|  |
| --- |
| **SVM Project and Gender Classification** |
| **Josh Gleason and Rod Pickens** |
| **Programming Assignment 4** |
| **Computer Science 679 – Pattern Recognition, UNR, Dr. Bebis** |
| **Due: Monday May 4, 2015** |
| **Submitted: Monday Monday 4, 2015** |

Contents

[1 Abstract 3](#_Toc417807382)

[2 Technical Discussion 3](#_Toc417807383)

[2.1 Gender classification 4](#_Toc417807384)

[2.1.1 Eigenvectors 4](#_Toc417807385)

[2.1.2 Eigenfaces 4](#_Toc417807386)

[2.2 Machine Learning 5](#_Toc417807387)

[2.2.1 Deductive Learning 7](#_Toc417807388)

[2.2.2 Inductive Learning 8](#_Toc417807389)

[2.3 Support Vector Machines 15](#_Toc417807390)

[2.4 Kernel Functions 19](#_Toc417807391)

[3 Project 22](#_Toc417807392)

[3.1 Experiment 1: SVM 22](#_Toc417807393)

[3.1.1 16x20 Images 22](#_Toc417807394)

[3.1.2 48x60 Images 22](#_Toc417807395)

[3.2 Experiment 2: Bayesian 22](#_Toc417807396)

[3.2.1 Training Parameters 22](#_Toc417807397)

[3.2.2 Testing on 16x20 Images 22](#_Toc417807398)

[3.2.3 Testing on 48x60 Images 22](#_Toc417807399)

[4 Conclusion 22](#_Toc417807400)

[4.1 Part A: SVM 22](#_Toc417807401)

[4.2 Part B: Bayesian 22](#_Toc417807402)

[4.3 Comparison 22](#_Toc417807403)

[5 Contributors 22](#_Toc417807404)

# Abstract

This paper describes our research and project for the fourth class project for the Computer Science pattern recognition class CS 679 taught by Dr. Bebis who is Department Chair of the Computer Science Department at the University of Nevada in Reno, Nevada.

The primary topics included in the main body of this report are a description of the project, learning machines, induction theory, theory of support vector machines (SVM), theory of kernel functions, and classifier results. The paper also compares SVMs with Bayes classifier.

We added to our project report a section that discusses the general theory of ***induction learning***, and we did this because we learned from our research on SVMs that SVMS correspond to a principle of induction learning known as structured risk minimization. In fact, as we understand from the lecture, structured risk minimization is contrasted with empirical risk minimization, and was we learned from research both are different induction principles.[[1]](#footnote-1)

It appears that machine learning is founded upon different induction principles of learning, and that each induction principle is implemented by one of many different estimation methods, known in the literature as learning methods. We discuss what we have learned in a tutorial section below both induction principles and learning methods.

# Technical Discussion

In this project, we developed both an SVM classifier and a Bayesian classifier to perform gender classification using as input features eigenfaces. With the SVM classifier, the features are mapped non-linearly to a higher dimensional feature space where a hyper-plane is used as the class discriminant. With the Bayesian classifier, the features are used in a standard normalized quadratic classifier.

In a prior class project, we studied eigenfaces and linear classification theory for the purpose of recognizing and detecting faces. To review that project, we discuss the theory of eigenfaces in Section 2.1.

The theory for support vector machines is based on the inductive principle of structured risk minimization, and support vector machines are a machine learning method associated with structured risk minimization. The discussion of inductive methods begins in Section 2.3 and concludes with the theory of support vector machines.

## Gender classification

### Eigenvectors

Eigenvectors are those vectors that are invariant in direction to the action of a matrix A on the vector as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

In the above, is a square nxn matrix, is an nx1 vector, and is a scalar. As mentioned above, the action of the matrix upon the vector is to scale, and to only scale, the vector and not to change the direction of the vector

### Eigenfaces

The eigenface approach taken and experimented in this paper is from the research of Turk and Pentland,[[2]](#footnote-2) and their work was motivated by the earlier works of Sirovich and Kirby who represented pictures using principal component analysis.[[3]](#footnote-3) Principal component analysis is a least squares approach for minimizing the error associated with a projection of the data onto a different basis.

