Machine learning

Lecture 5: Model evaluation and feature selection methods.

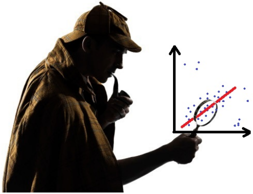
Aleksei Platonov
DingTalkID: aplatonov

13 April 2020
HDU

- Methods of classification and regression models evaluation.
- Model selection methods.
- Machine learning pipeline.
- Features processing: extraction, transformation, selection.

Model estimation: the motivation

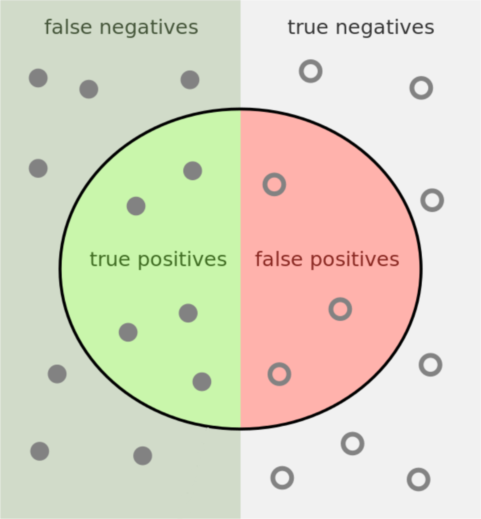
- An algorithm should be measurable. We need to understand the quality of model.
- The way of algorithm parameters tuning.
- It is the way of different machine learning algorithm comparison.
- The basic instruments of tracking changes in production data.



Basic terminology

Types of errors of binary classifier:

- TP - true positive
- TN - true negative
- FP - false positive
- FN - false negative



Point estimations

- Precision

$$Pr(a) = \frac{TP(a)}{TP(a) + FP(a)}$$

- Recall

$$Re(a) = \frac{TP(a)}{TP(a) + FN(a)}$$

- Accuracy

$$Acc(a) = \frac{TP(a) + TN(a)}{TP(a) + FP(a) + TN(a) + FN(a)}$$

- F-score

$$F_\beta(a) = (1 + \beta^2)\frac{Pr(a)Re(a)}{\beta^2 Pr(a) + Re(a)}, F = \frac{2Pr(a)Re(a)}{Pr(a) + Re(a)}$$

Model threshold

The most of algorithm allows estimating probability of class $+1$ in binary classification task:

- k-nearest neighbors using the ratio between number of objects of classes $+1$ and -1.
- Decision trees have special way to estimate probability and random forest can estimate probability using voting principle.
- Logistic regression can estimate the probability of $+1$.

We can make different classifiers changing the threshold values for probability of class $+1$.

Precision-Recall curves: the idea

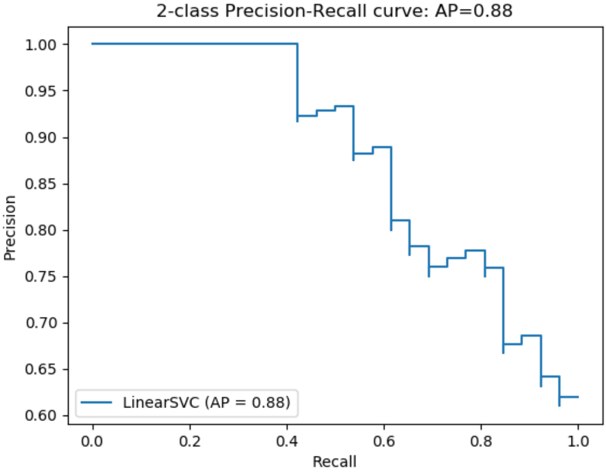
Algorithm 1 Precision-Recall curve computing

