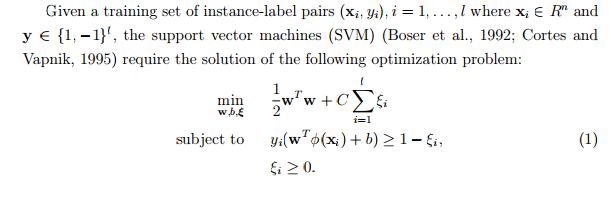
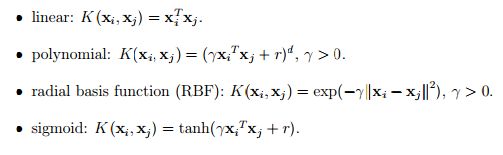
**5. Project Formulation and Setup**

Our problem deals with classification of images into one of several different classes. Large datasets are needed to train the classifier. This warrants the use of an efficient and robust multi-category classifier. For this project, we chose Support Vector Machines (SVM) and Neural Networks (NN) as both these classifiers are known to perform well for large multiclass classification. Both these models have been explained briefly below.

a.**SVM** : A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one “target value” (i.e. the class labels) and several “attributes” (i.e. the features or observed variables). The goal of the learning algorithm is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.



Here training vectors xi are mapped into a higher (maybe infinite) dimensional space by the function φ. SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. C > 0 is the penalty parameter of the error term. K(xi, xj ) ≡ φ(xi)T φ(xj ) is called the kernel function. The four basic kernels are:



SVM requires that each data instance is represented as a vector of real numbers. Hence, if there are categorical attributes, we first have to convert them into numeric data. Scaling before applying SVM is very important. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We thus linearly scaled each attribute to the range [0 , 1].

In general, the RBF kernel is a reasonable first choice. This kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear. Furthermore, the linear kernel is a special case of RBF. since the linear kernel with a penalty parameter C˜ has the same performance as the RBF kernel with some parameters (C, γ).

The SVM algorithms were implemented using LIBSVM is an integrated software for support vector classification, (C-SVC, [nu-SVC](http://www.csie.ntu.edu.tw/~cjlin/libsvm/#nuandone)) which supports multiclass classification.

A pictorial representation of the flow of the algorithm used to obtain the model and classify test images is shown below. This is an implementation of an already existing approach; basically training the classifier using optimal parameters obtained through some sort of validation procedure and then testing this formed model on unseen data.

Obtain Optimal C and gamma values

Train SVM in one-vs-rest (OVR) mode

Classify Test Samples using OVR model

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly.

These parameters had to be fine-tuned to get the optimal values for classification. This was done using cross-validation and will be explained in the subsequent section.

b.**Neural Networks**:

**(Preetham)**

**6. Methodology**

This section provides a brief overview of the entire procedure, beginning right from hypothesis and class label selection, all the way to performance measures and error analysis.

A flowchart is provided to visualize the framework. Each stage in this flowchart represents a class (or) group of operations.

Data Acquisition

-Hypothesis set

-Class label selection

-Dataset creation

Error Analysis

-Theoretical Formulation

Performance Measures

-Accuracy

-Confusion Matrix

Testing and Model Selection

-Tweak parameters

-Run Testing set

Training Process

-Obtain optimal parameters

-Train classifier

Cross-Validation

-n-fold

Feature Extraction

-Bag-of-features

Feature Reduction

- PCA

Pre-Processing

-Crop

-Image Inpainting

Procedure followed:

*Data Acquisition*

1. Hypothesis Set **(Preetham)**
2. Class labels selection **(Preetham)**
3. Dataset Creation **(Preetham)**

*Pre-Processing*

1. Crop : all the images are overlaid with text. This isn’t useful text and just behaves as noise in the image. Owing to the low quality of the image and the relatively huge size of the mask, inpainting algorithms too failed to provide us with a reasonable approximation. Thus, we adopted the option of cropping the images so that the part overlaid with text is cut and is no longer considered.
2. Inpaint : The patches at the bottom of the image which display the date, time and location are comparatively smaller than those at the top of the image. Thus, it was possible to apply the inpainting algorithm on them. This algorithm, which is based on a *weighted minimum norm-based* implementation will be discussed in the forthcoming section.
3. Feature Extraction : (Bag of Features) **(Preetham)**
4. Split into Training, Testing & Validation : **(Preetham)**
5. Feature Reduction : Since the feature matrix obtained for each of the sets is huge, it justifies considering some sort of feature reduction method to reduce the dimensionality of the data. The method we’ve used is Principal Component Analysis (PCA) and it’ll be summarized shortly.