Given a vector in an N dimensional space, it can be represented by a set of N orthogonal basis vectors as follows

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

The goal is to find an N x K transformation matrix U such that

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Where is an Nx1 rasterized image

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

And the basis vectors the right side of the equation are the standard basis (natural or canonical) vectors of Euclidean space as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Where is known as the Kronecker delta function

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Yielding a set of basis vectors as follows

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Moreover, the vector can be represented by a set of K orthogonal basis vectors *i*=1,…,K in a lower K (K<N) dimensional space, and each vector is an Nx1 vector. The lower K, again where K<N, dimensional space is defined as

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Principal component analysis selects the basis and coefficients to minimize the error

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

## Machine Learning

A learning machine is used to produce rules for a decision making machine, and once the decision making machine is trained with the rules, the decision making will be able to receives inputs and makes decisions. For example, the machine could receive as inputs weather information, for example, and make a decision about whether it will rain or not.

The learning machine learns from pairs of data {x, y} empirically measured from some aspect of the world, and creates a mathematical model f along operational parameters w\* that combined express the rules of the decision making machine. Once learning is complete, the model f receives as input values of x and produces as output a decision y’ which is the best approximation to what the truth y will be.

Cherkassky[[4]](#footnote-4) describes a learning machine as a method that learns from a pair of training values x and y where the inputs x are produced by a generator module and the y are produced by the system module. The learning machine then uses the inputs to approximate the true value of y given new samples x when the true value of y is unknown, Figure 1. Once the learning machine has completed learning, the rules f and parameters w\* developed by the learning machine are handed to a decision making machine, also in Figure 1.



Figure : Learning Machine (Cherkassky[[5]](#footnote-5))

There are two aspects to the learning machine in the Figure 1: training and testing. During training, the learning machine received inputs x and y and produces a model of as a function f(x,y,w\*) which best approximates the output of the system y when an unknown x arrives.

Testing of the learning machine occurs when only a single input x is input into the learning machine, and the learning machine predicts y (y’) based on x. The prediction y’ is compared with the true value of y, and the error rates between predicted and true are accumulated. The learning continues until the error between predicted y’ and true are minimized to an acceptable level. The performance of the learning is based on the results of the testing. Once learning is complete, the models f and w\* are handed to the decision making machine.

We now start describing learning mathematically. During training the learning machine builds a function f(x,y,w\*) that approximates the output from the system function h(x,w0) given as input data from the generator function g(x,w0). The learning machine f(x,y,w\*) estimates a function f(x,w\*) from training data x and y with the purpose of using f(x,w\*) in a decision making machine to predict h(x,w0) when only x valuables are available:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

The input x and y to the learning machine are independent and identically distributed (i.i.d.) pairs of data (*x,y*) distributed according to an unknown probability distribution function

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

If the test samples have the same probability distribution as the training samples, then the learning machine should correctly label unknown input samples x with the appropriate label y.

The best function[[6]](#footnote-6) for learning should be that function that minimizes the expected risk (error)

|  |  |  |
| --- | --- | --- |
|  |  | (12) |
|  |  |  |
|  |  |  |

The function is the loss function and P(x,y) is the known joint probability density function of the input x and y. Assuming a 0/1 loss function, then the expected risk is the minimum error function.

### Deductive Learning

In deductive learning, there is no need to build a learning machine; rather, a decision machine can be built directly from a known model f and parameters w that exactly specify the generator. It turns out that the model f(x,w) is identical to the generator g(x,w), and consequently the decision making machine is the best model available to predict values y. This is possible because the generator operates according to completely understood theory g(x,w), and no new information h(x) from the generator can invalidate the theoretical model g(x,w) of the decision machine.



