifier we trained last week
scores = cross_val_score(nb, train_features, [r[1] for r in reviews], cv=10)
print(scores)
```

results in:

```
[0.858  0.8612 0.8516 0.8528 0.8672 0.8664 0.8576 0.8652 0.8436 0.852 ]
```

In other words, we estimate the model 10 times on different trainig/validation data splits and get 10 different F1-scores (could be any other metric as well).

Cross-validation

Why would we want to do that?

- We could get some confidence interval around our scores
- Does not “waste” too much validation data
- ...and that’s important for hyperparameter tuning

See for more info

`https://scikit-learn.org/stable/modules/cross_validation.html`

Finding the optimal (hyper-)parameters
Grid-search

hyperparameter: a parameter of a model that is not learned through training, but specified in advance

Hyperparameter optimization with grid search

General idea

Rather than arbitrary trying some combinations of hyperparameters, let's systematically test and compare.

Hyperparameter optimization with grid search

General idea

Rather than arbitrary trying some combinations of hyperparameters, let's systematically test and compare.

Example

- To avoid overfitting, scikit-learn adds a *regularization term* to the loss function that is minimized to fit the regression.

Hyperparameter optimization with grid search

General idea

Rather than arbitrary trying some combinations of hyperparameters, let's systematically test and compare.

Example

- To avoid overfitting, scikit-learn adds a *regularization term* to the loss function that is minimized to fit the regression.
- Think of this term as a penalty for too complex models

Hyperparameter optimization with grid search

General idea

Rather than arbitrary trying some combinations of hyperparameters, let's systematically test and compare.

Example

- To avoid overfitting, scikit-learn adds a *regularization term* to the loss function that is minimized to fit the regression.
- Think of this term as a penalty for too complex models
- How much weight should our penalty carry? That's determined by a constant, C.

Hyperparameter optimization with grid search

General idea

Rather than arbitrary trying some combinations of hyperparameters, let's systematically test and compare.

Example

- To avoid overfitting, scikit-learn adds a *regularization term* to the loss function that is minimized to fit the regression.
- Think of this term as a penalty for too complex models
- How much weight should our penalty carry? That's determined by a constant, C.
- How to determine the best C? $\Rightarrow$ grid search

Hyperparameter optimization with grid search

Finding C in a logistic regression using 5-fold cross-validation

```
from sklearn.linear_model import LogisticRegressionCV
logregCV = LogisticRegressionCV(cv=5).fit(train_features, [r[1] for r in
      reviews])
```

Hyperparameter optimization with grid search

Finding C in a logistic regression using 5-fold cross-validation

```
from sklearn.linear_model import LogisticRegressionCV
logregCV = LogisticRegressionCV(cv=5).fit(train_features, [r[1] for r in
     reviews])
```

- Here, we just need to use `LogisticRegressionCV` instead of `LogisticRegression`.

Hyperparameter optimization with grid search

Finding C in a logistic regression using 5-fold cross-validation