procedure $\textsc{PrRe}(a, X^k, \delta)$ $\triangleright$ a - model, X^k - test set
    $pr\_re\_points \leftarrow []$
    for $threshold \leftarrow 0 \dots 1.0$ with $step = \delta$ **do**
        $TP, FP, FN \leftarrow (0, 0, 0)$
        for $(x_i, y_i) \in X^k$ **do**
            $P(+1|x_i) \leftarrow a.proba(x_i)$
            $\overline{y}_i \leftarrow$ **if** $P(+1|x_i) > threshold$ **then** $+1$ **else** -1
            if $y_i = +1$ and $\overline{y}_i = +1$ **then**
                $TP \leftarrow TP + 1$
            else if $y_i = -1$ and $\overline{y}_i = +1$ **then**
                $FP \leftarrow FP + 1$
            else if $y_i = +1$ and $\overline{y}_i = -1$ **then**
                $FN \leftarrow FN + 1$
        $Pr \leftarrow \frac{TP}{TP+FP}$, $Re \leftarrow \frac{TP}{TP+FN}$
        pr_re_points.add($[Re, Pr]$)
    return pr_re_points

Precision-Recall curves: an example

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt

disp = plot_precision_recall_curve(
        classifier,
        X_test,
        y_test
)
disp.ax_.set_title('2-class Precision-Recall curve: '
'AP={0:0.2f}'.format(average_precision))
```

Precision-Recall curves: an example



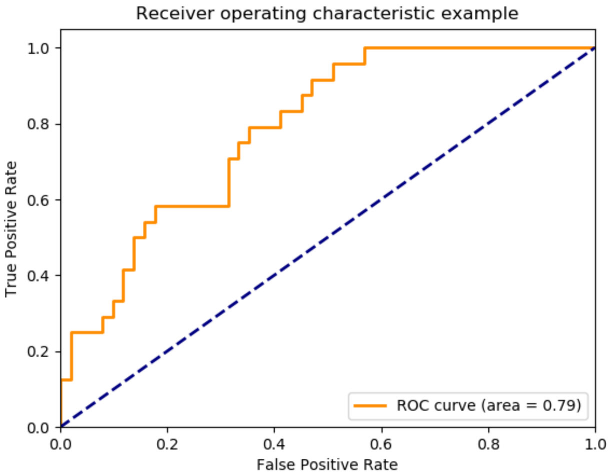
[1] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.htm

ROC-AUC: pseudocode

Algorithm 2 ROC-AUC computing

procedure $\mathrm{ROC\_AUC}(a, X^k, \delta)$ ▷ a - model, X^k - test set
    roc_points $\leftarrow$ []
    for *threshold* $\leftarrow 0 \dots 1.0$ with *step* $= \delta$ **do**
        $TP, FP, FN, TN \leftarrow (0, 0, 0, 0)$
        for $(x_i, y_i) \in X^k$ **do**
            $P(+1|x_i) \leftarrow a.proba(x_i)$
            $\overline{y}_i \leftarrow$ **if** $P(+1|x_i) > threshold$ **then** $+1$ **else** -1
            if $y_i = +1$ and $\overline{y}_i = +1$ **then**
                $TP \leftarrow TP + 1$
            else if $y_i = -1$ and $\overline{y}_i = +1$ **then**
                $FP \leftarrow FP + 1$
            else if $y_i = +1$ and $\overline{y}_i = -1$ **then**
                $FN \leftarrow FN + 1$
            else
                $TN \leftarrow TN + 1$
        $TPR \leftarrow \frac{TP}{TP+FN}$, $FPR \leftarrow \frac{FP}{FP+TN}$
        $roc_points.\mathrm{add}([FPR, TPR])$
    return *area_under_roc*(*roc_points*)

ROC-AUC: the idea



[2] https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Classifier estimations using Python

Scikit-Learn package: **sklearn.metrics**

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]

print(recall_score(y_true, y_pred, average='macro'))
# 0.33

print(precision_score(y_true, y_pred, average='macro'))
# 0.22

print(f1_score(y_true, y_pred, average='macro'))
# 0.26
```

Regression standard evaluation techniques

- Mean absolute error:

$$MAE(a) = \frac{1}{k}\sum_{i=1}^{k} |a(x_i) - y_i|$$

- Mean squared error (variance):

$$MSE(a) = \frac{1}{k}\sum_{i=1}^{k} (a(x_i) - y_i)^2$$

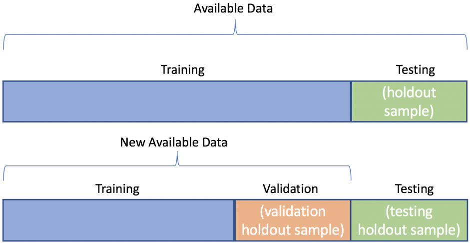
- R^2-score (coefficient of determination):

$$R^2(a) = 1 - \frac{\sum_{i=1}^{k} (y_i - a(x_i))^2}{\sum_{i=1}^{k} (y_i - E[y])^2}$$

Model selection: the motivation

- There are a lot of algorithms to machine learning task solving. How to choose an algorithm?
- One algorithm could have many parameters. How to choose values?