Figure : Decision machine based on deductive learning

In other words, a machine that makes decisions based on deduction is trained from a theoretically valid model f and w in which the desired output y of the generator is completely known.

Under deductive learning, the researcher does not need to build a learning machine to empirically learn a model of the world; rather, the models f and w are supplied by the researcher to the decision making machine. With this, a researcher completes the design of the decision making machine by selecting a loss function such that the machine operates by making decisions that minimize the expected risk as follows.

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

### Inductive Learning

Inductive reasoning is the process of discovering general concepts from observing a limited set of examples, e.g. the training discussed above. The goal of inductive reasoning is decide upon an approach for encapsulating general knowledge in a model f(x,w\*) of some aspect of the world. The model is used to take information x and produce a decision y=f(x,w\*). The learning is performed by measuring specific information x and y from some aspect of the world, and the data to create a model f(x,w\*) that predicts the state of the world y given new inputs x.

Because the inductive reasoning model evolves as new data x is made available for learning, a given inductive model can be invalidated by new information. In other words, the model can fail to predict the state of the world when new examples from the world are presented to the model.

When the conditions for ideal learning are unavailable, e.g. the probability distribution is unknown, then a researcher will be required to select from one of a number of inductive learning principles with which to base his machine learning engine. Once an inductive principle is selected, then the researcher will have to select a particular learning method from that inductive principle to construct the learning machine.

To illustrate with an example, assume that the researcher has no a-priori knowledge of the probability distribution for x and y. Given this, the researcher can proceed in the design of the learning machine basing on using a simple induction principle. A particularly simple induction principle is use is to base the design upon an approximation to the expected risk. Defining the loss function as the 0/1 or symmetric loss, the empirical risk can be defined as the sample mean of the loss associated with the outcomes of the experiment as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

Note that the empirical risk should asymptotically approach the expected risk.

According to Cherkassky,[[7]](#footnote-7) there are a number of inductive principles used in machine learning:

1. Empirical Risk Minimization
2. Structural Risk Minimization
3. Penalization
4. Bayesian Inference
5. Minimum Description Length

The author primarily focuses on relating the four inductive principles of Penalization, Bayesian Inference, Structured Risk Minimization, and Minimum Description Length, but he gives a short description of empirical risk minimization (to be discussed later), see Figure 3.



Figure : Inductive Principles in Learning

Cherkassky repeatedly highlights that there is an important distinction between inductive principles and learning methods. For a given inductive principle, many different learning methods (even infinitely many) can be used to execute the inductive principle. These different learning methods correspond to different classes of approximation functions (the decision rule g(x)), parameters, and the optimization technique (how the g(x) is found). The learning methods are also known as constructive implementations of an induction principle as shown in Figure 4.



Figure : Learning methods for a given inductive principle

An inductive method is concerned with a general approach to how to use data for learning, and the learning method that one uses from a particular inductive method is the implementation that gives the estimate, or what to do with the data.

Cherkassky segregates inductive principles into classical and adaptive, and he distinguishes the two inductive principles based on the amount of information available prior to learning. In other words, the two classes are distinguished by the amount of information assumed as valid prior to learning, Figure 3.

The classical inductive methods (e.g. empirical risk minimization) assume strong a-priori knowledge of the probability distributions whereas the adaptive inductive methods assume weak a-priori knowledge of the probability distributions.

To illustrate the above concepts, the author Cherkassky illustrates empirical risk minimization (ERM) along with the various learning methods that can be implemented under the rubic of ERM such as maximum likelihood, linear regression, polynomial regression, and fixed-topology neural networks.

In performing ERM such as maximum likelihood, the model is given, e.g. Gaussian density function, and then the parameters for that Gaussian distribution are found, e.g. the mean and standard deviation, using a method such as maximum likelihood.

ERM works well when the number of training samples is large relative to the model complexity (number of free parameters such as mean and standard deviation). For example, if only one free parameter is required to be estimated, e.g. the sample mean, and if many samples are available for estimating the sample mean, then maximum likelihood is a good estimator for the mean (assuming that assumed probability density model is the correct density model).