*Training Process*

1. Cross validation : Cross validation is a model **validation** technique for assessing how the results of a statistical analysis will generalize to an independent data set.
2. Obtain optimal parameters : The cross validation step gives us the optimal value/s of the parameter/s needed for the model. These could be the penalty term (C) or gamma in case of SVMs or could be the number of hidden layers, number of nodes, etc.
3. Train Classifier : Once the optimal parameters are obtained, the classifier can be trained.

*Testing, Validation and model Selection*

1. Test on Validation Set : Once a model is created after learning the training data, the model is tested on the validation set. Even though we carried out This could be due to a local minima. The parameters of the model can be tweaked if the results of the validation aren’t satisfactory. These trial are repeated till we are satisfied with the model.
2. Obtain robust model : The continuous improvement of the model by repetitive trials of the validation data ensures that the final model we obtain is very robust and resilient to overfitting.
3. Test on Testing Set : The model is then tested on the testing set.
4. Performance Measures : Several performance measures such as accuracy, misclassification rate, confusion matrix, etc are used to judge the performance of the classifier on the testing data.

*Error Analysis*

1. Finally, check whether the out-of-sample error satisfies the expression provided for the error bound.

**7. Implementation**

This section explores more in-depth the steps mentioned in the previous section. The following steps are covered:

7.1. Feature Space:

**(Preetham)**

|  |  |  |  |
| --- | --- | --- | --- |
| Pipe Joint Without Defect | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\1 Pipe joint without defect\PJwD_0007.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\1 Pipe joint without defect\PJwD_2294.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\1 Pipe joint without defect\PJwD_2211.png |
| Pipe crack | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\4 Pipe crack\PipeCrack_0037.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\4 Pipe crack\PipeCrack_0234.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\4 Pipe crack\PipeCrack_1499.png |
| Side pipe zoomed | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\7 Side pipe zoomed\SPZ_0132.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\7 Side pipe zoomed\SPZ_0592.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\7 Side pipe zoomed\SPZ_0730.png |
| Pipe joint + side pipe | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\6 Pipe joint + side pipe\PJSP_0052.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\6 Pipe joint + side pipe\PJSP_0109.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\6 Pipe joint + side pipe\PJSP_0199.png |
| Manhole | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\9 Manhole\ManHole_0502.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\9 Manhole\ManHole_0757.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\9 Manhole\ManHole_1042.png |
| Debris | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\10 Debris\Debris_0022.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\10 Debris\Debris_0454.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\10 Debris\Debris_1663.png |
| Corrosion | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\11 Corrosion\Corrosion_0005.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\11 Corrosion\Corrosion_0830.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\11 Corrosion\Corrosion_0452.png |
| Calcinated | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\16 Calcinated\Calcinated_1418.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\16 Calcinated\Calcinated_1694.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\16 Calcinated\Calcinated_0053.png |
| Text images | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\17 Text images\Text images_0025.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\17 Text images\Text images_0471.png | C:\Users\Shravan\Google Drive\Shared with Dell\Team SewerPipe\Programs\Main\data\17 Text images\Text images_0225.png |

7.2. Pre-processing and Feature Extraction

Once the class labels were decided, then began the pre-processing part.

Each image underwent the following stages of pre-processing and feature extraction:

--Cropping the Image

All the images are overlaid with text. This isn’t useful text and just behaves as noise in the image. Owing to the low quality of the image and the relatively huge size of the mask, inpainting algorithms too failed to provide us with a reasonable approximation. Thus, we adopted the option of cropping the images so that the part overlaid with text is cut and is no longer considered.

--Image Inpainting

**Inpainting** is the process of [reconstructing](http://en.wikipedia.org/wiki/Image_reconstruction) lost or deteriorated parts of [images](http://en.wikipedia.org/wiki/Digital_image) and [videos](http://en.wikipedia.org/wiki/Digital_video). The notion of *digital inpainting*was firstintroduced in the paper by Bertalmio-Sapiro-Caselles-Ballester (SIGGRAPH 2000). Smart digital inpainting models, techniques, and algorithms have broad applications in image interpolation, photo restoration, zooming and super-resolution, primal-sketch based perceptual image compression and coding, and the error concealment of (wireless) image transmission, etc.

Inpainting is rooted in the restoration of images. The methodology followed is :

* The global picture determines how to fill in the gap. The purpose of inpainting is to restore the unity of the work.
* The structure of the gap surroundings is supposed to be continued into the gap. Contour lines that arrive at the gap boundary are prolonged into the gap.
* The different regions inside a gap, as defined by the contour lines, are filled with colors matching for those of its boundary.
* The small details are painted, i.e. “texture” is added.

