Classification II + Python

COMP-599

Sept 21, 2016

Recap: Classification

- In NLP, determine some discrete property of a document:
  – Genre of the document (news text, novel, …?)
  – Overall topic of the document
  – Spam vs. non-spam
  – Identity, gender, native language, etc. of author
  – Positive vs. negative movie review

Recap: Steps

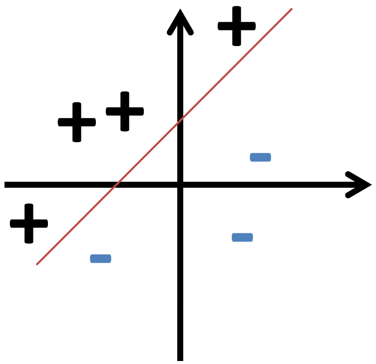
- Define problem and collect data set
- Extract features from documents
- Train a classifier on a training set
- Actually, train multiple classifiers using a training set; do model selection by tuning hyperparameters on a development set
- Use your final model to do classification on the test set

Recap: Classification Models

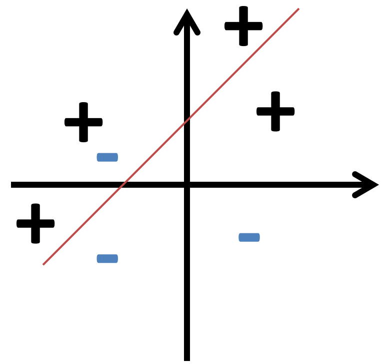
Some popular methods:

- Naïve Bayes
- Support vector machines
- Logistic regression
- Artificial neural networks

Linearly separable data



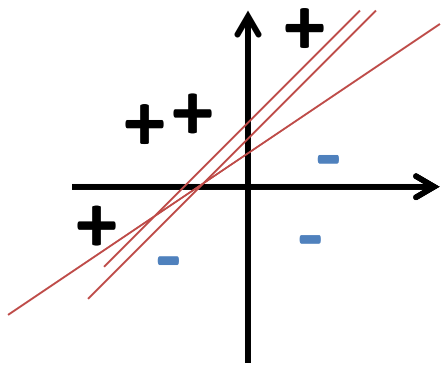
Linearly separable ☺



Not linearly separable ☹

Non-uniqueness of solutions

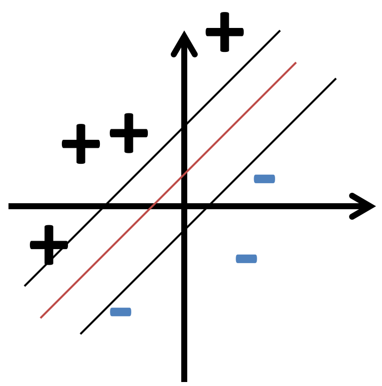
- Consider a set of linearly separable binary data points.
- How many separating hyperplanes can we find?



- An infinite number!!
- What is the best way to separate them?

Margin

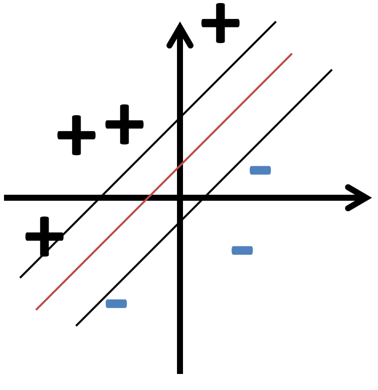
- For now, consider linearly separable data points.
- For a given separating hyper-plane, the margin is twice the distance between the hyperplane and the nearest training example.



