Applied Machine Learning
Lecture 5-2: Logistic regression and SVM

Selpi (selpi@chalmers.se)

The slides are further development of Richard Johansson's slides

February 7, 2020

Overview

A bit on perceptron

logistic regression

training a logistic regression classifier

detour: multiclass linear classifiers

support vector classification

optimizing the LR and SVM objectives

The perceptron algorithm and the simpler version

Perceptron algorithm

$\boldsymbol{w} = (0, \ldots, 0)$
repeat N times
  for $(\boldsymbol{x}_i, y_i)$ in the training set
    score $= \boldsymbol{w} \cdot \boldsymbol{x}_i$
    if a pos. is misclassified
      $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}_i$
    else if a neg. is misclassified
      $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}_i$
return $\boldsymbol{w}$

The perceptron algorithm and the simpler version

Perceptron algorithm

$\boldsymbol{w} = (0, \dots, 0)$
repeat N times
  for $(\boldsymbol{x}_i, y_i)$ in the training set
    score $= \boldsymbol{w} \cdot \boldsymbol{x}_i$
    if a pos. is misclassified
      $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}_i$
    else if a neg. is misclassified
      $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}_i$
return $\boldsymbol{w}$

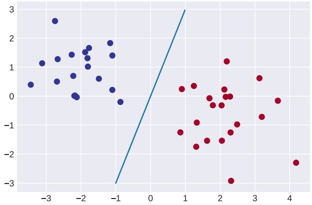
The simpler version
if the y_i are coded as +1 or -1:

$\boldsymbol{w} = (0, \dots, 0)$
for $(\boldsymbol{x}_i, y_i)$ in the training set
  score $= \boldsymbol{w} \cdot \boldsymbol{x}_i$
  if $y_i \cdot$ score ≤ 0
    $\boldsymbol{w} = \boldsymbol{w} + y_i \cdot \boldsymbol{x}_i$
return $\boldsymbol{w}$

how can we get the "certainty" of a linear classifier?

$$\text{score} = \boldsymbol{w} \cdot \boldsymbol{x}$$

- **large positive score**: quite certain that $\boldsymbol{x}$ belongs to the positive class
- **large negative score**: quite certain that $\boldsymbol{x}$ belongs to the negative class
- **near zero**: we are unsure



Overview

A bit on perceptron

logistic regression

training a logistic regression classifier

detour: multiclass linear classifiers

support vector classification

optimizing the LR and SVM objectives

The logistic regression model

- **logistic regression** is a method to train a linear classifier that gives a probabilistic output
- how to get the probability? use a **logistic** or **sigmoid** function:

$$P(\text{positive output}|\boldsymbol{x}) = \frac{1}{1 + e^{-\text{score}}}$$

where $e^{-\text{score}}$ = `np.exp(-score)`

The logistic regression model

- **logistic regression** is a method to train a linear classifier that gives a probabilistic output

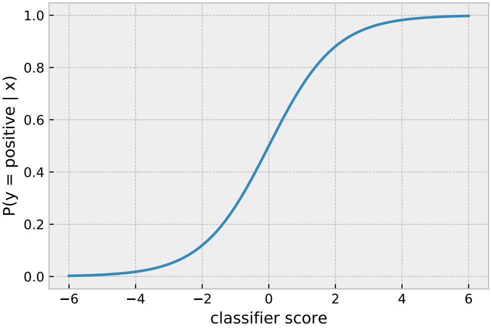
- how to get the probability? use a **logistic** or **sigmoid** function:

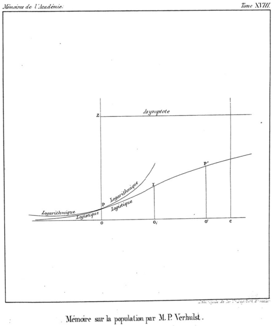
$$P(\text{positive output}|\boldsymbol{x}) = \frac{1}{1 + e^{-\text{score}}}$$

where $e^{-\text{score}} = \texttt{np.exp(-score)}$

$$P(\text{negative output}|\boldsymbol{x}) = 1 - \frac{1}{1 + e^{-\text{score}}} = \frac{1}{1 + e^{\text{score}}}$$

the logistic / sigmoid function [Verhulst, 1845]





interpreting the linear classifier's score

- the output score $\boldsymbol{w} \cdot \boldsymbol{x}$ can now be interpreted as the **log odds** in favor of the positive outcome
- **odds**: how much more likely is the positive outcome than the negative outcome?