```
from sklearn.linear_model import LogisticRegressionCV
logregCV = LogisticRegressionCV(cv=5).fit(train_features, [r[1] for r in
     reviews])
```

- Here, we just need to use `LogisticRegressionCV` instead of `LogisticRegression`.
- But we can use it to test any combination of choices (example at `https://scikit-learn.org/stable/auto_examples/model_selection/grid_search_text_feature_extraction.html`)

Grid-search takeaway

- When you want to systematically test what happens when you vary a hyperparameter, use grid-search to automatically do so and select the best value

Grid-search takeaway

- When you want to systematically test what happens when you vary a hyperparameter, use grid-search to automatically do so and select the best value
- sometimes already implemented (e.g., `LogisticRegressionCV` as direct replacement for `LogisticRegression`)

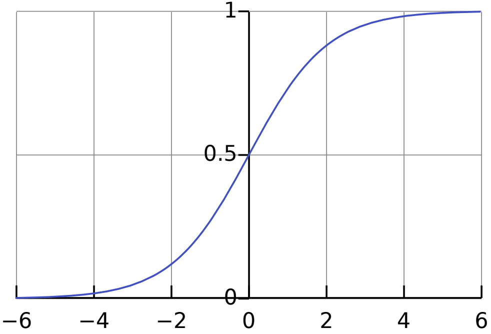
Grid-search takeaway

- When you want to systematically test what happens when you vary a hyperparameter, use grid-search to automatically do so and select the best value
- sometimes already implemented (e.g., `LogisticRegressionCV` as direct replacement for `LogisticRegression`)
- But `GridSearchCV` is very flexible: can be used in combination with pipeline (wait a minute...) for very different purposes

Finding the optimal (hyper-)parameters

Tuning decision thresholds with ROC curves

From estimate to label



In logistic regression, we use the *sigmoid function* to transform the estimates into probabilities.

To transform the probabilities into binary labels, we use a cutoff (default: 0.5).

Why use 0.5 as cutoff?

- It makes most sense (intuitively, mathematically)

Why use 0.5 as cutoff?

- It makes most sense (intuitively, mathematically)
- But remember our precision/recall tradeoff: maybe we want to be ‘stricter’ or ‘less strict’

Why use 0.5 as cutoff?

- It makes most sense (intuitively, mathematically)
- But remember our precision/recall tradeoff: maybe we want to be ‘stricter’ or ‘less strict’
- Maybe it is importance to us that our classifier is balanced and equally good in predicting both classes, even if overall accuracy suffers (slightly)

Why use 0.5 as cutoff?

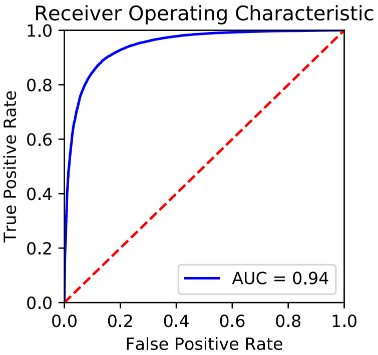
- It makes most sense (intuitively, mathematically)
- But remember our precision/recall tradeoff: maybe we want to be 'stricter' or 'less strict'
- Maybe it is importance to us that our classifier is balanced and equally good in predicting both classes, even if overall accuracy suffers (slightly)

Why use 0.5 as cutoff?

- It makes most sense (intuitively, mathematically)
- But remember our precision/recall tradeoff: maybe we want to be 'stricter' or 'less strict'
- Maybe it is importance to us that our classifier is balanced and equally good in predicting both classes, even if overall accuracy suffers (slightly)

Let's see what happens if we plot False Positives against True Positives (ROC-curve)

ROC Curve



- If we choose a threshold such that we get very little false positives, we also get too little true positives.
- Optimum in the upper left corner

So, how to we determine the exact value?

See notebook
`https://github.com/damian0604/bdaca/blob/master/rm-course-2/week10/Determining%20the%20cutoff-point%20in%20logistic%20regression.ipynb`

From feature set to final classification

Putting stuff together with pipelines

A pipeline

- Machine learning involves multiple steps (e.g., preprocessing $\rightarrow$ vectorizer $\rightarrow$ classification)
- We did all of them seperately
- Nothing wrong with that, but to ease use and evaluation *of the whole process*, we can define a `pipeline`.

Let's rewrite our example from last week as pipeline (and add cross-validation)

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
clf = LogisticRegressionCV()
pipe = make_pipeline(vec, clf)

pipe.fit([r[0] for r in reviews], [r[1] for r in reviews])
predictions = pipe.predict([r[0] for r in test])
```

Pipeline takeaway

- In principle, just a different way to write what we already did
- The more steps, the more relevant (e.g., reprocessing $\rightarrow$ vectorizer $\rightarrow$ dimensionality-reduction $\rightarrow$ classification)
- The more you rely on automated evaluation (e.g., grid search) of *multiple* steps in the pipeline, the more useful it is

From feature set to final classification

Visualizing feature weights with ELI5

Opening the black box

We said before that we are not so interested in the indivudual coefficients of, e.g., a logistic regression with 10,000 features.