Margin = width of the "strip" around the decision boundary

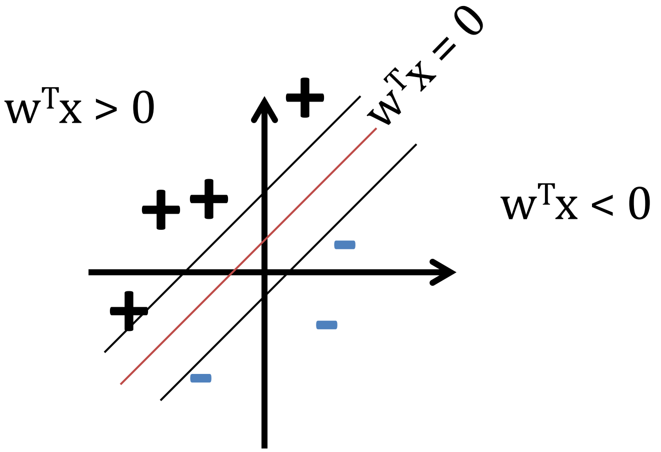
Margin

- SVM optimization problem in simple words: Maximize the margin



Decision Boundary

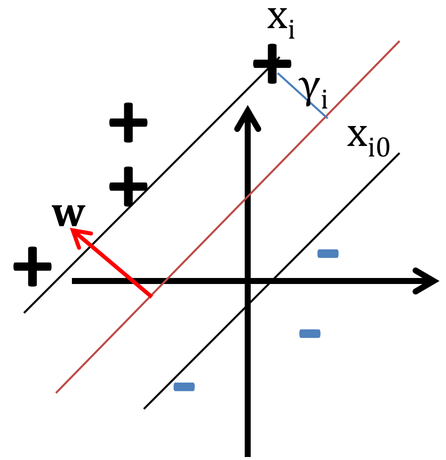
- Decision boundary: $w^T x=0$



Note: More generally, the decision boundary would be given by: $w^T x + b = 0$ where b is a bias term. For simplicity, we're assuming the bias term is 0.

Distance to the Decision Boundary

- Consider a set of data points $(\mathrm{x_i}, y_i)$ where the targets $y_i \in [-1; +1]$.
- Let $\mathrm{\gamma_i}$ be the distance from a point $\mathrm{x_i}$ to the boundary.
- $\mathbf{w}$ is the normal vector to the decision boundary.
- Point $\mathrm{x_{i0}}$ is the closest to $\mathrm{x_i}$ on the boundary.

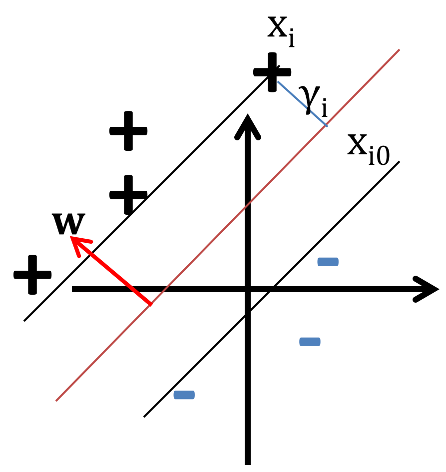


Distance to the Decision Boundary

- The vector from x_{i0} to x_i is: $\gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$

Note that:

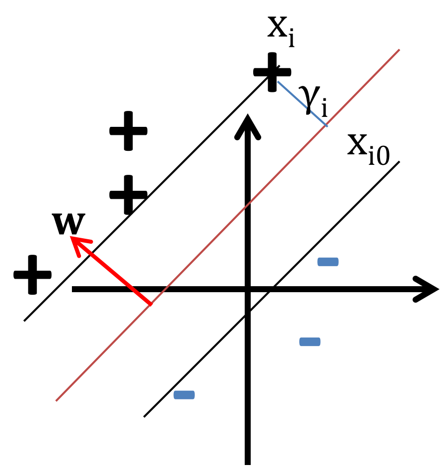
- γ_i is a scalar (distance between x_{i0} and x_i)
- $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit normal



Distance to the Decision Boundary

- Define: $\mathrm{x}_{\mathrm{i0}} = \mathrm{x}_{\mathrm{i}} - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- Remember that x_{i0} is on the decision boundary so $\mathbf{w}^{\mathrm{T}}\mathrm{x}_{\mathrm{i0}} = 0$

or: $\mathbf{w}^{\mathrm{T}}(\mathrm{x}_{\mathrm{i}} - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}) = 0$



Solving for γ_i, we get:

$$\gamma_i = \frac{\mathbf{w}^{\mathrm{T}}\mathrm{x}_{\mathrm{i}}}{||\mathbf{w}||} \quad \text{for the + class}$$

or $\gamma_i = y_i \frac{\mathbf{w}^{\mathrm{T}}\mathrm{x}_{\mathrm{i}}}{||\mathbf{w}||}$ for both classes