The inpainting problem :

Given an image and a region Ω inside it, the inpainting problem consists in modifying the image values of the pixels in Ω so that this region does not stand out with respect to its surroundings.

Our Method : Our method of image inpainting was based off a weighted minimum norm implementation. Given the image, we generate a mask of the image indictating the locations at which text is present. The algorithm makes use of the mask to determine the region in which inpainting is to be done. Once this region has been identified, all the pixels from this region are made zero. This can be considered equivalent to removing that region from the image. The missing pixels have to be replaced with values in such a way that the l2-norm of the solution is the least. The image (solution) is multiplied with a weighting matrix and at every iteration, the goal is to choose an x that minimizes the l2-norm of the solution. Different constraints are then applied to the image. For example, the image does not

have continuous 2D edges, but instead has \jagged" edges in locations where the

text occluded an image edge. This type of information could have been restored if

we had used a 2D version of the WMN constraint that imposed both horizontal and

vertical smoothness. Alternatively, we could have used the prior information that

this type of painting should have a fairly consistent spatial texture, or the constraint

that the painting should be compressible in a wavelet transform, or the constraint

that the painting is possibly compressible under a low-rank matrix representation

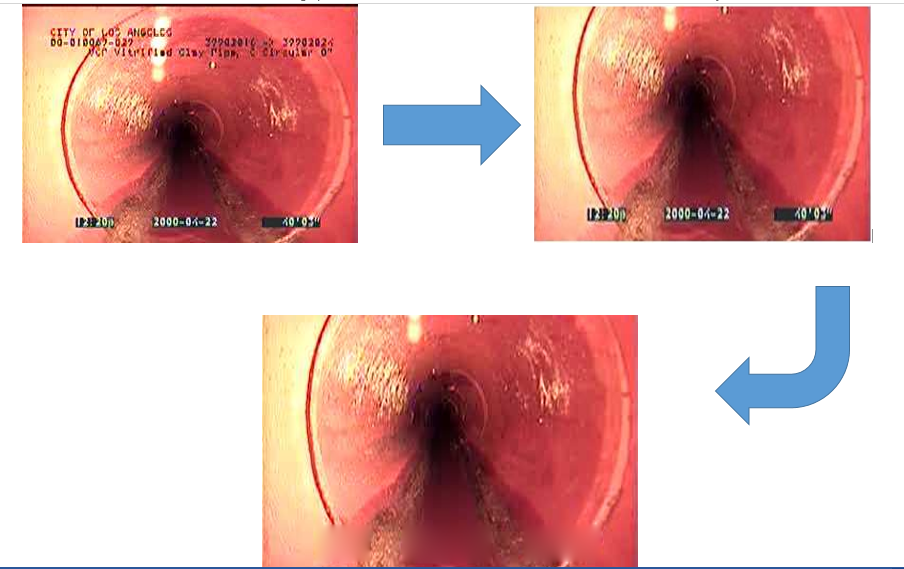
of the image.





Here, x\_WMN is the final solution we are interested in.

**Inpainting Results**:



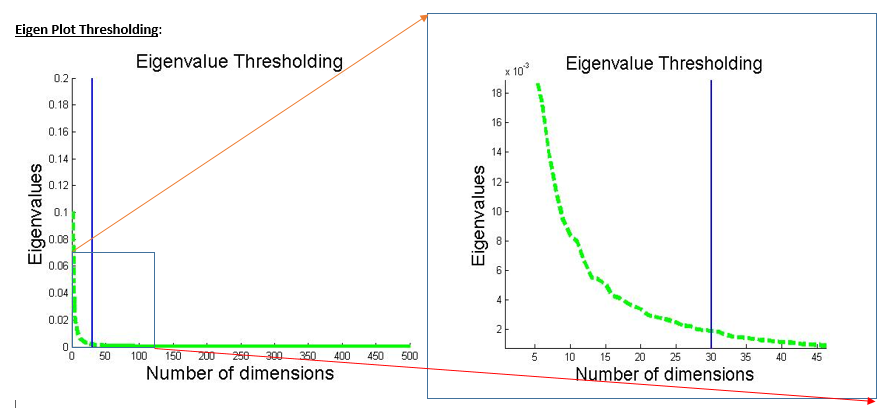
--Feature Extraction

**(Preetham)**

--Feature Reduction

Since the feature matrix obtained for each of the sets is huge, it justifies considering some sort of feature reduction method to reduce the dimensionality of the data. The method we’ve used is Principal Component Analysis (PCA). Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. PCA was invented in 1901 by *Karl Pearson*. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric. The principal components as a whole form an orthogonal basis for the space of the data. The first principal component is a single axis in space. When you project each observation on that axis, the resulting values form a new variable. And the variance of this variable is the maximum among all possible choices of the first axis. The second principal component is another axis in space, perpendicular to the first. Projecting the observations on this axis generates another new variable. The variance of this variable is the maximum among all possible choices of this second axis.