$$\text{odds} = \frac{p}{1 - p}$$

> *Knew that we ventured on such dangerous seas*
> *That if we wrought out life 'twas* ***ten to one***
>
> Shakespeare, *Henry IV, Part II*, Act I, Scene 1 lines 183–4.

making it a bit more compact

- if we code the positive class as +1 and the negative class as -1, then we can write the probability a bit more neatly:

$$P(y|\boldsymbol{x}) = \frac{1}{1 + e^{-y \cdot \text{score}}}$$

in scikit-learn

- LR is called `sklearn.linear_model.LogisticRegression`
- `predict_proba` gives the probability output

code example: using a logistic regression classifier

Overview

recall: the maximum likelihood principle

- in a probabilistic model, we can train the model by selecing parameters that assign a **high probability to the data**
- in our case, the parameters are the weight vector $\boldsymbol{w}$
- adjust $\boldsymbol{w}$ so that each output label gets a high probability

the likelihood function

- formally, the "probability of the data" is defined by the **likelihood function**
- this is the product of the probabilities of all m individual training instances:

$$\mathcal{L}(\mathbf{w}) = P(y_1|\mathbf{x}_1) \cdot \ldots \cdot P(y_m|\mathbf{x}_m)$$

- in our case, this means

$$\mathcal{L}(\mathbf{w}) = \frac{1}{1 + e^{-y_1 \cdot (\mathbf{w} \cdot \mathbf{x}_1)}} \cdot \ldots \cdot \frac{1}{1 + e^{-y_m \cdot (\mathbf{w} \cdot \mathbf{x}_m)}}$$

rewriting a bit...

- we rewrite the previous formula

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{1+e^{-y_1\cdot(\boldsymbol{w}\cdot\boldsymbol{x}_1)}} \cdot \dots \cdot \frac{1}{1+e^{-y_m\cdot(\boldsymbol{w}\cdot\boldsymbol{x}_m)}}$$

as

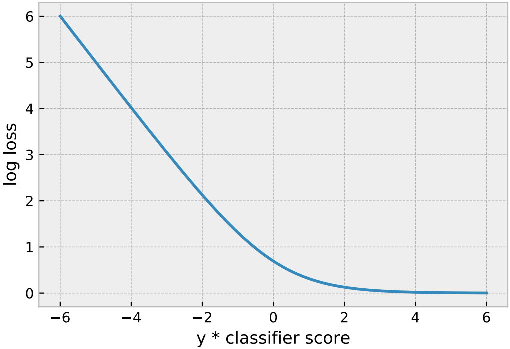
$$-\log \mathcal{L}(\boldsymbol{w}) = \mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}_1, y_1) + \ldots + \mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}_m, y_m)$$

where

$$\mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}, y) = \log(1 + \exp(-y \cdot (\boldsymbol{w} \cdot \boldsymbol{x})))$$

is called the **log loss** function

plot of the log loss



The fundamental tradeoff in machine learning

- **goodness of fit**: the learned classifier should be able to correctly classify the examples in the training data
- **regularization**: the classifier should be simple
- but so far in our LR description, we've just taken care of the first part!

$$-\log \mathcal{L}(\boldsymbol{w}) = \text{Loss}(\boldsymbol{w}, \boldsymbol{x}_1, y_1) + \ldots + \text{Loss}(\boldsymbol{w}, \boldsymbol{x}_m, y_m)$$

regularization in logistic regression models

- just like we saw for linear regression models (Ridge and Lasso), we can add a regularizer that **keeps the weights small**
- most commonly, the L_2 **regularizer**:

$$\|\boldsymbol{w}\|^2 = w_1 \cdot w_1 + \ldots + w_n \cdot w_n = \boldsymbol{w} \cdot \boldsymbol{w}$$

- …or an L_1 **regularizer**:

$$\|\boldsymbol{w}\|_1 = |w_1| + \ldots + |w_n|$$

which will do some feature selection

combining the pieces

- we combine the loss and the regularizer:

$$\frac{1}{N} \cdot \sum_{i=1}^{N} \text{Loss}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{\lambda}{2} \cdot \|\boldsymbol{w}\|^2$$

- in this formula, λ is a "tweaking" parameter that controls the tradeoff between loss and regularization
- note: in some formulations (including scikit-learn), there is a parameter C instead of the λ that is put before the loss

$$\frac{C}{N} \cdot \sum_{i=1}^{N} \text{Loss}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

check

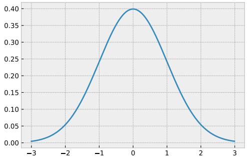
$$\frac{C}{N} \cdot \sum_{i=1}^{N} \mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

