Machine Learning for NLP
Lecture 6: Kernel-based classifiers

UNIVERSITY OF GOTHENBURG

Richard Johansson

October 5, 2015

overview

- **kernels** give us an interesting connection between **linear** and **example-based** classifiers
  - a **linear** classifier computes a score for each feature, and then sums the scores
  - an **example-based** classifier uses a similarity function to compare a new instance to the training examples
  - informally, kernels are **similarity functions**; formally, they are **dot products** in some transformed vector space
- we start from the linear classifiers and show that many of them have an alternative example-based form
- the selling point: get rid of feature engineering, use a similarity function instead

overview

the primal and dual forms

mapping feature vectors into higher-dimensional spaces

kernels in classifiers

the primal and dual forms of a linear classifier

- the **primal form** of a linear classifier is the one that we have seen so far, where the classifier is defined in terms of **features**:

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

- in the **dual form**, we instead state the scoring function in terms of the **training examples** $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$:

$$\text{score}(\boldsymbol{x}) = \sum_i \alpha_i \cdot (\boldsymbol{x}_i \cdot \boldsymbol{x})$$

where α_i is an importance weight for the training example $\boldsymbol{x}_i$

the primal and dual forms: reflection

primal

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

dual

$$\text{score}(\boldsymbol{x}) = \sum_i \alpha_i \cdot (\boldsymbol{x}_i \cdot \boldsymbol{x})$$

- the dual form can be seen as some sort of example-based classifier
- why do we say that the primal and the dual are related?
  - can we go from the dual to the primal?

the primal and dual forms: reflection

primal

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

dual

$$\text{score}(\boldsymbol{x}) = \sum_i \alpha_i \cdot (\boldsymbol{x}_i \cdot \boldsymbol{x})$$

- the dual form can be seen as some sort of example-based classifier
- why do we say that the primal and the dual are related?
  - can we go from the dual to the primal?

$$\boldsymbol{w} = \sum_i \alpha_i \cdot \boldsymbol{x}_i$$

  - it's not obvious that we could make the opposite conversion...

perceptron in the primal and dual forms

initialize to all zeros:
  primal: $\mathbf{w} = (0, \ldots, 0)$
  dual: ?
for $(\mathbf{x}_i, y_i)$ in the training set $(\mathbf{X}, \mathbf{Y})$
  if y_i is positive and score$(\mathbf{x}_i) <= 0$
    add $\mathbf{x}_i$ to the classifier
      primal: $\mathbf{w} = \mathbf{w} + \mathbf{x}_i$
      dual: ?
  else if y_i is negative and score$(\mathbf{x}_i) >= 0$
    subtract $\mathbf{x}_i$ from the classifier
      primal: $\mathbf{w} = \mathbf{w} - \mathbf{x}_i$
      dual: ?
return the classifier

perceptron in the primal and dual forms

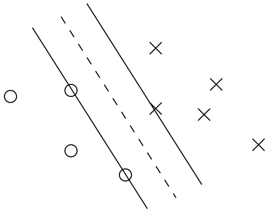
initialize to all zeros:
  primal: $\boldsymbol{w} = (0, \ldots, 0)$
  dual: $\alpha = (0, \ldots, 0)$
for $(\boldsymbol{x}_i, y_i)$ in the training set $(\boldsymbol{X}, \boldsymbol{Y})$
  if y_i is positive and score$(\boldsymbol{x}_i) <= 0$
    add $\boldsymbol{x}_i$ to the classifier
      primal: $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}_i$
      dual: ?
  else if y_i is negative and score$(\boldsymbol{x}_i) >= 0$
    subtract $\boldsymbol{x}_i$ from the classifier
      primal: $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}_i$
      dual: ?
return the classifier

perceptron in the primal and dual forms

initialize to all zeros:
    primal: $\mathbf{w} = (0, \ldots, 0)$
    dual: $\boldsymbol{\alpha} = (0, \ldots, 0)$
for $(\mathbf{x}_i, y_i)$ in the training set $(\mathbf{X}, \mathbf{Y})$
    if y_i is positive and score$(\mathbf{x}_i) <= 0$
        add $\mathbf{x}_i$ to the classifier
            primal: $\mathbf{w} = \mathbf{w} + \mathbf{x}_i$
            dual: $\alpha_i = \alpha_i + 1$
    else if y_i is negative and score$(\mathbf{x}_i) >= 0$
        subtract $\mathbf{x}_i$ from the classifier
            primal: $\mathbf{w} = \mathbf{w} - \mathbf{x}_i$
            dual: $\alpha_i = \alpha_i - 1$
return the classifier

what about the SVM?