If however, the number of free parameters (mean, covariance matrix) that must be estimated is large relative to the number of training samples, then the ERM methods performs poorly. Other factors can degrade the quality of the estimate from the learning machine. Suppose the learning machine is to learn the mean, then it could be the case that the mean is an insufficient statistic for the statistic (not well estimated by the sample mean) and will not work within the model.

#### Empirical Risk Minimization

Empirical risk minimization is the inductive approach associated with the classical statistical approaches of “parametric estimation” and “classification” as developed by R.A. Fisher (1890-1962).[[8]](#footnote-8)

R.A. Fisher was trained as a mathematician, and as a student, he had not formally studied statistics. He became involved in statistics during his first job with the city of London and in a later job working at the Rothamsted Experimental Station (one of the oldest agriculture research institutes in the world) in Harpenden, Herfordshire, England. He started working at the experimental station in the year 1919, and while at the station he performed agriculture crop research. During this time, Fisher used and developed modern statistical concepts to support his analysis in crop variation. His research also supported his personal passion for studies in eugenics.

During this time period while working at the experimental station, Fisher became especially involved in both statistical analysis and the design of experiments, and this was sparked by his reading of William Gosset’s paper on the Student’s t- finite sampling distribution. Later, Fisher became interested in curve fitting after reading Karl Pearson’s method of moments for curve fitting, which Fisher questioned as an appropriate method.

Fisher then recast Pearson’s curve fitting approach into the method of maximum likelihood. Fisher published his new approach which evidently created friction between himself and Pearson. During Fisher’s lifetime, he nearly single-handedly created modern empirical statistical theory, which falls under the inductive method of empirical risk minimization.[[9]](#footnote-9)

Under the rubic of empirical risk minimization, we learn that maximum likelihood is a specific “constructive method” or “learning method” used to estimate the parameters of a distribution given the researcher can correctly assume the proper probability density.[[10]](#footnote-10) Once a probability distribution for the data is assumed, the parameters are estimated based on “empirically” collected data (empirical in that the data is collected from an experiment).

As an interesting side note, we have learned that the maximum likelihood method can be associated minimizing the Kullback-Leiber divergence metric (as used in information theory).[[11]](#footnote-11) In other words, the Kullback-Leiber divergence metric is one of many learning machines that is associated with empirical risk minimization induction principle.

#### Structural Risk Minimization

Structural risk minimization was first presented in a paper by Vladimir Vapnik and Alexey Chervonenkis[[12]](#footnote-12) in 1974 as a method of optimal learning that combines “empirical risk minimization” with an added constraint that minimizes the complexity of the model (that is used to make decisions.[[13]](#footnote-13) In essence, the structural risk principle favors learning machines that have less complex models (e.g. the machine is based on low order polynomials rather than high order polynomials).

The author’s formulated a theory of statistical learning with the goal of generalizing the learning, and with guided by this goal the author’s determined to control both the empirical risk and the generalization ability of a learning machine by two factors: error-rate on training data and the capacity or confidence of the learning machine[[14]](#footnote-14):

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

The confidence is related to the Vapnik-Chervonenkis (VC) dimension (h) of the decision function class (how complex the decision function is) and the number of training samples (n) as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

The confidence interval holds with probability 1-, and note that for this to hold n > h.

The general intuition behind the statistical learning theory is that a researcher can design a learning machine with a high complexity decision boundary (high VC dimension) such that the empirical training error goes to zero. This is generally not desirable. In the case where the training errors go to zero, it is often the case that the confidence could be quite large while the learning machine predicts poorly; in other words, the classifier error >> expected risk.

At the other extreme, the researcher could lower the confidence interval to nearly zero at the risk of increasing the empirical error (too much generalization) which gives a learning machine poor performance; in other words, the error >> expected risk.[[15]](#footnote-15)

Vapnik and Chervnonkis recommend designing the learning machine so as to find a balance between confidence and empirical risk.

