Machine Learning with Kernels in Python

J. Andrew Howe, PhD

Experienced Quantitative Modeler
Certified Energy Risk Professional

Mimar Sinan Guzel Sanatlar Universitesi
March 2016

- Three major topics intertwined in today's lecture
  1. Fundamental machine learning concepts
  2. Unsupervised classification with Kernels
  3. Implementation in Python
- These will be demonstrated on Fisher's famous Iris dataset

Kernel Trick

- *Kernel* refers to a *Reproducing Kernel Hilbert Space* (RKHS)
- First investigated by the mathematician Aronszajn in 1950.
- Applying a kernel to some data $X \in \mathbb{R}^{n \times p}$ correspond to nonlinearly mapping X into a higher dimensional *feature space* $\mathbb{F}$ and then taking the dot product in this space
- Consider the map

$$\begin{aligned} \phi :& \mathbb{R}^2 \to \mathbb{R}^3 \\ \phi\left([x_1, x_2]'\right) =& \left[x_2^2, \sqrt{2}x_1x_2, x_2^2\right]' \end{aligned}$$

- For two vectors x_i and x_j, we have

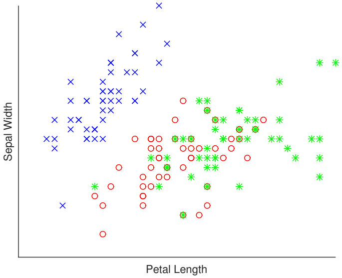
$$\begin{aligned} \phi\left(x_i\right)' \phi\left(x_j\right) &= \left[x_{i2}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2\right] \left[x_{j2}^2, \sqrt{2}x_{j1}x_{j2}, x_{j2}^2\right]' \\ &= \left[x_{i2}^2 x_{j2}^2, 2x_{i1}x_{i2}x_{j1}x_{j2}, x_{i2}^2 x_{j2}^2\right] \\ &= \left(x_i' x_j\right)^2 \end{aligned}$$

- This is called the *homogenous quadratic kernel*

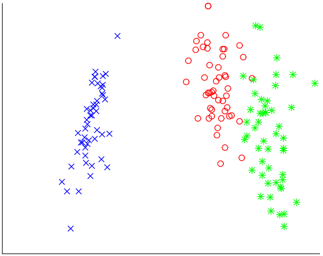
- Instead of mapping the data, then computing the dot product, we can compute this function $K(x_i, x_j) = (x_i' x_j)^2$
- This is called the *kernel trick*, and results in an $n \times n$-sized *kernel* or *Gram* matrix K.
- If our original data had $p = 4$ variables and $n = 150$ observations, K will be 150×150.
- Further statistical modeling is performed on K instead of X

Kernel Trick

- The kernel trick seems intuitively a bad idea
- As statisticians, we are frequently interested in *dimensional reduction* - *feature selection*, *feature extraction*, *feature engineering*
- However, the kernel trick inflates the dimensionality of our data
- Additionally, there are other issues that working with kernels cause:
  - K scales up rapidly with n (size and computation time)
  - K is often nonsingular and inversion is numerically unstable
  - The kernel trick is not reversible
  - Can't easily predict on new data
- The benefit we get from this is that clusters / groups in data can often be better separated in the higher-dimensional kernel space
- This is because the elements of the K matrix are all functions of distances between all pairs of observations



(a) Iris Data



(b) 1^{st} two Variables after Kernel Trick

Figure: Demonstrating Separation in Kernel Space; x is Setosa, o is Versicolor, and $*$ is Virginica

Some Kernel Functions

Function	Form
Linear Polynomial	$(\gamma x_i' x_j + c_0)^1$
Quadratic Polynomial	$(\gamma x_i' x_j + c_0)^2$
Cubic Polynomial	$(\gamma x_i' x_j + c_0)^3$
RBF (Gaussian)	$\exp\left(-\gamma \parallel x_i - x_j \parallel^2\right)$
Sigmoid	$\tanh\left(\gamma x_i' x_j + c_0\right)$
Laplace	$\exp\left(-\gamma \parallel x_i - x_j \parallel_1\right)$
Chi^2	$\exp\left(-\gamma \sum_i \frac{(x_i - x_j)^2}{x_i + x_j}\right)$