- recall that the SVM can be defined in terms of a few **support vectors**



- this shows how the classifier is determined by the examples
- for the support examples, the α's are non-zero

the representer theorem

- the **representer theorem** shows that
  - **if** the learning method is stated as a minimization of

$$\text{objective}(\boldsymbol{w}) = \text{regularizer}(\boldsymbol{w}) + \text{loss}(\boldsymbol{w})$$

  - **then** the solution can be written in the dual form:

$$\boldsymbol{w} = \sum_i \alpha_i \cdot \boldsymbol{x}_i$$

- this class of learning methods includes SVM and LR
- Schölkopf, Herbrich, and Smola (2001): *A generalized representer theorem*

overview

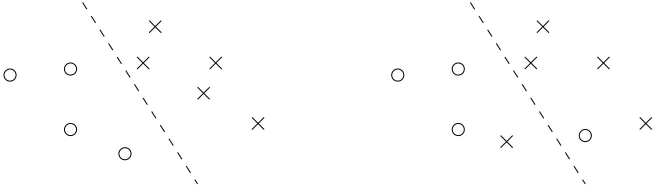
the primal and dual forms

mapping feature vectors into higher-dimensional spaces

kernels in classifiers

recap: linear separability

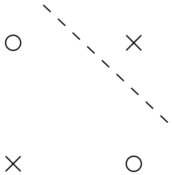
- some datasets can't be modeled with a linear classifier!



- a dataset is **linearly separable** if there exists a $\boldsymbol{w}$ that gives us perfect classification

a simple example of linear inseparability: an "XOR" situation

very good	Positive
very bad	Negative
not good	Negative
not bad	Positive

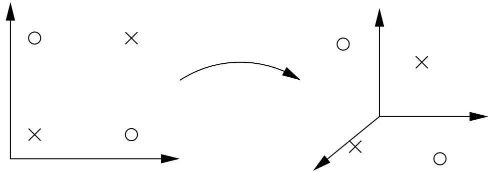


mapping into a larger vector space

- we may add "useful combinations" of features to make the dataset separable:

very good	very-good	Positive
very bad	very-bad	Negative
not good	not-good	Negative
not bad	not-bad	Positive

- from a geometrical viewpoint: we are creating a feature space with a higher dimensionality:



- lots of features $\rightarrow$ LOTS of combinations

mapping into a new vector space: formally

- we have some function ϕ that will take a feature vector $\boldsymbol{x}$ and convert it into a higher-dimensional vector $\phi(\boldsymbol{x})$
  - typically by forming combinations of the parts of $\boldsymbol{x}$
- then, instead of training a classifier on $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, we train it on $\phi(\boldsymbol{X}) = \phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_n)$
- it seems like a problem that ϕ would give a vector with a huge dimensionality, but we'll show later that we don't need to compute ϕ explicitly

example: XOR dataset

```
X = numpy.array([[1, 1],
                 [1, 0],
                 [0, 1],
                 [0, 0]])
Y = ['no', 'yes', 'yes', 'no']

clf = LinearSVC()
clf.fit(X, Y)

# linear inseparability, so we get less than 100% accuracy
print(accuracy_score(Y, clf.predict(X)))
```

example: XOR dataset converted into 3 dimensions

- let's apply the function

$$\phi([x1, x2]) = [x1^2, x2^2, \sqrt{2} \cdot x1 \cdot x2]$$

```
X = numpy.array([[1, 1, sqrt(2)*1*1],
                 [1, 0, sqrt(2)*1*0],
                 [0, 1, sqrt(2)*0*1],
                 [0, 0, sqrt(2)*0*0]])
Y = ['no', 'yes', 'yes', 'no']

clf = LinearSVC()
clf.fit(X, Y)

# in the 3-dimensional space, we get 100% accuracy
print(accuracy_score(Y, clf.predict(X)))
```

overview

the primal and dual forms

mapping feature vectors into higher-dimensional spaces

kernels in classifiers

- let's combine the two ideas we've been discussing:
  - converting examples $\mathbf{x}$ into higher-dimensional vectors $\phi(\mathbf{x})$
  - using the dual form of the classifiers
- then we get:

$$\text{score}(\mathbf{x}) = \sum_i \alpha_i \cdot (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}))$$

- we mentioned previously that it seems like a problem that $\phi(\mathbf{x})$ is huge...

the "kernel trick"

- in the dual form, the feature vectors $\phi(\mathbf{x})_1, \ldots, \phi(\mathbf{x})_n$ are used in **dot products only**:

$$\text{score}(\mathbf{x}) = \sum_i \alpha_i \cdot (\phi(\mathbf{x})_i \cdot \phi(\mathbf{x}))$$