Hold-out: the idea



- Collect a big dataset.
- Split it into three parts: training set, validation set, and test set.
- Use the training set to model training.
- Use the validation set to investigate problems.
- Use test set to estimate model parameters.

Hold-out in Python

```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)

print(X)
# array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])
print(list(y))
# [0, 1, 2, 3, 4]

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.33, random_state=42
)

print(X_train)
# array([[4, 5], [0, 1], [6, 7]])
print(y_train)
# [2, 0, 3]

print(X_test)
# array([[2, 3], [8, 9]])
print(y_test)
# [1, 4]
```

LOO: the idea and pseudocode

Algorithm 3 LOO generation folds method

procedure $\mathrm{LOO}(X^l)$
  $\quad$ *folds* $\leftarrow$ []
  $\quad$ **for** $k \in 1 \ldots l$ **do**
  $\quad\quad$ $(x_i, y_i) \leftarrow X^l[i]$
  $\quad\quad$ $X^{l'} \leftarrow X^l \setminus \{(x_i, y_i)\}$
  $\quad\quad$ folds.append($X^{l'}$, (x_i, y_i))
  $\quad$ **return** folds

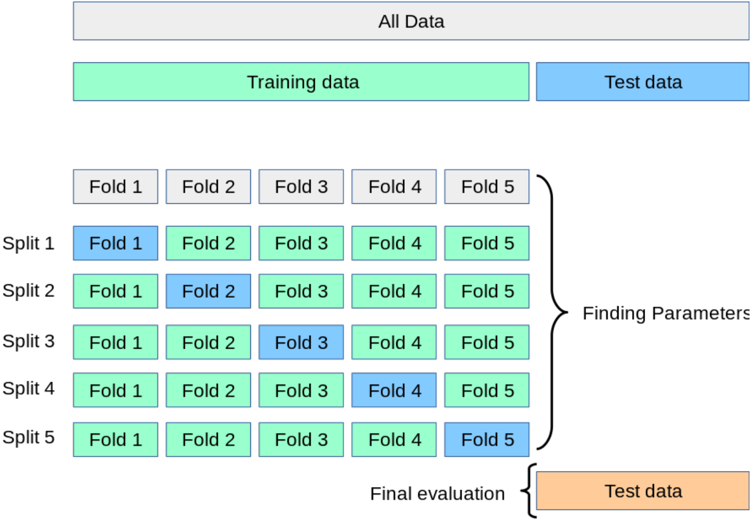
Algorithm 4 LOO usage for classifier evaluation

procedure $LOO\_Eval(X^l, \mu)$ $\quad\quad \triangleright$ μ - training algorithm
  $\quad$ $folds \leftarrow LOO(X^l)$
  $\quad$ $errors \leftarrow 0$
  $\quad$ **for** $fold \in folds$ **do**
  $\quad\quad$ $(X^{l'}, (x_i, y_i)) \leftarrow fold$
  $\quad\quad$ $a(x) \leftarrow \mu(X^{l'})$
  $\quad\quad$ **if** $a(x_i) \neq y_i$ **then**
  $\quad\quad\quad$ $errors \leftarrow errors + 1$
  $\quad$ **return** $errors/|folds|$

LOO in Python

```
import numpy as np
from sklearn.model_selection import LeaveOneOut
X = np.array([[1, 2], [3, 4], [5, 6]])
y = np.array([1, 2, 3])
loo = LeaveOneOut()

print(loo.get_n_splits(X))

for train_index, test_index in loo.split(X):
        X_train = X[train_index].tolist()
        X_test = X[test_index].tolist()
        y_train, y_test = y[train_index], y[test_index]
        print("TRAIN: X = {}, y = {}".format(X_train, y_train))
        print("TEST: X = {}, y = {}".format(X_test, y_test))

# TRAIN: X = [[3, 4], [5, 6]], y = [2 3]
# TEST: X = [[1, 2]], y = [1]
# TRAIN: X = [[1, 2], [5, 6]], y = [1 3]
# TEST: X = [[3, 4]], y = [2]
# TRAIN: X = [[1, 2], [3, 4]], y = [1 2]
# TEST: X = [[5, 6]], y = [3]
```