From the statistical learning theory developed by Vapnik and Chervnonkis, the induction principle of structural risk minimization was formulated and leads to a class of learning machines known as support vector machines (SVMs) which are linear classifiers. SVMs are often combined with non-linear functions that map the inputs x into a higher dimensional feature space to improve classification. In this higher dimensional space , the learning machine f(x,w) can build a low complexity decision boundary, such as a hyper-plane, such that the classifier error is minimized and the generalization ability of the classifier is maximized.[[16]](#footnote-16)

#### Penalization Induction Principle

The risk[[17]](#footnote-17) used in the inductive principle of penalization is determined as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

The empirical risk denotes training error, and the penalty function is a non-negative functional associated with each possible estimate of f(x,w). The parameter weights the penalty relative to the empirical risk in decision making. The genetic algorithm discussed in class appears to be a machine learning method associated with penalized induction.

#### Minimum Description Length Induction Principle

This principle[[18]](#footnote-18) contrasts with all other methods in that the approach regards models as codes, as in information theory. The approach is to encode the training data such that the code length represents the data in a way that generalizes the learning machine.

#### Bayes Induction Principle

Bayes induction principle[[19]](#footnote-19) is based on having available a-priori probability information along with a-priori defined likelihood functions. Given the a-priori probabilities, a classifier is defined that reduces empirical risk. Assuming that the loss function is the 0/1 loss function, then the Bayesian induction produces a learning machine based on Bayes’ principle

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

With the 0/1 loss function, the Bayes classifier minimizes the expected risk and error. The Bayes decision rule then becomes defined as

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

As an interesting side note, Cherkassky states that the penalization inductive method has been formulated in terms of Bayes induction by some researchers given that the penalty and the priors can be considered similar, but Cherkassky concludes otherwise because the penalty is not based on a prior probability distribution.[[20]](#footnote-20)

## Support Vector Machines

The support vector machine (SVM) is a learning machine (or constructive method) that belongs to the structural risk minimization inductive principle, as described above. The SVM is primarily a two-class classifier that can be extended to multiple-classes. The optimization criterion is to maximize the margin of separation between the features of the classes which ensures that the VC dimension is minimized along with the empirical risk. Training a SVM is equivalent to solving a quadratic programming problem with linear constraints.[[21]](#footnote-21)

To describe the SVM, the inventors assume classification without training error (that which minimizes the empirical error) because they assume complete class separability. Given this assumption, the following linear functional defines a hyper-plane between classes that operates with no error (assuming the classes are separable):

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

Where and is a feature vector. The classifier derived from the above functional is simply

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

The learning machine defines both learning the vector w (the vector perpendicular to the hyperplane separating the classes) and the parameter b (the position of the hyperplane). With properly defined w and b, a support vector learning machine maximizes the distance between the nearest neighbors (support vectors) of the classes with the decision boundary. The boundary of the decision surface, the hyperplane, is given as

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

And as shown in Figure 4



Figure : Hyperplane wx + b (defined by vector w) and margin

The distance between the classes as defined by the boundary can be determine from two exemplars and that lie on the boundaries: and . The distance between the lines is determine as

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

The maximization of the distance across the margin, which in turn reduces the VP dimension, is done by quadratic programming to minimize as follows

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

We are unable to solve this minimization directly, so we use Langrangian multipliers to find w and b, as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

The solution is rather straight forward, but prior to solving this optimization, we introduce kernel functions to map the feature vector x non-linearly to a higher dimensional space to ensure class separability (as assumed above). The kernel function comes to us via a non-linear function that directly maps x into a higher (possibly much higher) dimensional feature space in which the classes become linearly separable. The function is defined as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

Given this function, one develops the learning algorithm in the space rather than original space with training samples now defined as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (27) |

where . We now substitute for x into the above Langrangian

|  |  |  |
| --- | --- | --- |
|  |  | (28) |

With this and to optimize the learning machine, we maximize with respect to w and b, and then we minimize with respect to the Langrangian multipliers .[[22]](#footnote-22)

