Large Margin Classification Using the Perceptron Algorithm (Part 2)

Henry Tan

Georgetown University

April 13, 2015

Analysis - Theorem 3

Theorem 3

Assume all examples $(\vec{x}, y)$ are generated i.i.d. Let E be the *expected* number of mistakes that the online algorithm A makes on a randomly generated sequence of $m+1$ examples. Then given m random training examples, the expected probability that the randomized leave-one-out conversion of A makes a mistake on a randomly generated test instance is at most $E/(m+1)$. For the deterministic leave-one-out conversion, this expected probability is at most $2E/(m+1)$.

Analysis - Theorem 3 - Intuition

Randomized

E - expected number of mistakes
$m + 1$ - total number samples

Therefore expected probability that the last sample is a mistake is $E/(m + 1)$. Selecting a random subsequence has at most the same error probability.

Deterministic

Shouldn't be difficult but can't think of the right formulation.

Analysis - Theorem 3 - Corollary 1

Corollary 1

Simple application of Theorem 2 and 3.

Analysis - Theorem 3 - Corrollary 2

Prediction vector is only changed when a mistake occurs and only changes based on the mistake $\rightarrow$
error probability depends only on the mistakes.

Re-define

- R to be the maximum length over all mistakes
- D to be the deviation over all mistakes

Very similar to SVM - accuracy depends only on a small fraction of "errors"

Analysis - Theorem 4

From another paper -
Similar to Theorem 3 but predicting only with the final prediction vector.

Probability of error on x_{m+1} test instance
$\leq \frac{1}{m+1} E[min\{k, (\frac{R}{\gamma})^2\}]$
Main difference - no dependence on the deviation.

Authors mention that due to dependence on k, number of mistakes, this indicates that running for a single epoch, $T = 1$, might be better than to convergence.

Incorrect : This is not an implication of this proof. The proof provides an expected upper bound on the error, not expected error.

Theorem 4 - Question

Question

Grace -
3. Compare Theorem 4 and the bound for expected error of SVM, also by Vapnik.

Theorem 4 - comparison with SVM bound

Bound on expected error of SVM

Don't have access to the book *Statistical Learning Theory* but I found this paper[1] which cites it (but also may be a similar but not quite the bound being referred to).

$p_{err} \leq \frac{1}{l} E[\frac{R^2}{\gamma^2}]$

Where

$\gamma =$ size of margin

$R =$ maximal distance of each training sample from some optimally chosen vector

$l = \#$ of training samples

[1] `http://research.microsoft.com/en-us/um/people/manik/projects/trade-off/papers/chapelleml02.pdf`

Theorem 4 - comparison with SVM bound 2

SVM - $\frac{1}{l} E[\frac{R^2}{\gamma^2}]$

Perceptron - $\leq \frac{1}{m+1} E[min\{k, (\frac{R}{\gamma})^2\}]$

Qualitatively

Essential support vectors are basically the same as the "mistakes" of the perceptron algorithm

Allowance for some optimally chosen vector instead of from the origin is probably because SVM is finding a hyperplane so if all vectors are translated/rotated identically in the space, the problem is the same. Maybe.

Contribution - Question

Question

Yifang -
2. So the real contribution of this paper is proving the upper-bound of mistakes in both linearly separable case and linearly inseparable case? It does not really compare against SVM?

The contribution is the voted-perceptron (which is a combination of various ideas).
The proofs show that the error bound is similar to that given for SVM.

Brief comparison to SVM at the end where SVM wins in accuracy.
However, the Perceptron algorithm is conceptually simpler.

SVM Comparison - or lack thereof

Questions

Yuankai 3. The authors claim that their algorithm is much faster than SVM. Is it just asymptotically faster or actually runs faster? In their evaluation, I didn't see them comparing actual running time of their algorithm with SVM.

Brendan 3. Is there no runtime comparison to SVM? I thought that was the major advantage?

Grace 1. Could you please show the connection between today's paper and svm, and the connections between today's paper and online learning? Please list the similarities and differences.

SVM Comparison

Authors Claim

Pg 2 Simple and easy to implement

Pg 2 Expected generalization error ... almost identical to the bounds for SVM in the linearly separable case

Table 3 Comparison of support vectors and error rate for polynomial $d = 3$ kernel.

Table 3 Summary

SVM has slightly lower error rate compared to large T perceptron.

# of support vectors is much smaller for Perceptron even with $T = 30$

SVM Comparison

Parameters

$d = \#$ of dimensions
$n = \#$ of training samples
$k = \#$ of errors/support vectors
$c =$ kernel computation complexity

Standard Perceptron Complexity

Training - $O(dn)$ Test - $O(d)$

Vote/Average Perceptron Complexity with Kernel

Training - $O(c * k * n)$ Test - $O(c * k)$

SVM with Kernel

Training - $\Omega(cn^2)$, $O(cn^3)$ Test - $O(c * k)$

SVM Comparison - Similarities and differences

Similarities

- Margin based - Both only consider the observations which disagree with some prediction function
- Linear but can use Kernels

Difference

Perceptron can only separate data that is separable by a hyperplane going through the origin.
SVM can use any hyperplane.