SVM Optimization Problem

- Remember the margin is 2M: twice the distance to the <u>closest</u> training example (from the decision boundary) so M = $\min_i \gamma_i$.
- But we want to maximize the margin so the SVM classifier essentially does: $\max_{\mathbf{w}} \min_i \gamma_i$.
- First formulation:

$$\max_{\mathbf{w}} M \text{ such that } y_i \frac{\mathbf{w}^{\mathrm{T}}\mathrm{x}_{\mathrm{i}}}{||\mathbf{w}||} \geq M.$$

SVM Optimization problem

- For some optimization purposes, add constraint: $||\mathbf{w}||M = 1$ and optimize ½ $||\mathbf{w}||^2$ instead of $||\mathbf{w}||$.
- <u>SVM optimization problem:</u>

Minimize ½ $||\mathbf{w}||^2$

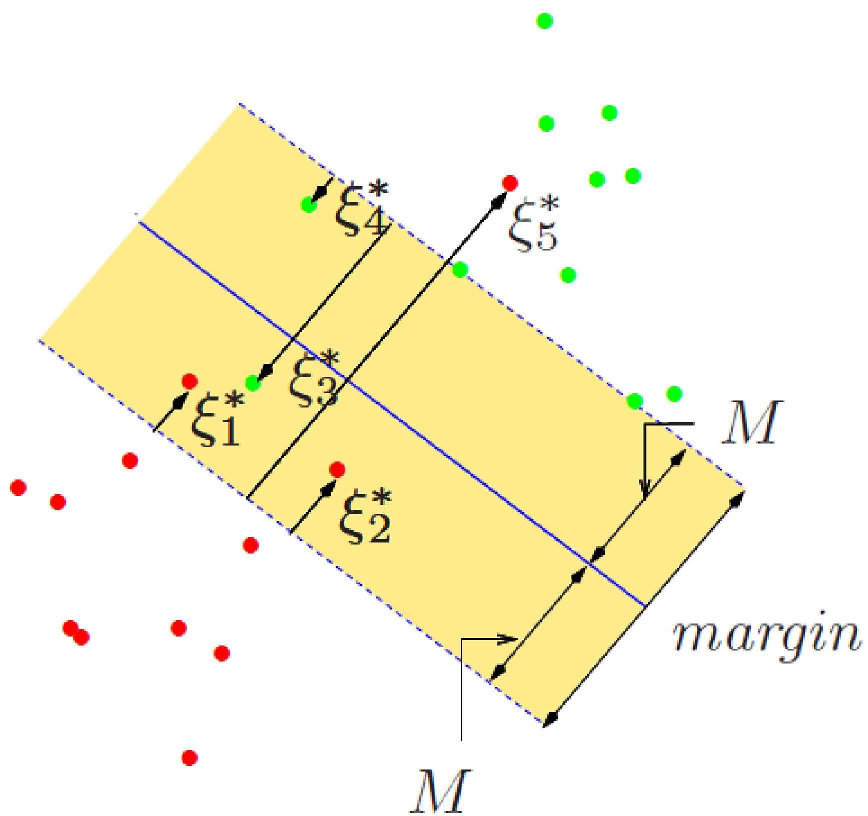
with respect to $\mathbf{w}$

subject to $y_i \mathbf{w}^{\mathrm{T}} \mathrm{x}_{\mathrm{i}} \geq 1$

(This is a standard quadratic programming problem. We know many ways that allow us to solve it.)

Soft SVM

- Accounts for non-linearly separable data points by introducing slack variables (ξ_i).



Source: Elements of Statistical Learning, Hastie et al. 2013

Soft SVM

- Optimization problem becomes:

$$\text{Minimize } C \sum_i \xi_i + ½\, ||\mathbf{w}||^2$$

$$\text{with respect to } \mathbf{w}, \xi$$

$$\text{subject to } y_i \mathbf{w}^{\mathrm{T}} \mathrm{x_i} \geq 1 - \xi_i$$

where the constant C is a parameter to be specified.