- a **kernel** K is a function that corresponds to a dot product in some transformed vector space

$$\text{score}(\mathbf{x}) = \sum_i \alpha_i \cdot K(\mathbf{x}_i, \mathbf{x})$$

- the "kernel trick": in some cases, we may compute the kernel **without actually computing ϕ**

example: quadratic kernel

- recall the previous example, where we had

$$\phi([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2} \cdot x_1 \cdot x_2]$$

- in this case, we can compute the high-dimensional dot product without actually making the high-dimensional vectors:

$$K(\boldsymbol{a}, \boldsymbol{b}) = \phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{b}) = (\boldsymbol{a} \cdot \boldsymbol{b})^2$$

- this is called a **quadratic kernel**

quadratic kernel: derivation

$$
\begin{aligned}
\phi([a_1, a_2]) \cdot \phi([b_1, b_2]) &= [a_1^2, a_2^2, \sqrt{2} a_1 a_2] \cdot [b_1^2, b_2^2, \sqrt{2} b_1 b_2] = \\
&= a_1^2 b_1^2 + a_2^2 b_2^2 + 2 a_1 a_2 b_1 b_2 = \\
&= (a_1 b_1 + a_2 b_2)^2 = \\
&= ([a_1, a_2] \cdot [b_1, b_2])^2
\end{aligned}
$$

some common kernels

- many of the commonly used kernels are just some nonlinear function applied to the normal dot product
- **polynomial kernel**: $K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a} \cdot \boldsymbol{b} + c_0)^d$
  - ... where d is called the **degree** and c_0 the **offset**
  - this category includes the linear and quadratic kernels
  - in general, a polynomial kernel with degree d will implicitly form all combinations of d features
- **RBF kernel**: $K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\gamma \cdot |\boldsymbol{a} - \boldsymbol{b}|^2)$

kernel SVM in scikit-learn

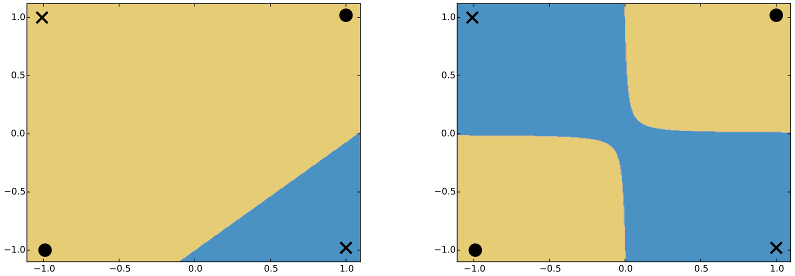
- the classifier `sklearn.svm.SVC` is an SVM implementation that uses kernels, as opposed to `LinearSVC` that we saw before
- examples:
  - quadratic: `SVC(kernel='poly', degree=2)`
  - RBF: `SVC(kernel='rbf')`
- you can also use your own kernel, instead of one of the built-in
  - `SVC(kernel=my_kernel_function)`

example: XOR dataset with quadratic-kernel SVM

```
X = numpy.array([[1, 1],
                 [1, 0],
                 [0, 1],
                 [0, 0]])
Y = ['no', 'yes', 'yes', 'no']

clf = SVC(kernel='poly', degree=2, C=1000)
clf.fit(X, Y)

# we get 100% accuracy because the quadratic kernel
# implicitly works in the 3-dimensional space

print(accuracy_score(Y, clf.predict(X)))
```

decision boundaries: linear and quadratic SVM

- a kernel-based classifier is linear in the high-dimensional space, but non-linear in the original space
- example: linear SVM compared to SVM with quadratic kernel



kernels as similarity functions

- by using a kernel, we got rid of the feature transformation ϕ
- we can **dispose of the feature extraction** step as well!
- in that case, the kernel function $K(\boldsymbol{a}, \boldsymbol{b})$ becomes a **similarity function** between two objects $\boldsymbol{a}$ and $\boldsymbol{b}$
- some interesting kernels useful in NLP:
  - **string kernels**: how many substrings do the two strings have in common?
  - **tree kernels**: how many subtrees do the two trees have in common?
    - many papers by Moschitti on this topic
  - **graph kernels**
  - **lexicon-based similarity functions**, for instance with WordNet

kernels in practice

- as we discussed, a kernel-based classifier can be seen as an example-based classifer
  - so they share the weakness of being slow at test time
  - as the training set grows, the classifier becomes slower...
- when should we use a kernel?
  - when defining a similarity function is easier than coming up with features
  - or when we think the features interact in some complicated way
- the alternative: extract features as normal, try to use your intuition and form feature combinations manually or by trial and error