SVM Comparison - Summary

Summary

The Perceptron Algorithm is :

- Simpler implementation and concept (Not an optimisation problem)
- Potentially faster training time (no need to solve quadratic optimisation, fewer support vectors)
- Probably faster when running predictions.

SVM is:

- More accurate

Kernel Trick

Questions

Yuankai 2. (Basics) What is kernel function and kernel method? How are they used in machine learning? Can you give a simple example to illustrate the idea?

Brad 3. Can you do an overview of kernel functions in general and how they relate to dimensionality?

Kernel Trick

Why?

Perceptron algorithm (and SVM without kernels) work best with linearly separable data.
However, even 2D Data may not be linearly separable.

Transforming the data into higher dimensions can be expensive for large numbers of dimensions, e.g., computing an infinite dimensional vector, or computing expensive transforms, e.g., $x' = x^{100}2^y$.

Kernel functions - computes inner product between transformed vectors using a shortcut - simpler functions that take original vectors as input.

Possible with the perceptron algorithm (and SVM) since observations are only used in inner products.

Kernel Trick

Fundamental Idea (Recap)

Kernels are a way to compute a value using 2 vectors in such a way that it is the equivalent to the inner product between 2 other related vectors in a higher dimensional space.
i.e. -
Given vectors x, y and function k, compute some value $q = k(x, y)$.
The correct k yields $k(x, y) = q =< x'|y' >$,
for vectors x', y' where x' is related to x, and y' is related to y

Kernel Trick - Questions 2

Question

Tavish -
3)While converting the voted-perceptron algorithm to a kernel function, how is the dimensionality of $\Phi(x)$ and $\Phi(y)$ is determined? And how will this affect the accuracy of classification?

Dimensionality of $\Phi(x), \Phi(y)$ is determined entirely by the Kernel chosen. Only certain functions can be chosen as kernels. Expansion of the function may tell you which basis expansion it implies.

Classification accuracy - If the data distribution is more separable in the high dimensional space induced by the Kernel, classification accuracy will be better.

Kernel - Vector Addition and Inner Product

Original prediction vector: effectively a sum of observations (with signs).
Kernel Based prediction vector : a sum of observations, but in the higher dimension space.

Cannot sum the mistake vectors then compute the Kernel function against a new observation (due to non-linearity).

Kernel - Vector Addition and Inner Product 2

Kernel function $K(x, y)$, mistakes $\vec{x_1}...\vec{x_k}$, next observation $\vec{x}$

Do Not

Set Prediction vector $\vec{v} = \sum_{j=1}^{k} y_j \vec{x_j}$

Straightforward computation of - $K(\vec{x}, \vec{v})$

OR

$\vec{v} = \sum_{j=1}^{k} y_j f(\vec{x_j})$ where $f()$ is the basis expansion function

Do

$$K(\vec{x}, \vec{v}) = \sum_{j=1}^{k} y_{i_j} K(\vec{x}, \vec{x_{i_j}})$$

Kernel - Vector Addition and Inner Product 3

Cost

With k mistakes, this incurs a cost of k Kernel Function evaluations.

However, the voting improvement that they propose also incurs a cost of k Kernel evaluations (one evaluation for each unique prediction vector).

But the k Kernel Computations for basis expansion are the same as the ones required for the voting procedure.

Experiments - Data

NIST OCR Database

Labelled digital images of handwritten digits

- 60,000 training, 10,000 test samples
- 784 dimensions – $28 * 28$ pixels

The authors use various versions of the perceptron algorithm with polynomial kernels up to $d = 6$

Experiments - Multiclass Classification

Standard perceptron algorithm is a binary classifier.
Authors perform multiclass classification by using a series of binary classifiers - one for each class

To predict instance x, they compute the score using each classifier and choose the highest score as the class.

Experiments - Multiclass Classification

Question

Brendan -
2. Is it normal to reduce a multi-class problem to a series of binary classification problems?

As far as I can tell, yes. It seems like a straightforward blackbox style generalisation of binary classification to multiclass.

Experiments - Types

Preliminary

Remember that during training, multiple prediction vectors $v_1, ..., v_k$ are generated and each has their own respective weights (amount of time before an error occurs).
Let x be the test sample

Types of Perceptron prediction

Last Only the final prediction vector v_k is used

Vote *Author's* - take weighted mean of $sign[< v_i | x >]$ for each v_i

Avg Weighted mean of each $< v_i | x >$

Random Randomly selected prediction vector is used.

Types of Perceptron prediction - Normalisation

Normalisation of the prediction vectors

Unimportant with binary classification (sign is enough),
but may affect results for multiclass classification since the dominant score is chosen as the winner.