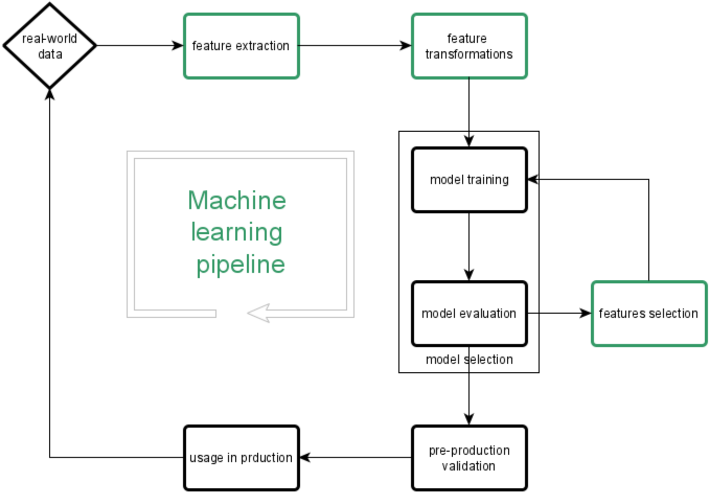
Cross-Validation: the idea



Cross-Validation in Python

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn import tree

data = load_breast_cancer()
X, y = data.data, data.target

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, X, y, cv=5) # accuracy
print(scores.tolist())
# [0.91, 0.92, 0.91, 0.934, 0.89]

t = (scores.mean(), scores.std() * 2)
print("Accuracy: %0.2f (+/- %0.2f)" % t)
# Accuracy: 0.92 (+/- 0.03)

