Learning Theory

Lecture 3

Manuel Balsera

IEOR, Columbia University

February 2, 2018

Outline

1. Introduction
2. The Learning Problem
   - Training
   - Error Function
   - Hypothesis Space
   - Learning Algorithm
3. Model Evaluation
   - Training Error
   - Testing
4. Generalization Theory
   - VC Generalization Bound
   - Bias-Variance Tradeoff
5. Bias-Variance Decomposition
   - Bias-Variance Decomposition
   - Bias-Variance Trade-off
   - Bias-Variance Regimes
6. Conclusion
   - Recap
   - Conclusion
7. Distance and Text
   - Distance

Today's goals

Learning Goals Today

- Understand the components of a supervised learning problem.
- Understand the **Bias-Variance trade-off**.
- Understand how to process text to produce features suitable for learning.

- We will follow the notation of *Learning From Data* [1] for the learning process.
- A good, clear presentation of the bias-variance trade-off is *the ISL book*, page 33 and following [2].
- For the document distance section, a good reference is *An Introduction to Information Retrieval* [3], specially chapter 6.

Motivation

Consider the default's sklearn svm class:

```
class sklearn.svm.SVC(C=1.0,
                      kernel=‘rbf‘, degree=3, gamma=‘auto‘,
                      coef0=0.0, shrinking=True,
                      probability=False,
                      tol=0.001, cache_size=200,
                      class_weight=None, verbose=False,
                      max_iter=−1,
                      decision_function_shape=‘ovr‘,
                      random_state=None)
```

- There are lots of arguments to this class.
- Trying all possible combinations is not possible (or desirable).
- Many other classifiers have a constructor like that.
- We still need to chose between them classifiers, and make decisions on those parameters.
- We need some kind of **mental map** of what a classifier is trying to do.

The goal of this lecture is to set up a general **framework** for solving ML problems.

Today's Lecture

- Machine learning problems have very **broad** applicability.
- The same techniques can be re-used in many different contexts.

then

- Today we will discuss ML in the **abstract** to help set up a common **language** than we will use to describe all the different techniques
- Once we are deep into the details on a particular procedure within a particular application it is useful to step back and think what we are doing in the more abstract framework to get oriented.
- We will use a (hopefully familiar) **linear regression** problem as an example to make the theory concrete.
- Finally we will discuss a **practical problem**: text representations suitable for machine learning.

Machine Learning Components

The ingredients of a ML model are

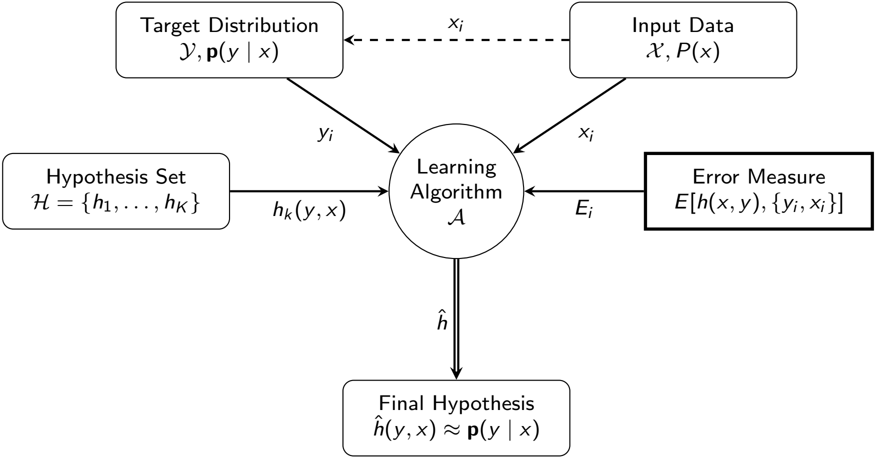
- a input data space $\mathcal{X}$, with a probability distribution $P(x)$.
- a Target space $\mathcal{Y}$ with an unknown conditional probability $\mathbf{p}(y \mid x)$.
- a sequence of training data $y_i, x_i \in \mathcal{Y} \times \mathcal{X}$ generated from P and $\mathbf{p}$.
- $y_i|_{x_i}$ is **independent** of $y_j|_{x_j}$ if $i \neq j$.
- A set of hypothesis $h(y, x) \in \mathcal{H}$ that approximate $\mathbf{p}$.
- An loss (error) function $E(h, \{x_i, y_i\})$ that measures how far h is from $\mathbf{p}$.
- A learning algorithm $\mathcal{A}$ to select a good $h \in \mathcal{H}$ given E and the training data.