The choice of how many dimensions to reduce to is made through visual inspection of the plot of the eigenvalues, also called the ‘scree plot’. The plot for our Feature Matrix is provided below.



Original number of Dimensions = D = 500

Reduced number of Dimensions = D’ = 30.

7.3. Training Process

Once our cleaned and processed feature matrix is ready, we can begin the training process.

The first step is Cross Validation. **Cross-validation**, sometimes called **rotation estimation,** is a [model validation](http://en.wikipedia.org/wiki/Model_validation) technique for assessing how the results of a [statistical](http://en.wikipedia.org/wiki/Statistics) analysis will generalize to an independent data set. The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the *validation dataset*), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent data set (i.e., an unknown dataset, for instance from a real problem), etc. One round of cross validation involves

[partitioning](http://en.wikipedia.org/wiki/Partition_of_a_set) a [sample](http://en.wikipedia.org/wiki/Statistical_sample) of [data](http://en.wikipedia.org/wiki/Data) into [complementary](http://en.wikipedia.org/wiki/Complement_(set_theory)) subsets, performing the analysis on one subset (called the *training set*), and validating the analysis on the other subset (called the *validation set* or *testing set*). To reduce [variability](http://en.wikipedia.org/wiki/Variance), multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Cross-validation is important in guarding against [testing hypotheses suggested by the data](http://en.wikipedia.org/wiki/Testing_hypotheses_suggested_by_the_data) (called "[Type III errors](http://en.wikipedia.org/wiki/Type_III_error)) especially where further [samples](http://en.wikipedia.org/wiki/Statistical_sample) are hazardous, costly or impossible to collect.

1. Obtain optimal parameters : The cross validation step gives us the optimal value/s of the parameter/s needed for the model. These could be the penalty term (C) or gamma in case of SVMs or could be the number of hidden layers, number of nodes, etc.

**The optimal parameters obtained for the SVM with RBF Kernel are :**

**C = 2 and gamma = 16.**

**Complexity\_Timetaken**

1. Train Classifier in OVR (one-vs-rest mode) Once the optimal parameters are obtained, the classifier can be trained.

In [machine learning](http://en.wikipedia.org/wiki/Machine_learning), **multiclass** or **multinomial classification** is the problem of [classifying](http://en.wikipedia.org/wiki/Statistical_classification) instances into more than two classes. While some classification algorithms naturally permit the use of more than two classes, others are by nature [binary](http://en.wikipedia.org/wiki/Binary_classification) algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

**One-vs.-rest**

The *one-vs.-rest* (or *one-vs.-all*, OvA or OvR) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

### One-vs.-one

In the *one-vs.-one* (OvO) reduction, one trains *K* (*K* − 1) / 2 binary classifiers for a *K*-way multiclass problem; each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes. At prediction time, a voting scheme is applied: all *K* (*K* − 1) / 2 classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier.[[1]](http://en.wikipedia.org/wiki/Multiclass_classification#cite_note-bishop-1):339

Like OvR, OvO suffers from ambiguities in that some regions of its input space may receive the same number of votes.

**Number of training samples = 8,376.**

**Original Dimension (D) = 500.**

**New Dimension (D’) = 30.**

The model is tested repeatedly on the validation set until suitable parameters are obtained. This reduces the extent of overfitting or underfitting.

7.4. Testing, Validation and Model Selection

1. Test on Validation Set : Once a model is created after learning the training data, the model is tested on the validation set. Even though we carried out This could be due to a local minima. The parameters of the model can be tweaked if the results of the validation aren’t satisfactory. These trial are repeated till we are satisfied with the model.
2. Obtain robust model : The continuous improvement of the model by repetitive trials of the validation data ensures that the final model we obtain is very robust and resilient to overfitting.
3. Test on Testing Set : The model is then tested on the testing set.
4. Performance Measures : Several performance measures such as accuracy, misclassification rate, confusion matrix, etc are used to judge the performance of the classifier on the testing data.