Classification with Kernels

- We will perform Kernel Support Vector Machine (SVM) classification on the Iris dataset
- Along with seeing the performance of Kernel SVM for classification, we will see several general machine learning techniques
- I have written my own code (mostly in MATLAB) to do everything we'll see here - the kernel trick, supervised classification, cross-validation, grid search, feature selection
- Knowing how to code for machine learning can be invaluable for all of us
- However, we will use the *scikit-learn* machine learning package in Python, which does much of it and makes automation of machine learning methods easy

- First, we use Logistic Regression to build a supervised classification model for the Iris Data, with the result:
  `Logistic Regression:  96.00%`
- Now, let's use the SVM with the linear polynomial, quadratic polynomial, RBF, and sigmoid kernels to classify the iris data
- Using the *svm.SVC* object which implements Kernel Support Vector Machine classification, we have:
  ```
  Linear Kernel:  99.33%
  Quadratic Kernel:  98.67%
  RBF Kernel:  98.67%
  Sigmoid Kernel:  4.00%
  ```
- But there is a possibility that these correct classification rates are all inflated because the model has been *overfit* to the observed data
- We can fix this with *cross-validation*

Cross-Validation

- Cross-validation (CV) refers to a general technique where the data is partitioned into different groups:

Training	Data used to fit the model
Testing	Data used to measure the performance of the model
Validation	Data used to validate learning for iterative-algorithms (optional, used less frequently)

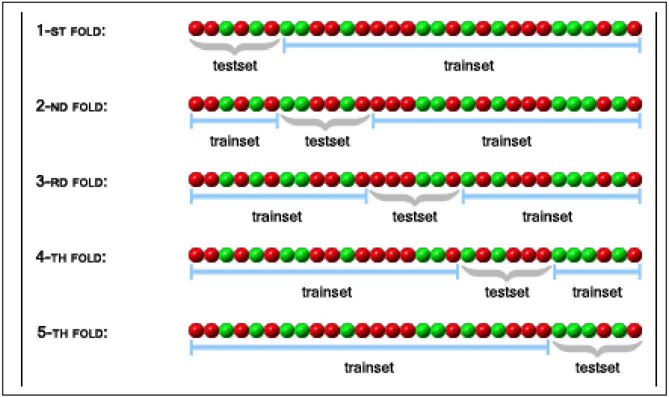
- There are three broad types of cross-validation
  - leave-p out
  - K-fold
  - randomized

Cross-Validation

- With leave-p out cross validation, we create $\binom{n}{p}$ partitions of the data
- For each, the training set has $n - p$ observations, with the remaining p as the testing set
- For large n and p, it becomes impractical to create all leave-p out CV sets
- If $n = 150$ and $p = 15$: 1.6239221627803363e+20 sets

Cross-Validation

- in *K-fold CV*, the data is divided into partitions such that every training set is different



An Example of 5-fold CV

- The data is often split into K-folds many times, with the first fold being randomly selected each time

- In randomized CV, a certain percentage of the data is randomly (uniformly) selected for inclusion in the training set
- The remaining observations are used as the testing set
- Since every observation has an equal chance of being selected for training, there's no guarantee that every partition will be different
- I generally use randomized cross-validation
- The scikit-learn package makes it easy to implement all these (and more) types of cross-validation

- Using the *cross_validation.ShuffleSplit* object and the *cross_validation.cross_val_score* function, the correct classification for each kernel is:

Correct Classification Summary

	Linear	Quadratic	RBF	Sigmoid
Min.	93.3%	90.0%	91.7%	0.0%
Mean	97.5%	95.1%	96.8%	26.5%
Median	98.3%	96.7%	96.7%	28.3%
Max.	100.0%	100.0%	100.0%	31.6%