Goal of ML

Find $h \in \mathcal{H}$ with as small as possible $E(h, \{y_i, x_i\})$ on new **unseen** data samples $\{y_t, x_t\}$.

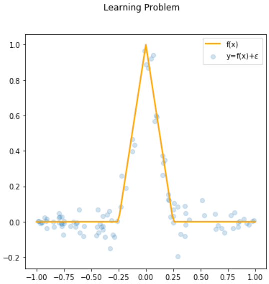
Note: many times we will limit ourselves to **point estimates** of y, in which case $y = h(x)$ is a function of x only.

The Learning Problem: Training



Example Learning Problem

- $x \in \mathcal{X} = (-1, 1) \in \mathbb{R}$
- $y \in \mathcal{Y} = \mathbb{R}$
- $P(y|x) = \mathcal{N}(f(x), \sigma(x))$, i.e. $y = f(x) + \epsilon$ with $\mathrm{Var}(\epsilon) = \sigma(x)^2$
- $f(x) = 1 - 4|x|$ for $|x| < \frac{1}{4}$, 0 otherwise.
- $\sigma^2(x) = \sigma_0(1 - x^2)$



Goal: Learn $P(y|x)$, i.e, find best $\hat{h}(x) \approx f(x)$

Error Function

The Error Function is really a functional
Its arguments are

- $h(y, x)$: our estimate of $p(y \mid x)$
- a sequence of observations $\{y_i, x_i\}$

It returns a measure of how well h approximates the conditional distribution of y:

$$E[h, \{y_i, x_i\}_{i=1}^{N}] \to \mathbb{R}^+ \tag{1}$$

If the hypothesis set provides only a point estimate $\hat{y} = g(x)$, we can think of E as a function of $\hat{y}_i = g(x_i)$

Error Function Examples

- Log Likelihood:

$$E[h,\{y_i,x_i\}] = -\frac{1}{N}\sum_{i=1}^{N}\log h(y_i,x_i) \tag{2}$$

- Classification Error for Binary Target Variables:

$$E[h,y_i,x_i] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}(y_i=1)(1-h(1,x_i)) + \mathbb{1}(y_i=0)h(1,x_i) \tag{3}$$

- For point estimates, least square error:

$$E[g,y_i,x_i] = \frac{1}{N}\sum_{i=1}^{N}(y_i-g(x_i))^2 \tag{4}$$

Note For our example problem we will use least square error, which is equivalent to max likelihood.

Choice of Error Function

The error function is an input to a ML problem.
Choice depends on the cost errors.
For example: medical diagnosis:

- C_{FP}: cost of false positive (wongly diagnosed with cancer).
- C_{FN}: cost of false negative (missed diagnosis of cancer).

$$\begin{aligned} E[h,\{y_i,x_i\}] = & \frac{1}{N}\sum_{i=1}^{N} C_{\mathrm{FP}} \quad \mathbb{1}(y_i=1)(1-h(1,x_i)) \\ + & \frac{1}{N}\sum_{i=1}^{N} C_{\mathrm{FN}} \quad \mathbb{1}(y_i=0)h(1,x_i) \end{aligned} \tag{5}$$

Choice of Error Function

The choice of error function is **Domain Specific**, it should be determined at the beginning of each learning task.

For simplicity, in class we will mostly focus in **Classification Error Rate**.

Hypothesis Space

- In is important to be clear about what space of models $\mathcal{H}$ we are searching when we solve the problem.
- Particular algorithms $\mathcal{A}$ may only work for some hypothesis spaces $\mathcal{H}$ (i.e. Linear Regression for Linear Models). But, in many cases, we can change $\mathcal{H}$ and $\mathcal{A}$ independently
- Larger $\mathcal{H}$ usually fit the data better (lower training error), but may perform worse on new data (overfitting).
- Any model choice taken based on the labeled data (y_i, x_i) must be "accounted for"
  If we choose between a Tree model and a Logistic Regression model based on performance on the training set, then $\mathcal{H} = \mathcal{H}_{\text{Tree}} \cup \mathcal{H}_{\text{Logistic}}$
- Choosing a model *a priori* without looking at the data, or looking only at x_i, but not y_i does not increase the size of $\mathcal{H}$.