- how do we convert this into an algorithm?

probabilistic justification of the regularizers

- by adding the regularizer, we are carrying out **maximum a posteriori** estimation instead of maximum likelihood
  - in MAP estimation, we use a **prior** to push the parameters in a desired direction

$$\boldsymbol{w}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{w}} p(\mathrm{data}|\boldsymbol{w}) \cdot p(\boldsymbol{w})$$

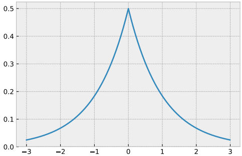


- L_2 regularizer = **Gaussian prior**

$$-\log p(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|^2 + \mathrm{constant}$$

- L_1 regularizer = **Laplace prior**

$$-\log p(\boldsymbol{w}) = \lambda |\boldsymbol{w}| + \mathrm{constant}$$



Overview

A bit on perceptron

logistic regression

training a logistic regression classifier

detour: multiclass linear classifiers

support vector classification

optimizing the LR and SVM objectives

two-class (binary) linear classifiers

- a **linear classifier** is a classifier that is defined in terms of a scoring function like this

$$\text{score} = \boldsymbol{w} \cdot \boldsymbol{x}$$

- this is a **binary** (2-class) classifier:
  - return the first class if the score > 0
  - ...otherwise the second class
- how can we deal with non-binary (**multi-class**) problems when using linear classifiers?

approaches to multiclass classification

- **idea 1**: break down the complex problem into simpler problems, train a classifier for each separately
- **idea 2**: modify the learning algorithm so that it can handle the multiclass case directly

idea 1: reduction from multiclass to binary

- **one-versus-rest** (“long jump”):
  - for each class c, make a binary classifier to distinguish c from all other classes
  - so if there are n classes, there are n classifiers
  - at test time, we select the class giving the highest score
- **one-versus-one** (“football league”):
  - for each pair of classes c_1 and c_2, make a classifier to distinguish c_1 from c_2
  - if there are n classes, there are $\frac{n \cdot (n-1)}{2}$ classifiers
  - at test time, we select the class that has most “wins”

example

- assume we're training a classifier of fruits and we have the classes `apple`, `orange`, `mango`
- in one-vs-rest, we train the following three classifiers:
  - `apple` vs `orange+mango`
  - `orange` vs `apple+mango`
  - `mango` vs `apple+orange`
- in one-vs-one, we train the following three:
  - `apple` vs `orange`
  - `apple` vs `mango`
  - `orange` vs `mango`

example (continued)

- we train classifiers to distinguish between `apple`, `orange`, and `mango`, using one-vs-rest
  - so we get $\boldsymbol{w}_{\texttt{apple}}$, $\boldsymbol{w}_{\texttt{orange}}$, $\boldsymbol{w}_{\texttt{mango}}$
- for some instance $\boldsymbol{x}$, the respective scores are

$$[-1, 2.2, 1.5]$$

so our guess is `orange`

in scikit-learn

- scikit-learn includes implementations of both of the methods we have discussed:
  - `OneVsRestClassifier`
  - `OneVsOneClassifier`
- however, the built-in algorithms (e.g. `Perceptron`, `LogisticRegression`) will do this automatically for you
  - they use one-versus-rest

idea 2: multiclass learning algorithms

- is it good to separate the multiclass task into smaller tasks that are trained independently?
  - maybe training should be similar to testing?
- let's make a model where one-vs-rest is used while training
  - we'll see how this can be done for logistic regression

binary LR: reminders

- the **logistic** or **sigmoid** function:

$$P(\text{positive output}|\boldsymbol{x}) = \frac{1}{1 + e^{-\text{score}}}$$

```
def sigmoid(score):
    return 1 / (1 + np.exp(-score))
```

- when training, we minimize the log-loss

$$\text{Loss}(\boldsymbol{w}, \boldsymbol{x}, y) = \log(1 + \exp(-y \cdot (\boldsymbol{w} \cdot \boldsymbol{x})))$$

multiclass LR using the softmax

- the **softmax** function is used in multiclass LR instead of the logistic:

$$P(y_i|\mathbf{x}) = \frac{e^{\text{score}_i}}{\sum_k e^{\text{score}_k}}$$

```
def softmax(scores):
    expscores = np.exp(scores)
    return expscores / sum(expscores)
```