Opening the black box

We said before that we are not so interested in the indivudual coefficients of, e.g., a logistic regression with 10,000 features. But sometimes we might:

- Spot errors (e.g., overfitting/features with tremendous weight that do not generalize beyond our data)

Opening the black box

We said before that we are not so interested in the indivudual coefficients of, e.g., a logistic regression with 10,000 features.
But sometimes we might:

- Spot errors (e.g., overfitting/features with tremendous weight that do not generalize beyond our data)
- Make (a bit more) explainable to (lay) audiences what's going on.

Opening the black box

We said before that we are not so interested in the indivudual coefficients of, e.g., a logistic regression with 10,000 features. But sometimes we might:

- Spot errors (e.g., overfitting/features with tremendous weight that do not generalize beyond our data)
- Make (a bit more) explainable to (lay) audiences what's going on.

Opening the black box

We said before that we are not so interested in the indivudual coefficients of, e.g., a logistic regression with 10,000 features.
But sometimes we might:

- Spot errors (e.g., overfitting/features with tremendous weight that do not generalize beyond our data)
- Make (a bit more) explainable to (lay) audiences what's going on.

We could just look at (sort) the coefficients from the classifier, but there's something better: `eli5` ("Explain Like I'm Five")

In [98]:
```
import eli5
eli5.show_weights(pipe, top=10)
```

Out[98]:

y=1 top features

Weight?	Feature
+9.043	great
+8.487	excellent
+6.908	perfect
… 37662 more positive …	
… 37178 more negative …	
-6.507	worse
-7.347	poor
-8.341	boring
-8.944	waste
-8.976	bad
-9.152	awful
-12.749	worst

In [111]:
```
eli5.show_prediction(clf, test[0][0],vec=vec)
```

Out[111]:

y=1 (probability **0.844**, score **1.689**) top features

Contribution?	Feature
+1.920	Highlighted in text (sum)
-0.232	<BIAS>

it is a rare and fine spectacle, an allegory of death and transfiguration that is neither preachy nor mawkish. a work of mature and courageous insight, northfork avoids arthouse distinction by refusing to belong to a kind. unlike the most memorable and accomplished film to impose an obvious comparison, wim wenders' 1987 wings of desire (der himmel über berlin), it sustains an ambivalence in a narrative spectrum spanning from the mundane to the supernatural. this story of earthly and celestial eminent domains in the american west withholds the fairytale literalness that marked its german predecessor in the ad hoc genre of angels shedding their wings with obsequious sentimentalism. its celestial transcendence, be it inspired by doleful faith or impelled by a fever dream, never parts ways with crud and rot. this firm grounding redounds to great credit for writers and directors mark and michael polish.

(example using the classifier `clf`, vectorizer `vec`, and pipeline `pipe` from privious slides)

From feature set to final classification

Last suggestions

Some further ideas to look into

Balancing classes

Your classifier probably works better if you have approximately the same amount of annotated training data for both classes (e.g., pos/neg). If getting such data is not an option, you may consider weighing accordingly, e.g. using `LogisticRegression(class_weight='balanced')`

Some further ideas to look into

More advanced pipelines

Consider constructing advanced pipelines, including a dimension reduction step:
`https://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html`

Some further ideas to look into

Combine different feature sets

E.g, use BOW-features as well as features such as sentence length, number of sentences (or whatever)

https://scikit-learn.org/stable/auto_examples/hetero_feature_union.html

Next meetings

Friday

No meeting (Easter break)

Next Wednesday: Unsupervised machine leraning 1

Principal Component Analysis, Clustering, and related techniques

Also (end of) next week: Take home exam!