D_{VC} Dimension

For **classification problems** the size of $\mathcal{H}$ is quantified by the D_{VC} dimension

The **Vapnik–Chervonenkis** (D_{VC}) Dimension

D_{VC} is the largest number of points for which $\mathcal{H}$ can generate any combination of class labels y_i, for $i = 1, \ldots D_{VC}$ and any set of points x_i in $\mathcal{X}$.

- For linear regression D_{VC} is equal to the number of *independent* fitted parameters.
- For more general models D_{VC} is usually impossible to compute.
- In general we will use the number tunable model specific parameter as a proxy for model complexity.

Hyper Parameters

- It is common to have a *family* of different hypothesis spaces $\mathcal{H}_\kappa$ parametrized by a **hyperparameter** κ.
  For example, if $\mathcal{X} = \mathbb{R}$ we can consider linear regression on powers of x, $1, x, x^2, \ldots, x^D$, then D is a hyperparameter.
- If κ is chosen **a priori**, without resort to the training data the hypothesis set is

$$\mathcal{H} = \mathcal{H}_\kappa \tag{6}$$

- If we **choose** the best κ within the model family:

$$\hat{h} = \arg\min_{\hat{h}_\kappa} E(\hat{h}_\kappa, \{x_i, y_i\}) \tag{7}$$

  The hypothesis space is really

$$\mathcal{H} = \bigcup_\kappa \mathcal{H}_\kappa \tag{8}$$

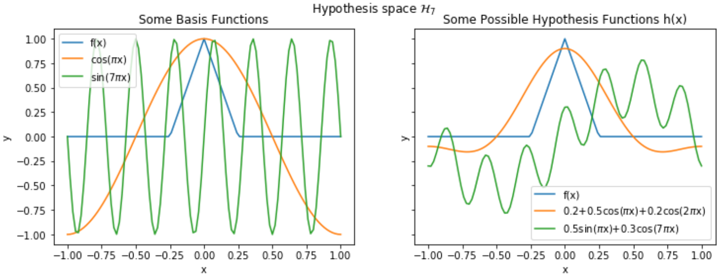
- We will work later on the course of methods to do this carefully accounting for the extra model complexity.

Hypothesis Space for Learning Example

Continuing our learning example, we define

$$\mathcal{H}_K = \mathrm{span}\left(\{\cos(k\pi x), \sin(k\pi x)\}_{k=0}^{K}\right) \tag{9}$$

That is, all the linear combinations of $\sin(k\pi x)$ and $cosk\pi x)$ for $0 \leq k \leq K$.



Learning Algorithm

Learning Algorithm

A learning algorithm $\mathcal{A}$ is a procedure to find an **approximation** to

$$\hat{h} = \arg\min_{h \in \mathcal{H}} E(h; \{y_i, x_i\}) \tag{10}$$

Exact Algorithm Only for some problems like Linear Regression.

Grid Search if $\mathcal{H} \subset \mathbb{R}^D$ exhaustive search on a D-dimensional grid. Unfeasible for large D.

Random Search sample $\mathcal{H}$ according to some probability distribution.

Steepest Descent applicable to a wide set of learning problems.

Specialized Algorithm Some error functions (hinge error functions) have specialized algorithms (convex optimization).

Learning Algorithm Performance

- Algorithm choice affects the **computational complexity** of a learning problem.
- Provided the approximation $\hat{h}$ is good, it does not affect which hypothesis $\hat{h}$ is selected (**generalization performance**)

Learning Algorithm for Our Example

In our example, possible functions are parametererized as

$$h(x; \{w_k, w'_k\}_{k=0}^{K}) = \sum_k w_k cos(k\pi x) + w'_k sin(k\pi x) \tag{11}$$

To simplify notation we will **flatten** the parameters and write

$$h(x; W) = \sum_d w_d h_d(x) \tag{12}$$

with $h_d(x) = cos(d\pi x)$ if $d \leq K$ and $h_d(x) = sin((d-K-1)\pi x)$ if $K < 2(K+1)$ **??**
With this notation the Error function is a quadratic function of $W = \{w_d\}$

$$E[W, \{y_i, x_i\}] = \frac{1}{N} \sum_i \left(y_i - \sum_d w_d h_d(x_i) \right)^2 \tag{13}$$

If we define the matrix $H_{i,d} = h_d(x_i)$ the **optional** solution to our problem is

$$\hat{W} = (H^T H)^{-1} H^T Y \text{ and } \hat{h}(x) = h(x, \hat{W}) \tag{14}$$

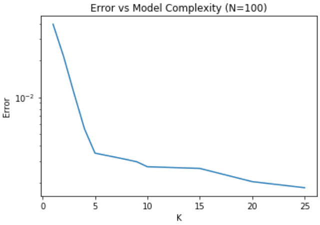
This are multivariate linear regression **normal equations** and should be familiar.