Results

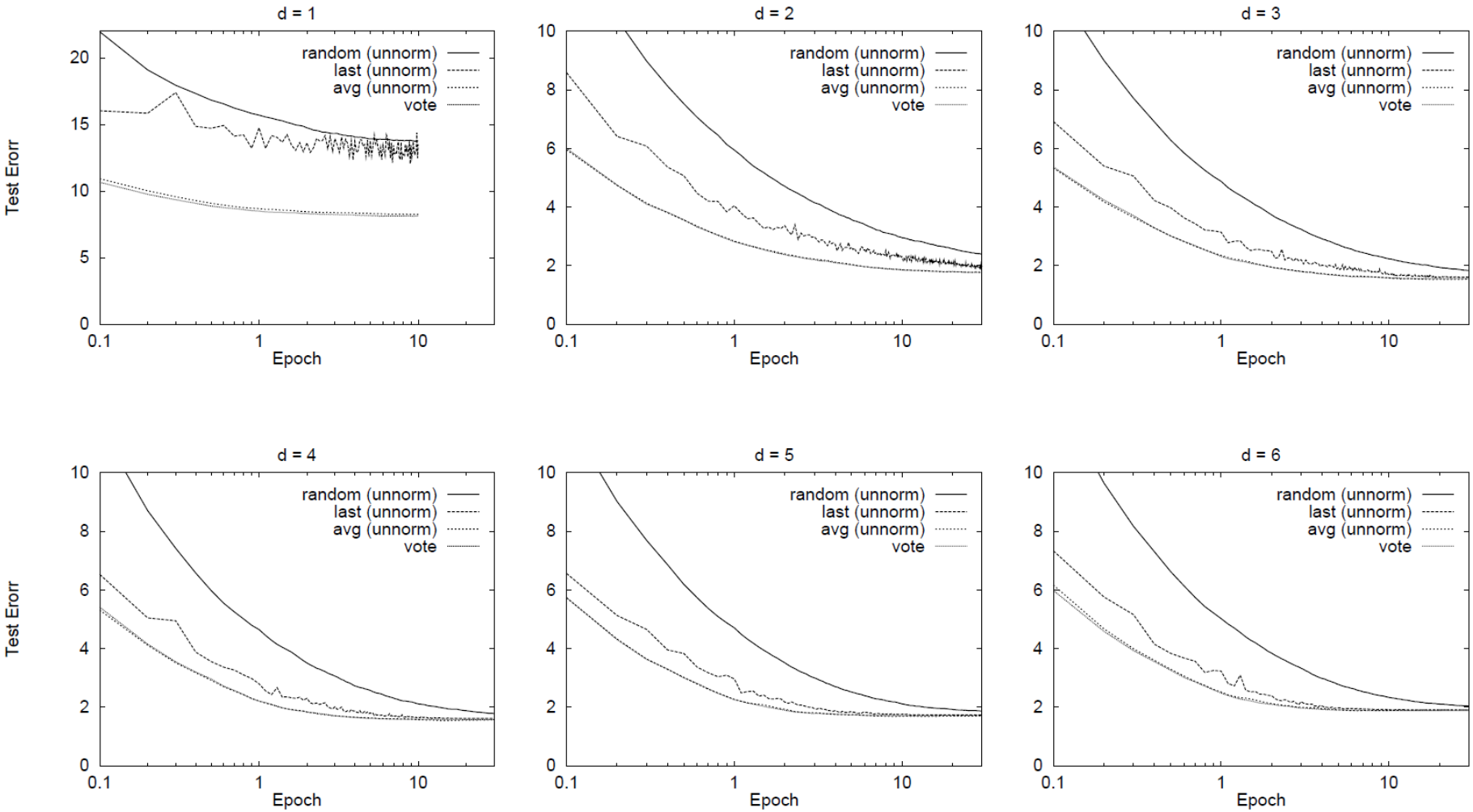
Question

Jiyun -
3. What does T means in Table 1?

The authors call it the *epoch* number.
Each epoch is one run through the training data.

The algorithm doesn't guarantee an optimal or consistent solution in a single pass. However, on linearly separable data, the authors show that the prediction vector eventually converges.

Results - Graph



Results - Speed

Mistakes vs SupVec

Mistakes - whenever the prediction vector disagrees with a label.
SupVec - Unique observations which are mistakes (possible with $T > 1$)

Algorithm complexity is dependent on SupVec, not mistakes, since Kernel computations can be re-used.

Linear Separability

Occurs around $d = 5$. The algorithm is actually much faster at higher d due to the linear separability, reducing the number of errors and support vectors.

Results - Summary

Trends

Error rate drops as d increases till 4 or 5.

Error rate drops as T increases, until convergence between $10 \leq T \leq 30$.

Misc

Random is the worst performing method, worse than Average, even though it had tighter proof bounds on the error rate.

Normalisation, or lack theoreof, shows little effect on the results.

Previous work on the same dataset (Lecun et al. 1995) showed an algorithm which achieves an error rate of 0.7%
Voted-perceptron at $T = 30$, $d = 4$ has error about 1.6%