- But what about those parameters we set for the kernel functions - could it be those are affecting the classification performance?
- We need to tune, or optimize, the parameters to identify a model with the best performance

- We defined four Kernel SVM objects with static parameters

Some Kernel Functions

Kernel	Parameters
Polynomial	$d = 1,\ c_0 = 0$
Polynomial	$d = 2,\ c_0 = 0$
RBF (Gaussian)	$\gamma = \frac{1}{p}$
Sigmoid	$\gamma = \frac{1}{p},\ c_0 = 0$

- Perhaps a non-homogenous cubic polynomial kernel ($d = 3,\ c_0 \neq 0$) would outperform the linear?
- What if we set $\gamma = 1$ for the RBF and Sigmoid?
- There are several different non-stochastic methods to optimize the parameters

- If we have a countable set of possible reasonable alternative values for each parameter, we could perform a grid search of all combinations of parameters
- Alternatively, if we know a range of possible values for each parameter, we can iteratively optimize each parameter with a real-valued univariate search against a profile score
- Yet a third option would be full multivariate optimization
- We'll see how to easily perform grid search - under cross-validation - to tune the parameters of Kernel SVM for the iris data

- We want to try all these models:

  Polynomial $d = [1, 2, 3]$, $\gamma = [\frac{1}{p}, 1, 2]$, $c_0 = [-1, 0, 1]$

  RBF $\gamma = [\frac{1}{p}, 1, 2]$

  Sigmoid $\gamma = [\frac{1}{p}, 1, 2]$, $c_0 = [-1, 0, 1]$
- This is a total of $27 + 3 + 9 = 39$ models, which we evaluate using the *fit* method of the *grid_search.GridSearchCV* object
- According to this, the model which gives the best correct classification performance uses the non-homogenous Linear Polynomial kernel with $c_0 = -1$
- Using the same 100 randomized CV partitions, this model obtains an average correct classification of 97.5%, with a median of 98.3%

- I mentioned that the kernel trick was not reversible
- After we create a classification model in the feature space $\mathbb{F}$, there's no way to go back and obtain a classification model in the original data space
- As well as complicating prediction of new observations, this makes it difficult for us to identify the most influential variables in our dataset, and to perform feature selection
- To perform feature selection with Kernel SVM classification, we need to sequentially select subsets of features and fit the Kernel model
- We can do this in Python, simultaneously with cross-validation and parameter tuning, easily

- Since the Iris data has $p = 4$ variables, there are $2^4 - 1 = 15$ possible subset models, which is low enough to reasonable perform combinatorial subset analysis
- The scikit-learn package has an object that performs recursive feature extraction, which is like reverse selection for regression
- It seems that this object won't work with our grid search for parameter tuning, though
- It's relatively simple to code our own procedure to perform the cross-validated parameter tuning on all possible subsets for this data

Feature Selection

- The result of performing the cross-validated grid search on all possible subsets of the Iris data is:

  Best Subset All Variables

  Best Model Non-homogenous linear polynomial with $c_0 = -1$

- For larger datasets, complete enumeration of all possible subsets of features is not reasonable, so more advanced techniques, such as the genetic algorithm, must be used

Why Python

- Several of you may be wondering why I'm showing Python, as opposed to MATLAB or R
- MATLAB is very expensive, and most companies won't pay for it; there is Octave, a FOSS version of MATLAB, but it is not used so much in the real world
- There are several reasons to prefer Python over R, R:
    - has a steep learning curve
    - is not designed to operate well in a business environment
    - is maintained by statisticians, not software developers (you wouldn't go to the world's best knee surgeon for brain surgery, would you?)
- Python provides us several benefits:
    - heavily used by IT and software developers, so it's easy to implement machine learning models in production environments
    - developed with a focus on readability and productivity
    - extremely flexible and extensible
- I use the Anaconda distribution of Python, which automatically installs many common packages used for scientific computing