Training Error

Training Error

In a set of nested models $\mathcal{H}_\kappa$ training error is a **decreasing** function of model complexity κ.

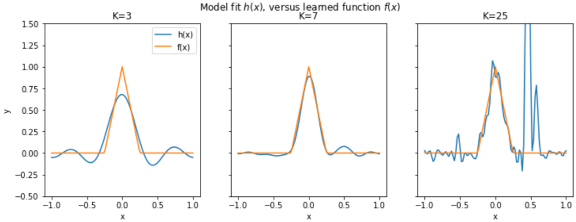
- Given nested models $\mathcal{H}_{\kappa_0} \subset H_{\kappa_1}$ if $\kappa_0 \leq \kappa_1$
- $E(\hat{h}_{\kappa_0}; \{y_i, x_i\}) = \min_{h \in \mathcal{H}_{\kappa_0}} E(h; \{y_i, x_i\})$
- but $\hat{h}_{\kappa_0} \in \mathcal{H}_{\kappa_0} \subset \mathcal{H}_{\kappa_1}$
- therefore $E(\hat{h}_{\kappa_0}; \{y_i, x_i\}) \leq \min_{h \in \mathcal{H}_{\kappa_1}} E(h; \{y_i, x_i\}) = E(\hat{h}_{\kappa_{\cancel{0}1}}; \{y_i, x_i\})$



In our running example ($\kappa = K$):

Final Hypothesis: Example Problem

Given a sample of $N = 100$ observations, the optimal solution $\hat{h}(x)$ (minimizing error on the training sample) with $K = 3, 7$ and 25 are

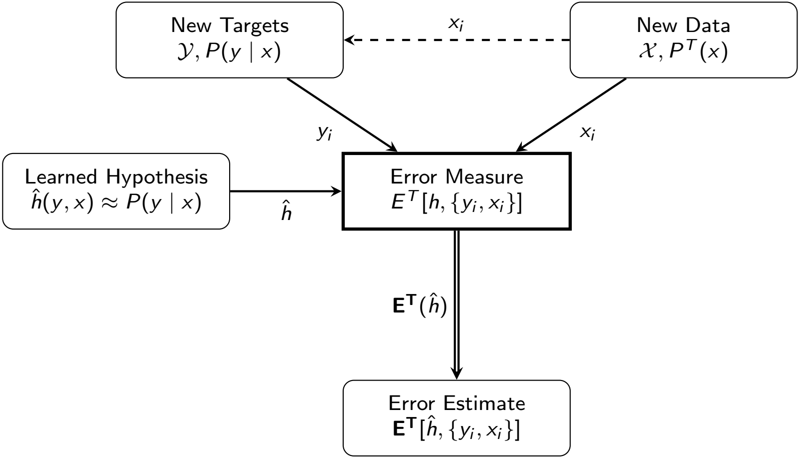


- $K = 3$ seems not to fit well (not enough flexiblity in the functions).
- $K = 25$ **over-fits**.
- But training error is alsways decreasing with K.
- Without the true function $f(x)$, how can measure fit quality?

Test Data

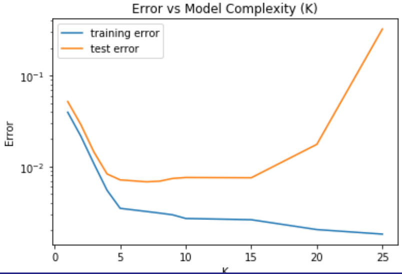
We need to use new **test data** to measure generalization performance.