from sklearn import metrics
scores = cross_val_score(clf, X, y, cv=5, scoring='f1_macro')
t = (scores.mean(), scores.std() * 2)
print("F1-score: %0.2f (+/- %0.2f)" % t)
# F1-score: 0.91 (+/- 0.03)
```

Machine learning pipeline



Features extraction: the motivation and data types

- Understanding your task: you see the data you need to process
- The way to create new features from **raw data**. It could increase the quality of ML algorithm
- Some data couldn't be processed without specia l data extraction functions:
  - Textual data
  - Images
  - Geodata
  - Dates and times

Textual data

- We can't send textual data into training algorithm as is: raw text doesn't contain useful information for ML at all
- The text processing pipeline is needed:
  1. Tokenization: "Don't forget about ML's lectures" $\rightarrow$ "Do", "n't", "forget", "about", "ML", "'s", "lectures"
  2. Normalization: "I've been there" $\rightarrow$ "to be [past perfect tense]"
  3. Statistical analysis of words co-occurrence: tf/idf, pmi

$$tf \times idf = tf[T] \cdot idf[t_i] = \frac{cnt(t_i)}{\sum_{t \in T} cnt(t)} \cdot log_2 \frac{|D|}{|d \in D, t_i \in|}$$

  4. Filtering stop words and lexicon creation:

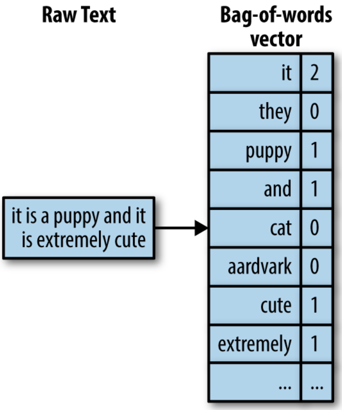
$$L(i) \rightarrow t_i, T(t_i) \rightarrow i$$

[1]Daniel Jurafsky, James H. Martin, Speech and Language Processing, 2nd Edition 2nd Edition

Textual data

After all this word we could generate feature vectors of text:

- Bag of words
- Neural networks
  - Word2Vec [Mikolov. Distributed Representations of Words and Phrases and their Compositionality]
  - Fasttext [Bojanowski. Enriching Word Vectors with Subword Information]
  - Glove [Pennington. GloVe: Global Vectors for Word Representation]



[2] www.coursera.org/learn/natural-language-processing-tensorflow

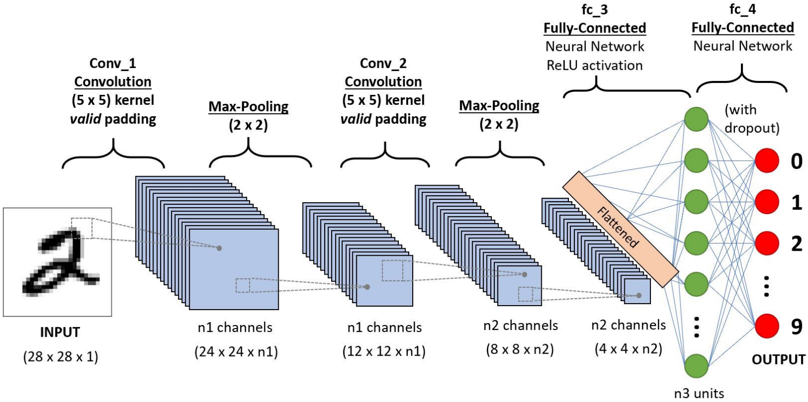
Images

- Images already have an interpretable numeric structure:
  - we could process them in ML directly.
  - images are too detailed, there are a lot of noise.
- There are a lot classic methods:[3]
  - Gamma correction
  - Filtering, noise reduction
  - Morphological processing and clusterization
- Trend of today: neural networks [again...]

[3] Rafael C. Gonzalez. Digital Image Processing

[4] Ian Goodfellow, Yoshua Bengio,& Aaron Courville (2016). Deep Learning. MIT Press.

Images

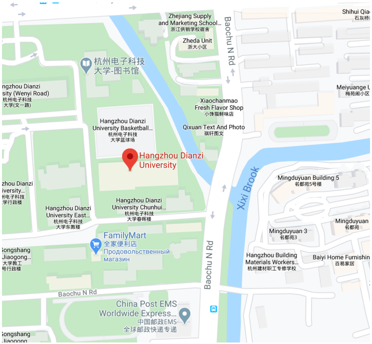


---

[5] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Geodata

- Latitude and longitude
  - + Has unambiguous interpretation
  - - Suffer from hardware precision
- String with address
  - + Independent from hardware
  - - Contains misprints, need to be geocoded
- Tons of useful information: nearest shops, banks, metro stations, traverse paths



(lat: 30.286844, lon: 120.140743)

Date and time

- Many different formats:
  - UNIX time (millis from 1970)
  - ISO 8601: MM-DD-YYYY HH:MI:SS (European format)
  - Russian format: DD-MM-YYYY HH:MI:SS
  - a.m., p.m. (13:00 == 1 p.m.)
- Complex comparison logic:

$$01 - 01 - 199213 : 00 : 00 > 01 - 01 - 199200 : 00 : 00,$$

  but

$$01 - 01 - 199213 : 00 : 00 < 01 - 02 - 199200 : 00 : 00,$$

- And don't forget about: different types of calendar, local holidays, weekends etc.

Features transformation: the motivation

- Deeper understanding the task:
  - Feature value distributions analysis
  - Understanding dependencies between feature values
- Simplify training: not all algorithms could accept any numeric ranges (e.g. k-NN).

Normalization

- z-scaling (scikit-learn: **StandardScaler**):
  - if an algorithm is sensitive to outliers in data (linear model)
  - if algorithm suffer from big difference between ranges of different feature values (kNN)

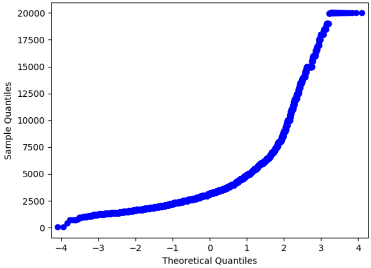
$$z_{f_i}(x) = \frac{f_i(x) - E[f_i]}{\sigma[f_i]},$$

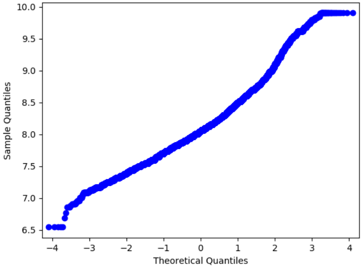
  where $E[f_i]$ - the mean value of f_i, $\sigma[f_i]$ - the standard deviation of f_i
- min-max scaling (scikit-learn: **MinMaxScaling**)
  - don't change the shape of feature values distribution
  - useful for representation on graphs

$$\overline{f_i}(x) = \frac{f_i(x) - min[f_i]}{max[f_i] - min[f_i]}$$

Normalization

Logarithmic transformation:





$$f_i^{'}(x) = log_{10}(f_i(x))$$

Binarization

- Split numeric interval into non-overlapping ranges and make binary vector (scikit-learn **KBinsDiscretizer** or **Binarizer** in case of ranges):