Taking the partials of the Langrangian and setting to zero so that we can find the extremes, we have

|  |  |  |
| --- | --- | --- |
|  |  | (29) |

And

|  |  |  |
| --- | --- | --- |
|  |  | (30) |

Solving the first partial gives us

|  |  |  |
| --- | --- | --- |
|  |  | (31) |

Which reduces to

|  |  |  |
| --- | --- | --- |
|  |  | (32) |

Solving the second partial gives us

|  |  |  |
| --- | --- | --- |
|  |  | (33) |

Which reduces to

|  |  |  |
| --- | --- | --- |
|  |  | (34) |

The latter is an amazing result because it gives us convolution. We believe this because the optimization produces a learning machine that involves producing the outcome from a sum of dot products, which in turn is a convolution of the feature vectors in the higher dimensional space.

Substituting w into the Langrangian and maximizing based on the Langrangian multipliers we get the following quadratic optimization problem

|  |  |  |
| --- | --- | --- |
|  |  | (35) |

Substituting a kernel function for the non-linear mapping functions , we get the quadratic optimization problem:

|  |  |  |
| --- | --- | --- |
|  |  | (36) |

With the following linear constraints

|  |  |  |
| --- | --- | --- |
|  |  | (37) |

And

|  |  |  |
| --- | --- | --- |
|  |  | (38) |

In matrix notation, the above simplifies to[[23]](#footnote-23)

|  |  |  |
| --- | --- | --- |
|  |  | (39) |

This clearly shows the fact that the optimization is performed on a quadratic surface with the following constraints

|  |  |  |
| --- | --- | --- |
|  |  | (40) |

The individual entries in the matrix are defined as

## Kernel Functions

In the treatment of SVMs above, we derived the learning machine using kernel functions rather than for x alone for the purpose of expediting the discussion. We did this because the solutions to the SVM give are identical to solving x alone. We now discuss using Kernel functions to augment the non-linear SVM solution. Recall that we mapped the input features into a high dimensional feature space using a non-linear mapping such that the input features become linearly separable in the space The non-linear mapping is defined as

|  |  |  |
| --- | --- | --- |
|  |  | (41) |

The mapping produces a feature space such that the classes are separable and can be learned by a linear SVM where the linear SVM is defined as[[24]](#footnote-24)

|  |  |  |
| --- | --- | --- |
|  |  | (42) |

The non-linear mapping is substituted into the linear SVM in place of x in the above equation as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (43) |

The decision boundary is still a linear surface (a hyper-plane), and the hyper-plane is defined in the feature space . Assuming the samples x in the feature space are separable, then a linear classifier that minimizes the empirical risk and the VD dimension will give good generalization results. Unfortunately, the heavy computational burden of the dot product when given very large input feature vectors x and mapping to much larger dimension feature spaces limits the applicability of the non-linear mapping, without modification. It so happens that dot products in the feature space can be computed with a function known as the kernel K which gives identically results as but in the lower dimensional space of the input features.

The kernel function K can be used as a substitute for the feature space dot products as long as the function K satisfies the following equality

|  |  |  |
| --- | --- | --- |
|  |  | (44) |

The function K must meet Mercer’s theorem to be a substitute. When applicable, the substitution of for x produces a non-linear mapping that can be input into the following learning decision rule:

|  |  |  |
| --- | --- | --- |
|  |  | (45) |

To conclude this section, the authors copy the “kernel\_method” web page as discussed on the Wikipedia because this article gives a very interesting treatment of the kernel function as used in machine learning. Quoting the Wikipedia directly[[25]](#footnote-25):