The Learning Problem: Testing



The Generalization Problem

- We use the training set y_i, x_i to select the hypothesis h closest to the true $P(y \mid x)$.
- Under which conditions can we expect the training's set error rate E_{training} to be close to the test set E_{test}?
- If they are far apart, optimizing the training error rate is pointless.
- In general we expect larger number of samples N to bring test and training error closer together
- Larger hypothesis complexity pulls apart training and test errors because overfitting



VC Generalization Bound

For **Binary Classification** problems there is the

VC Bound

$$\mathbb{E}[E_{\text{test}}] \lessapprox E_{\text{training}} + k\sqrt{\frac{D_{\text{VC}} \log N}{N}} \tag{15}$$

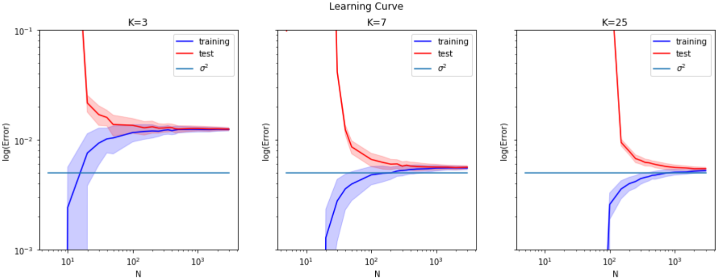
- This is called the **Vapnik–Chervonenkis** generalization bound.
- D_{VC} is the Vapnik-Chervonenkis dimension. For a liner model is equal to the number of fitted parameters.
- The proportionality constant k is very conservative in practice.
- The bound is useful **qualititively**: keep ratio of points to parameters constant.

Learning Curve

Learning Curve

A plot of training and testing error as a function of sample size

- Training Error **grows** with sample size
- Testing Error **decreases** with sample size
- For large enough size, training and testing errors converge



Bias-Variance Decomposition

With some simplifying assumptions we can find a nice decomposition of test error for the point estimate $\hat{f}(x)$ of

$$y = f(x) + \sigma(x)\,\epsilon, \quad \epsilon \sim \mathcal{N}(0,1) \tag{16}$$

Consider an average over all possible **training data** sets $\mathcal{D}$ and noise ϵ.

$$\mathbb{E}_{\mathcal{D},\epsilon}[(y - \hat{f}(x))^2] = \mathbb{E}_{\epsilon}[(y - f(x))^2] + (f(x) - \bar{f}(x))^2 + \mathbb{E}_{\mathcal{D}}[(\hat{f}(x) - \bar{f}(x))^2] \tag{17}$$

using

Irreducible Error $\sigma^2(x) = \mathbb{E}_{\epsilon}[(y - f(x))^2]$.
No method can have error lower than this.

Average Prediction $\bar{f}(x) = \mathbb{E}_{\mathcal{D}}[(\hat{f}(x)]$

Bias $\left\{f(x) - \bar{f}(x)\right\}^2$

Variance $\mathbb{E}_{\mathcal{D}}\left[\left\{\hat{f}(x) - \bar{f}(x)\right\}^2\right]$

we obtain the decomposition

Bias-Variance Decomposition

$$\mathbb{E}_{\mathcal{D},\epsilon}[(y - \hat{f}(x))^2] = \sigma^2(x) + \mathrm{Bias}(x) + \mathrm{Variance}(x) \tag{18}$$

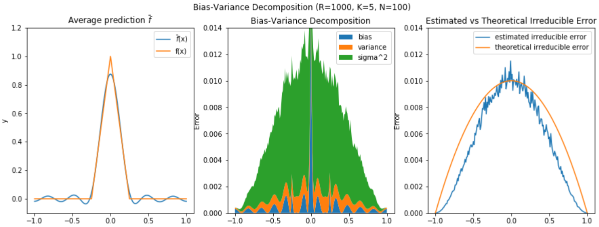
Bias-Variance Decomposition in our Example

- We approximate the average over all training data samples by generating R data samples from $P(X)$ and $P(Y|X)$
- let's call $\hat{f}_r$ to the best function approximation on trial $r = 1, \ldots, R$

$$\bar{f}(x) \approx \frac{1}{R} \sum_r \hat{f}_r(x) \tag{19}$$

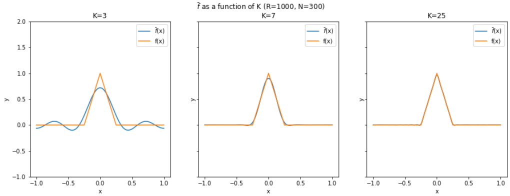
$$\mathrm{Bias}(x) \approx \left\{ f(x) - \bar{f}(x) \right\}^2 \tag{20}$$

$$\mathrm{Variance}(x) \approx \frac{1}{R} \sum_r \left\{ \hat{f}_r(x) - \bar{f}(x) \right\}^2 \tag{21}$$