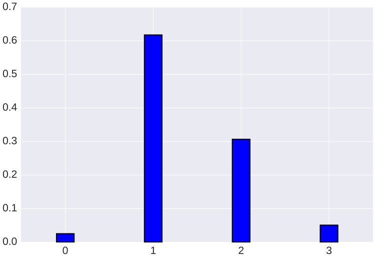
[exercise: make `softmax` numerically stable]

softmax example

```
def softmax(scores):
    expscores = np.exp(scores)
    return expscores / sum(expscores)

scores = [-1, 2.2, 1.5, -0.3]
print(softmax(scores))
```

```
array([ 0.02517067,  0.61750026,  0.30664156,  0.05068751])
```



cross-entropy loss

- when training, the softmax probabilities lead to the **cross-entropy** loss instead of the log loss

$$\text{Loss}_{CE}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = -\log P(y_i|\boldsymbol{x}_i) = -\log \frac{e^{\text{score}_i}}{\sum_k e^{\text{score}_k}}$$

- just like the log-loss:
  - high probability for the correct label $y_i \Rightarrow$ low loss
  - low probability for $y_i \Rightarrow$ high loss

multiclass LR in scikit-learn

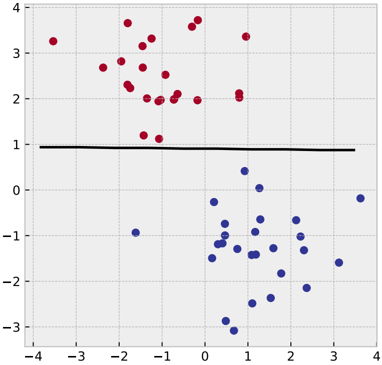
- `LogisticRegression(multi_class='multinomial')`
- (otherwise, separate classifiers are trained independently)

Overview

- A bit on perceptron
- logistic regression
- training a logistic regression classifier
- detour: multiclass linear classifiers
- support vector classification
- optimizing the LR and SVM objectives

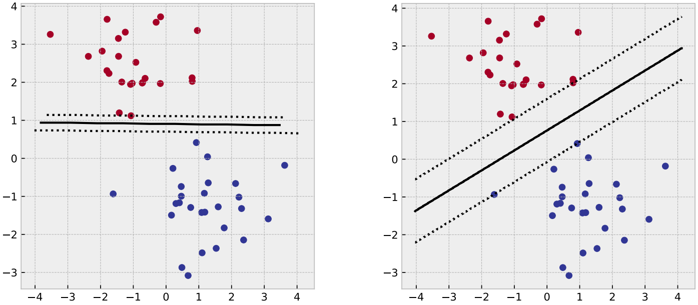
geometric view

- using lines to separate 2-dimensional data
- using planes to separate 3-dimensional data



margin of separation

- the **margin** γ denotes how well $\boldsymbol{w}$ separates the classes:



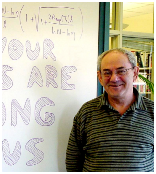
- γ is the shortest distance from the separator to the nearest training instance

large margins are good

- a result from statistical learning theory:

$$\text{true error} \leq \text{training error} + \text{BigUglyFormula}\left(\frac{1}{\gamma^2}\right)$$

- larger margin $\rightarrow$ better generalization



В. Н. ВАПНИК, А. Я. ЧЕРВОНЕНКИС

ТЕОРИЯ
РАСПОЗНАВАНИЯ
ОБРАЗОВ

СТАТИСТИЧЕСКИЕ ПРОБЛЕМЫ ОБУЧЕНИЯ

Structural risk minimization theorem

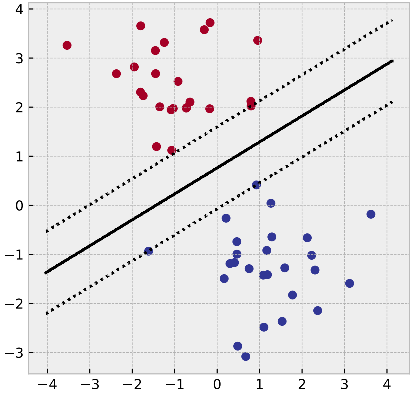
$$\text{error}_{true}(h) \leq \text{error}^{\gamma}_{train}(h) + C\sqrt{\frac{\frac{R^2}{\gamma^2}\ln m + \ln\frac{1}{\delta}}{m}}$$

$$\text{error}^{\gamma}_{train}(h) = \text{num. points with margin} < \gamma$$

[source]