Intro to Python

- Widely used high-level, general-purpose programming language
- First version: 20 February 1991
  - Python 3 released in 2008
  - But we'll use Python 2.7
- Very important: It's all about <u>indentation</u>!
  - Wrong indentation will lead to an error

If statement

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Single'
... else:
...     print 'More'
```

https://docs.python.org/2/tutorials/

For loops

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
defenestrate 12
```

https://docs.python.org/2/tutorials/

Useful functions

- The range function
  Useful if you do need to iterate over a sequence of numbers. It generates lists containing arithmetic progressions:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
```

https://docs.python.org/2/tutorials/

Useful functions

- The split function

```
>>> x = 'blue,red,green'
>>> x.split(",")
['blue', 'red', 'green']
>>> a,b,c = x.split(",")
>>> a
'blue'
>>> b
'red'
>>> c
'green'
```

Useful functions

- The join function

```
>>> x = 'blue,red,green'
>>> x.split(",")
>>> a,b,c = x.split(",")
>>> s = "-"
>>> s.join([a,b,c])
'blue-red-green'
```

Classes and Objects

- Objects are an encapsulation of variables and functions into a single entity.
- Objects get their variables and functions from classes.
- Classes are essentially a template to create your objects.

Classes and Objects

- Simple example:

```
>>> class MyCourse:
...         name = "NLP"
...         def function(self): #called method
...              print "I love NLP."
```

- Assign MyClass to an object:

```
>>> myObject = MyCourse()
```

Now the variable "myObject" holds an object of the class "MyCourse" that contains the variable and the function defined within the class called "MyCourse".

Classes and Objects

- Accessing elements of an object

```
>>> myObject.name
'NLP'
```

```
>>> myOtherObject = MyCourse()
>>> myOtherObject.name = "comp599"
>>> print myOtherObject.name
comp599
```

Numpy

- Fundamental package for scientific computing with Python.
- Has a powerful N-dimensional array object

```
>>> import numpy as np
>>> x = np.array([[1,2,3],[4,5,6]],np.int32)
array([[1, 2, 3],
       [4, 5, 6]])
```

- Many useful functions

Numpy

- Array Slicing
  - Generate views of the data
  - Format: start : stop : step

```
>>> import numpy as np
>>> x = np.array([[1,2,3],[4,5,6]],np.int32)
array([[1, 2, 3],
       [4, 5, 6]])
>>> y = x[:,1]
>>> y
array([2, 5])
>>> z = np.array([0,1,2,3,4,5,6,7,8,9])
>>> z[1:7:2]
array([1, 3, 5])
```

Scikit-learn

- Machine Learning package in Python.
- Includes many classification, regression and clustering algorithms.
- Also, includes some datasets.

Example: Linear Regression

http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html

```
import numpy as np
from sklearn import datasets, linear_model
# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# The mean square error
print("Residual sum of squares: %.2f"
% np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))
```

NLTK

- Natural Language ToolKit
- Contains useful NLP tools such as stemmers, lemmatizers, parsers with a bunch of corpora

NLTK

- Tokenizers
  - Divide string into lists of substrings.
  - For example, tokenizers can be used to find the words and punctuation in a string:

```
>>> from nltk.tokenize import word_tokenize
>>> s = "Good muffins cost $3.88 in New York. Please buy
me two of them."
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New',
'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them',
'.']
```

http://www.nltk.org/_modules/nltk/tokenize.html

NLTK

- Stemmers
  - Remove morphological affixes from words, leaving only the word stem.
- Example: Porter stemmer

```
>>> from nltk.stem.porter import *
>>> stemmer = PorterStemmer()
>>> plurals = ['caresses', 'flies', 'dies', 'mules','denied',
'died', 'agreed', 'owned', 'humbled', 'sized', 'meeting', 'stating',
'siezing', 'itemization', 'sensational', 'traditional', 'reference',
'colonizer', 'plotted']
>>> singles = [stemmer.stem(plural) for plural in plurals]
>>> print(' '.join(singles))
caress fli die mule deni die agre own humbl size meet state siez
item sensat tradit refer colon plot
```

http://www.nltk.org/howto/stem.html

NLTK

- Lemmatizers
  - Determine the lemma of words
- Example: WordNet Lemmatizer

```
>>> from nltk.stem import WordNetLemmatizer
>>> wnl = WordNetLemmatizer()
>>> words = ['dogs', 'churches', 'aardwolves', 'abaci',
'hardrock']
>>> lemmata = [wnl.lemmatize(word) for word in words]
>>> for lemma in lemmata: print lemma
dog
church
aardwolf
abacus
hardrock
```

http://www.nltk.org/_modules/nltk/stem/wordnet.html

Final Notes

- Acknowledgment: The first part of the lecture is partially based on (a light version of) material from lectures 11 and 12 of Joelle Pineau's course COMP598 - Applied Machine Learning (Fall 2015).
- Check out the tutorial, "*An intro to Applied Machine Learning in Python*", by fellow RL-labber Pierre-Luc Bacon.