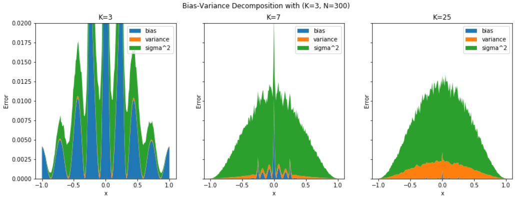


Bias-Variance Dependence of Model Complexity

Bias decreases with model complexity



But Variance increases due to **over-fitting**

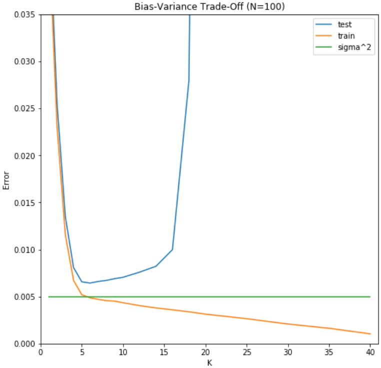


Bias-Variance Trade-off

For a nested family of hypothesis sets $\mathcal{H}_\kappa$ and a fixed training data size N there is a

Bias-Variance Trade-Off

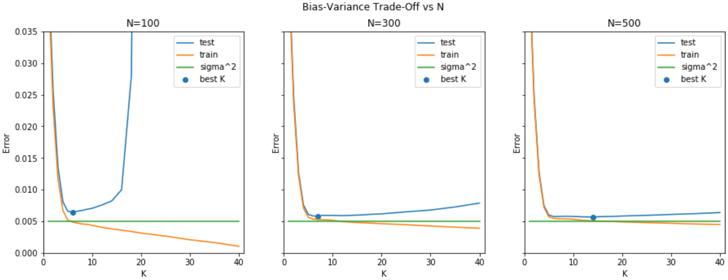
- Bias **decreases** as a function of model complexity κ.
- Variance **increases** as a function of model complexity.
- There is an optimal value of κ where Bias and Variance balance each other.



Note: Notice how training error is below irreducible error σ^2, that is **overfitting**!

Bias-Variance Trade-off and Sample Size

As sample size N increases, model variance decreases, and the bias-variance trade off gets pushed towards **more complex** models

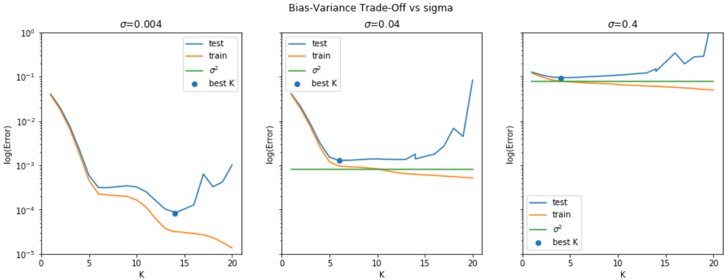


Bias-Variance Trade-off and Irreducible Error

Irreducible Error

Irreducible error is a measure of the variability of y **not explained** by x.

- Theoretical limit to ML performance.
- We can never do better.
- For realistic problems may be impossible to estimate.
- For fixed N, an increase of irreducible error pushes the bias-variance trade off towards **simpler** models



Learning Checklist

Repeat this procedure in a loop

- get $\mathcal{A}$ working: $E[\hat{h}] \approx \min_{h \in \mathcal{H}} E(h)$
- Performance is sufficient: **stop**
- Error dominated by

Bias $E_{\text{train}} \approx E_{\text{test}}$: Increase model complexity.

Variance $E_{\text{train}} \ll E_{\text{test}}$:

- Best: Increase N (training input size)
- Ok: Decrease model complexity.

Irreducible Error $E_{\text{test}} \approx \sigma^2$: Find new input variables $\mathcal{X}'$.

SVM classifier Revisited

Let's sort the arguments to the SVM classifier we started with based on what we learned today

Learning Algorithm $\mathcal{A}$ coef0, tol,cache_size,max_iter,random_state,shrinking

Hypothesis space $\mathcal{H}$ C, kernel, degree,gamma

Error Function C,class_weight, decision_function_shape,

Output probability, verbose

We can use the checklist to decide which of this arguments may help us move forward with our learning problem.

Conclusion

- We have studied the components of a machine learning problem.
  - Input $\mathcal{X}$ and target $\mathcal{Y}$ domains
  - Hypothesis space $\mathcal{H}$
  - Error function E.
  - Learning Algorithm $\mathcal{A}$.
- These concepts are **very general**, can be applied to a large number of problems.
- **Out of sample** testing is required to asses performance of ML procedures.
- Bias-Variance decomposition provide us with a framework to understand the trade-offs involved in learning.