“Kernel methods can be thought of as [instance-based learners](http://en.wikipedia.org/wiki/Instance-based_learning): rather than learning some fixed set of parameters corresponding to the features of their inputs, they instead "remember" the i-th training example (\mathbf{x}_i, y_i)by learning a corresponding weight w_i. Prediction for unlabeled inputs, i.e., those not in the training set, is treated by the application of a [similarity function](http://en.wikipedia.org/wiki/Similarity_function) k, called a **kernel**, between the unlabeled input \mathbf{x'} and each of the training inputs \mathbf{x}_i. For instance, a kernelized [binary classifier](http://en.wikipedia.org/wiki/Binary_classifier) typically computes a weighted sum of similarities

\hat{y} = \sgn \sum_{i=1}^n w_i y_i k(\mathbf{x}_i, \mathbf{x'}),

where

* \hat{y} \in \{-1, +1\} is the kernelized binary classifier's predicted label for the unlabeled input \mathbf{x'} whose hidden true label y is of interest;
* k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R} is the kernel function that measures similarity between any pair of inputs \mathbf{x}, \mathbf{x'} \in \mathcal{X};
* the sum ranges over the *n* labeled examples \{(\mathbf{x}_i, y_i)\}_{i=1}^n in the classifier's training set, with y_i \in \{-1, +1\};
* the w_i \in \mathbb{R} are the weights for the training examples, as determined by the learning algorithm;
* the [sign function](http://en.wikipedia.org/wiki/Sign_function) \sgn determines whether the predicted classification \hat{y} comes out positive or negative.

Kernel classifiers were described as early as the 1960s, with the invention of the [kernel perceptron](http://en.wikipedia.org/wiki/Kernel_perceptron).[[1]](http://en.wikipedia.org/wiki/Kernel_method#cite_note-1) They rose to great prominence with the popularity of the [support vector machine](http://en.wikipedia.org/wiki/Support_vector_machine) (SVM) in the 1990s, when the SVM was found to be competitive with [neural networks](http://en.wikipedia.org/wiki/Artificial_neural_network) on tasks such as [handwriting recognition](http://en.wikipedia.org/wiki/Handwriting_recognition).

# Project

This project consists of several experiments to compare various versions of the eigenface recognition algorithm on different data sets.

Prior to showing results for the class project, we studied the decision surfaces as we varied the cost parameter, and we gained new intuition to the SVM. We begin showing the results from these learning experiments.

## Learning Experiments

### Polynomial Kernel

#### Degree 1

Figure 6 shows three different experiments with the SVM using a polynomial kernel. The red dots correspond to one class and the blue dots correspond to a second class, and the decision boundary is a line surrounded on one side by a red region and on the other side by a blue region. Each point is a support vector, and the decision boundary is linear in all cases, as expected from a polynomial of degree one.

|  |  |  |
| --- | --- | --- |
| a: Cost 0.2 | b: Cost 0.5 | C: Cost 1000 |

Figure : Polynomial kernel with different costs

From these experiments we can readily observe the impact of the cost. As expected, as the cost becomes larger, the margin increases. In the first case, the cost is 0.2 which is small. The SVM attempts to minimize slack errors with priority over the margin, and we see that the margin grows to the right to minimize slack error at the cost of classifier error. As the cost increases to 0.5, the slack error and the margin are both weighted such that a decision boundary is produced that correctly classifies all points, but the margin is not maximized nor is slack error minimized. As the cost increases to 1000, the slack errors are weighted less than the margin, and the margin approaches that expected to maximize the separation between classes.

#### Degree 3

Figure 7 shows an SVM using a polynomial kernel of degree 3, and the curvature of the boundary seems appropriate of a polynomial of third degree. Each point is a support vector.