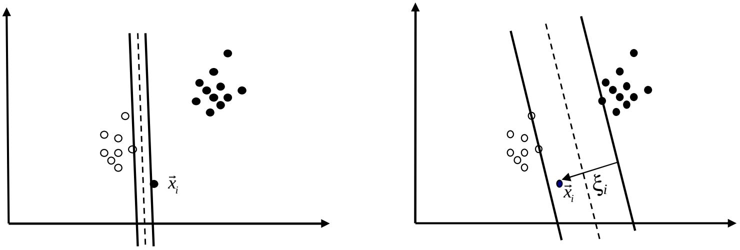
support vector machines

- **support vector machines** (SVM) or support vector classifiers (SVC) are linear classifiers constructed by selecting the $\boldsymbol{w}$ that maximizes the margin



soft-margin SVMs

- in some cases the dataset is inseparable, or nearly inseparable
- **soft-margin SVM**: allow some examples to be disregarded when maximizing the margin



A) Hard Margin SVM

B) Soft Margin SVM

stating the SVM as an objective function

- the hard-margin and soft-margin SVM can be stated mathematically in a number of ways
- we'll skip the details, but it can be shown (see Daumé's book) that the soft-margin SVM can be stated as minimizing

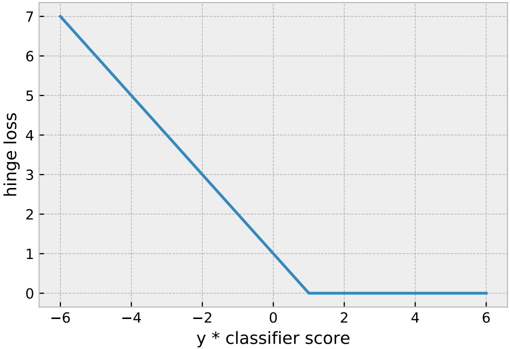
$$\frac{C}{N} \cdot \sum_{i=1}^{N} \mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

where

$$\mathrm{Loss}(\boldsymbol{w}, \boldsymbol{x}, y) = \max(0, 1 - y \cdot (\boldsymbol{w} \cdot \boldsymbol{x}))$$

is called the **hinge loss**

plot of the hinge loss



in scikit-learn

- linear SVM is called `sklearn.svm.LinearSVC`

Overview

A bit on perceptron

logistic regression

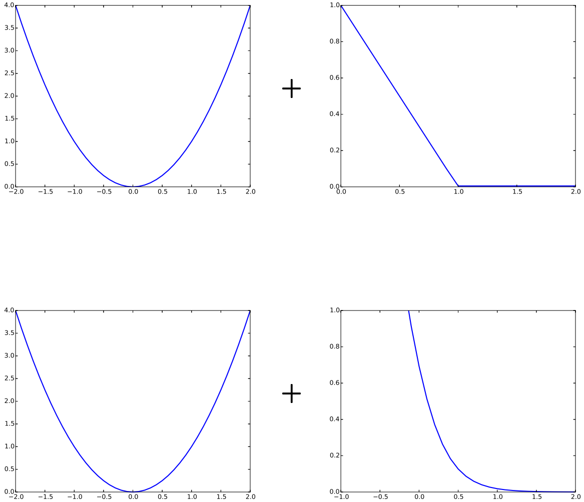
training a logistic regression classifier

detour: multiclass linear classifiers

support vector classification

optimizing the LR and SVM objectives

SVM and LR have convex objective functions



optimizing SVM and LR

- since the objective functions of SVM and LR are convex, we can find $\boldsymbol{w}$ by stochastic gradient descent
- pseudocode:
  - set $\boldsymbol{w}$ to some initial value, e.g. all zero
  - iterate a fixed number of times:
    - select a single training instance $\boldsymbol{x}$
    - select a "suitable" step length η
    - compute the gradient of the hinge loss or log loss
    - subtract step length · gradient from $\boldsymbol{w}$
- note the similarity to the perceptron!

missing pieces

- setting the **learning rate** η
  - in principle, one can try to select a "small enough" value of η
  - in practice, it's better to decrease η gradually
  - we can use the **Pegasos** algorithm to set η as follows:

$$\eta = \frac{C}{t} = \frac{1}{\lambda \cdot t}$$

    where
    - t is the current step $(1, 2, \ldots)$
    - C or λ is the loss/regularization tradeoff
- **gradients** for SVM and LR loss functions (hinge and log loss)

Next lecture

- Gradient Boosting
- Evaluation methods