Distance

A function on $\mathcal{S} \times \mathcal{S} \to \mathbb{R}_0^+$ with the following properties is called a distance:

Distance (Metric)

Postive Definite $D(s_1, s_2) \overset{>=}{\leq} 0$ and $D(s_1, s_2) = 0 \Leftrightarrow s1 = s2$.

Symmetric $D(s_1, s_2) = D(s_2, s_1)$.

Subadditive $D(s_1, s_3) \leq D(s_1, s_2) + D(s_2, s_3)$

If a function has those properties except for subadditivity is called a **semi-metric** $\mathcal{S}$ could could be very general

- $\mathbb{R}^K$, the space of K-dimensional vectors $\mathbb{R}^K$
- the space of all $R \times C$ color images
- the space of all possible English text documents

Distance II

If $\mathcal{S} = \mathbb{R}^K$ we can define

1. Weighed Euclidean Distance:

$$D(s_1, s_2) = \sqrt{\sum_{k=1}^{K} w_k (s_{1,k} - s_{2,k})^2} \qquad (22)$$

2. More generally, we can weight with an arbitrary, positive-defined symmetric matrix $W = (w_{k,k'})$

$$D(s_1, s_2) = \sqrt{(s_1 - s_2)^T W (s1 - s2)} \qquad (23)$$

3. L_q norm

$$D(s_1, s_2) = \left\{ \sum_{k=1}^{K} |s_{1,k} - s_{2,k}|^q \right\}^{\frac{1}{q}} \qquad (24)$$

Especial cases are the euclidean norm L_2, the Manhattan norm L_1, and the supremum norm L_∞.

Similarity

- A positive definite symmetric function v sim on $\mathcal{S} \times \mathcal{S} \to \mathbb{R}_0^+$ is called a **similarity measure**.

sim(s1,s2)<=1

- If $\mathrm{sim}(s_i 1 s_2) \leq 1$ and $\mathrm{sim}(s_1, s_2) = 1 \Leftrightarrow s_1 = s_2$ sim is called **discerning**
- A discerning similarity induces a semi-metric $d(s_1, s_2) = 1 - \mathrm{sim}(s_1, s_2)$

An example in $\mathbb{R}^D$ is the

cosine similarity

$$\cos(s_1, s_2) = \frac{s_1 \cdot s_2}{|s_1||s_2|} \tag{25}$$

.

- Cosine distance is not a discerning similarity ($d(x_1, x_2) = 0$ and $x1 \neq x2$). But we still talk about the **cosine** distance.
- Usually our problem is things are two far apart, not too close.
- See Wolfram for a large collection of similarities and related distances.

Document Distance

We would like to define a distance metric between

1. A break-in at the U.S. Justice Department's World Wide Web site last week highlighted the Internet's continued vulnerability to hackers. Unidentified hackers gained access to the department's web page on August 16 and replaced it with a hate-filled diatribe labeled the "Department of Injustice" that included a swastika and a picture of Adolf Hitler...
2. The U.S. Postal Service announced Wednesday a plan to boost online commerce by enhancing the security and reliability of electronic mail traveling on the Internet. Under the plan, businesses and consumers can verify that e-mail has not been tampered with...

But **How**?

Bag of Words Representation

Bag-of-Words Model

represents a document as a multiset (bag) of its words, disregarding grammar and even word order but (optionally) keeping multiplicity.

It is really a family of related representations.
A document s will be represented by a vector X_d

- The index $d = 1, \dots V$ has as many dimensions as there are words in the **Vocabulary**.
- The component X_d can be one of

Set	1 or 0 representing if the word w is present or not in document.
Count	number of times word w appears in document.
Tf-idf	word d count n_d is scaled by the inverse number of documents N_d where d appears in **corpus** of documents.

See https://en.wikipedia.org/wiki/Tf-idf for **many more** weighting choices.

Term Frequency - Inverse Document Frequency (TF-IDF)

In **Information Retrieval** applications (given a query find closest match in a corpus) is traditional to score document distance using a vector of **Tf-Idf** features:

Tf-Idf Features

Given word d in vocabulary $1, \dots V$

Xd:component

$$X_d = \mathrm{Tf}_d \cdot \mathrm{Idf}_d \tag{26}$$

Term Frequency Tf_d is the number of times word d appears in document (word count). It is not really a frequency that this is the traditional nomenclature.