7.5 Error Analysis

In order to measure the predictive performance of a function f : X → Y, we use a loss function. A loss function l : Y × Y → R+ is a non-negative function that quantifies how bad the prediction f(x) is if the true label is y. In the classification case, binary or otherwise, a natural loss function is the 0-1 loss:  Given a loss function, we can define the risk of a function f : X → Y as its expected loss under the true underlying distribution:

. Note that the risk of any function f is not directly accessible to the learner who only sees the samples. But the samples can be used to calculate the empirical risk of f: . Minimizing the empirical risk over a fixed class F ⊆ YX of functions leads to a very important learning rule, namely empirical risk minimization (ERM):

. If we knew the distribution P then the best function from F would be:

. Without restricting ourselves to the class F, the best possible function to use is:

 The excess risk of ERM relative to  can be decomposed as:



By the definition of ERM, the difference  is non-positive. The difference  is also easy to deal with since it deals with a fixed function . Using Hoeffding’s inequality, we have, with probability at least 1 − δ,

 [1]-[tewari]

We know fn lies in F. So we can clearly bound

 --🡪 *Generalization Error Bound*

We need to control the following quantity:

. Now, we can appeal to a powerful idea known as symmetrization. The

basic idea is to replace with .

We obtain a bound in terms of the Rademacher Complexity: 

(Massart’s finite class lemma) :



Covering Numbers

Let (D, ρ) be any (pseudo-)metric space and fix a subset T ⊆ D. A set T 0 ⊆ D is said to be

an α-cover of T if 

In other words “balls” of radius α placed at elements of T 0 “cover” the set T entirely. The covering number (at scale α) of T is defined as the size of the smallest cover of T (at scale α).

A major result connecting Rademacher complexity with covering numbers is due to *Dudley’s entropy integral bound :* For any F consisting of real valued functions bounded by 1, we have

Real Valued Functions: Fat Shattering Dimension

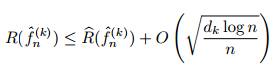
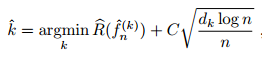
The appropriate notion for determining learnability using a finite number of samples turns out to be a scale sensitive combinatorial dimension known as the fat shattering dimension. Fix a class F consisting of bounded real valued functions f : X → [0, 1] and a scale α > 0. We say that a sequence  is α-shattered by F if there exists a witness sequence s1:n ∈ Rn such that, for every 1:n ∈ {±1} n, there is an f ∈ F such that . The fat-shattering dimension of F at scale α is the size of the longest sequence that can be α-shattered by F:

Significance: Using this bound, the sample complexity of learning a real valued function class of bounded range using a Lipschitz loss l is  where d = fatcε (F) for a universal constant c > 0. The importance of the fat shattering dimension lies in the fact that if ∞ for some α > 0, then no learning algorithm can have bounded sample complexity for learning the functions for arbitrarily small values of ε, δ.

Structural Risk Minimization: Consider the classification setting with 0-1 loss. Suppose we have a countable sequence of nested function classes:

 where  has dimension . The function  chosen by ERM over Fk

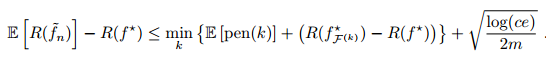
satisfies the bound:

 with high probability. As k increases, the classes become more complex. So, the empirical risk term will decrease with k whereas the VC dimension term will increase. The principle of structural risk minimization (SRM) chooses a value of k by minimizing the right hand side above. Namely, we choose: 

Assume that there are positive numbers c and m such that for each k we have the guarantee



Then the penalized estimator ˜fn with satisfies :



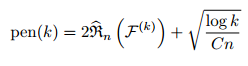
Skipping a few steps of calculation, we get, for any k,

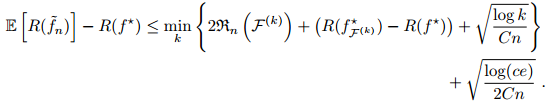


Data Driven Penalties Using Rademacher Averages

We know that there exist universal constants C, c such that 

Thus, we can use the data dependent penalty

 to get the bound:

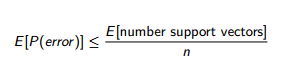


Specific Case: VC dimension of an SVM



*diameter* is the diameter of the smallest sphere that can enclose all the high-dimensional term vectors from the training set

*margin* is the smallest margin we’ll let the SVM use.



In our case, RBF kernels represent an infinite feature space and have infinite VC dimension.