```
X = np.array([[ -3., 5., 15 ],
              [  0., 6., 14 ],
              [  6., 3., 11 ]])
est = preprocessing.KBinsDiscretizer(
        n_bins=[3, 2, 2],
        encode='ordinal'
).fit(X)
print(est.transform(X))
array([[ 0., 1., 1.],
       [ 1., 1., 1.],
       [ 2., 0., 0.]])
```

Binarization

- Encode nominal feature values as k-dimensional vector (k - the number of different values of feature):

```
enc = preprocessing.OneHotEncoder()
X = [['male', 'from␣US', 'uses␣Safari'],
     ['female', 'from␣Europe', 'uses␣Firefox']]
enc.fit(X)
print(enc.transform(
        [['female', 'from␣US', 'uses␣Safari'],
     ['male', 'from␣Europe', 'uses␣Safari']]
).toarray())
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

Features cooperation

In some cases it is useful to make new features combining existed features in **non-linear functions**:

- i.e. Let's we have a features: f_1 - the distance from point A to point B, f_2 - the time spend to the travel. We can obviously generate new feature: $f_3 = \frac{f_1}{f_2}$ - the average speed of travel and it could be useful.
- Sometimes we can just generate different combinations of features in an exhaustive manner: **PolynomialFeatures** in scikit-learn

Missing values processing

- Many tasks have missing data in their datasets. It could have many reasons: expensive measuring, mistakes, noised channel, etc.
- No all algorithms could work with missing feature values (i.e. kNN, linear models).
- But, we could transform missed values (scikit-learn **Imputer**):
  - Simple imputing: use a mean (median or the most frequent) value of non-missing values.
  - Sophisticated imputing: train a model to predict missing value (i.e. Decision Tree or Random Forest).

Features selection: the motivation

- Noisy features could spoil a great algorithm. Remember: garbage in - garbage out.
- Sometimes we need to evaluate features importance for task.
- Decrease computational complexity in production environment.
- Reduction of manual analysis of datasets.

Statistical approach

- Remove constant-like features because they aren't informative - the features with low variance value [scikit-learn **VarianceThreshold**]:

$$Var(f_j) = \frac{1}{l} \sum_{i=1}^{l} (f_j(x_i) - E[f_j])^2 \leq threshold$$

- Use informational criteria from statistics [scikit-learn **SelectKBest**]:
  - For regression: F-value, mutual information [scikit-learn **f_regression, mutual_info_regression**]
  - For classification: χ^2, F-value, mutual information [scikit-learn **chi2, f_classif, mutual_info_classif**]

Regularization and model-based approach

- We learnt the regularization method in linear models based on minimization of $||w||^2$ - the length of weight-vector. There is also another method (L1-regularization):

$$Q_\tau(w, X^l) = Q(w, X^l) + \frac{\tau}{2} \sum_{j=1}^{n} |w_j| \to \min_w$$

  See parameter **penalty** in linear models of scikit-learn
- Linear model learn vector of weight. Each weight could represent importance of feature.
- Tree-based models could filter uninformative features - they won't be added to tree using pruning.
- We can train models with different subsets of features and compare them using model selection method.

Conclusions

Today we discussed:

- Methods of classification and regression estimation.
- Features and model selection methods.
- Methods of features analysis and transformation.
- Pipeline of data processing in ML.