Inverse Document Frequency : `sklearn` defaults to a *smoothed* definition

$$\mathrm{Idf}_d = \log\left(\frac{1+C}{1+\hat{N}_d}\right) + 1 \tag{27}$$

where

- C is the number of documents in corpus.
- $\hat{N}_d$ is number of documents containing word d.

Document Cosine Distance

Given any of the representations X_d above we can define the document's

cosine distance

$$D(X_1, X_2) = 1 - \frac{X_1 * X_2}{\sqrt{|X_1|^2 |X_2|^2}} \tag{28}$$

where $*$ is the vector dot product, $|X_i|$ is the usual vector norm.
The **normalization** corrects for the difference in length of the documents. It would not be appropriate if document length is a important feature of the problem at hand.

What is a “word” anyways?

We still have a lot of **choices** we could make:

- Does punctuation ".", "?" count as a word?
- is "New York" one word or two, what about "U.S."?
- Do "car" and "cars" count as different words?
- What about "safe" and "safely"
- What do we do about miss-spelled words, do we try to fix them?
- Do we consider different capitalizations of the same word: "Car" versus "car"?
- Do we include high frequency, low information content words such as "a" and "the" in our bag of words?

All this choices are **problem dependent**. If we have a particular ML problem to solve we will use our **domain knowledge** to make a decision in the context of that specific task.

NLTK

We will rely on python's NLTK (Natural Language Toolkit) to perform tasks that require domain knowledge about text processing:

tokenization breaking character streams into words

stemming normalizing words into their roots: "cars" → "car", "safely" → "safe"

stop words removal of high frequency, low information words that we will consider just **noise** and ignore.

DiGram Text Model

One limitation of the **Bag of Words** model is that it completely ignores **word order**. We can generalize **by** defining the

Digram Text Representation

the vector of counts s_w where w runs over **ordered pairs** of the words that appear in text

- In the bag of words representation w could be “cat”, “dog”,”black”,”nice”,...
- In the Digram representation w could be “black cat”,”nice cat’,”nice dog”,...
- potentialy w runs over $V \times V$ a **very large** dimensional space!.
- A lot of possible diagrams are empty due to constrains of language, very **sparse** representation.
- All the distance, tf-idf, etc tools from the bag of word representation carry over.

N-Gram Text Model

One limitation of the **Bag of Words** model is that it completely ignores **word order**. We can generalize it be defining the

Digram Text Representation

of text is the feature vector of the counts s_w where w runs over **ordered pairs** of the words that appear in the document.

- In the bag of words representation w could be “cat”, “dog”,”black”,”nice”,...
- In the Digram representaiton w could be “black cat”,”nice cat’,”nice dog”,...
- potentialy w runs over $V \times V$ a **very large** dimensional space!.
- A lot of possible diagrams are empty due to constrains of language, very **sparse** representation.
- All the distance, tf-idf, etc tools from the bag of word representation carry over.

n-Gram Representation

n-gram Text Representation

of text is the feature vector of the counts s_w where w runs over **ordered n-tuples** of the words that appear in the document.

Example: **“the black cat purrs happily”**

1-gram	[“the”,”black”,”cat”,”purrs”,”happily”]
2-gram	[“the black”,”black cat”, “cat purrs”,”purrs happily”]
3-gram	[“the black cat”,”black cat purrs”,”cat purrs happily”]

- n-grams potentially V^n dimensional!
- n-grams encode information:

syntactic	“the cat” more likely ($\gg$) than “cat the”
semantic	“cat purrs” $\gg$ “dog purrs”; “purrs happily” $\gg$ “purrs angrily”.

- need lots of data: Google computed English 5-grams trained over the **whole internet**!

Data Pipeline

Data Pipeline

a series of transformations applied sequentially to input data to generate a set of features suitable for a ML algorithm

For document processing we used `NLTK` to generate the text **features** s_w.
This a **general pattern**: we will need a **domain specific** library to do **feature engineering**.
For example: for image processing, we would use `scipy.ndimage` to do

- color manipulation
- image cropping
- image resolution changes
- etc.

You should expect to spend **70%** of a ML project time working on the data pipeline.

Bibliography

Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin.
Learning From Data.
AMLbook.com, 2001.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani.
An Introduction to Statistical Learning with Applications in R.
Springer New York Inc., 2013.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze.
Introduction to Information Retrieval.
Cambrige University Press, 2008.
https://nlp.stanford.edu/IR-book/information-retrieval-book.html.