Figure : Polynomial of degree 3

### Radial Basis Function Kernel

The radial basis function is a based on Gaussian kernel, and we show two cases of using a Gaussian kernel with different standard deviations. In Figure 8a, with the smaller standard deviation σ = 0.10, we see that the classifier decision boundary is nearly piece-wise linear. In Figure 8b, with the larger standard deviation σ = 0.50, we see that the classifier decision boundary is smoothed. Each point is a support vector.

|  |  |
| --- | --- |
| a: σ = 0.10 | b: σ = 0.50 |

Figure : RBF with different σ

## Experiment 1: SVM

### 16x20 Images

#### Polynomial Kernels

|  |  |
| --- | --- |
|  |  |
|  |  |

#### RBF Kernels

|  |  |
| --- | --- |
|  |  |
|  |  |

### 48x60 Images

#### Polynomial Kernels

|  |  |
| --- | --- |
|  |  |
|  |  |

#### RBF Kernels

|  |  |
| --- | --- |
|  |  |
|  |  |

## Experiment 2: Bayesian

### Training Parameters

### Testing on 16x20 Images

### Testing on 48x60 Images

# Conclusion

## Part A: SVM

## Part B: Bayesian

## Comparison

# Contributors

1. Cherkassky, Vladimir, “Inductive Principles for Learning from Data,” http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.7820&rep=rep1&type=pdf [↑](#footnote-ref-1)
2. M. Turk and A. Pentland, “Eigenfaces for Recognition,” Journal of Cognitive Neuroscience, vol. 3, no. 1 pp. 71-86, 1991. [↑](#footnote-ref-2)
3. L Sirovich and M Kirby, “Low-dimensional procedure for the characterization of human faces,” Journal of the Optical Society of America A, 4(3), 519-524. [↑](#footnote-ref-3)
4. Cherkassky, Vladimir, “Inductive Principles for Learning from Data,”page 2. [↑](#footnote-ref-4)
5. Cherkassky, Vladimir, “Inductive Principles for Learning from Data,”page 2. [↑](#footnote-ref-5)
6. Klaus-Robert Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernard Scholkoph, “An Introduction to Kernel-Based Learning Algorithms,” IEEE Transactions on Neural Networkds, Vol. 12, No. 2, March 2001. [↑](#footnote-ref-6)
7. Ibid, page 5. (Cherkassky, Vladimir, “Inductive Principles for Learning from Data”) [↑](#footnote-ref-7)
8. http://en.wikipedia.org/wiki/Ronald\_Fisher [↑](#footnote-ref-8)
9. Spanos, Arias, “R. A. Fisher: how an outsider revolutionized statistics,” http://errorstatistics.com/2014/02/17/r-a-fisher-how-an-outsider-revolutionized-statistics-2/ [↑](#footnote-ref-9)
10. Ibid, page 8. (Cherkassky, Vladimir, “Inductive Principles for Learning from Data” ) [↑](#footnote-ref-10)
11. Kevin Swersky, “Inductive Principles for Learning Restricted Boltzmann Machines,” Master’s Thesis, University of British Columbian, August 2010. [↑](#footnote-ref-11)
12. http://en.wikipedia.org/wiki/Structural\_risk\_minimization [↑](#footnote-ref-12)
13. Bebis, George, Lecture on “Support Vector Machines,” page 7, 2015. http://www.cse.unr.edu/~bebis/CS479/ [↑](#footnote-ref-13)
14. Corinna Cortes and Vladimir Vapnik, “Support-Vector Networks,” Machine Learning, 20, pages 273-297, 1995. [↑](#footnote-ref-14)
15. Klaus-Robert Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernard Scholkoph, “An Introduction to Kernel-Based Learning Algorithms,” IEEE Transactions on Neural Networkds, Vol. 12, No. 2, March 2001. [↑](#footnote-ref-15)
16. Vladimir Vapnik, “The Nature of Statistical Leaning Theory,” Springer-Verlag, New York, 1995. [↑](#footnote-ref-16)
17. Cherkassky, Vladimir, “Inductive Principles for Learning from Data,”page 8. [↑](#footnote-ref-17)
18. Ibid, page 13. (Cherkassky, Vladimir, “Inductive Principles for Learning from Data.”) [↑](#footnote-ref-18)
19. Ibid, page 11. (Cherkassky, Vladimir, “Inductive Principles for Learning from Data.”) [↑](#footnote-ref-19)
20. Ibid, page 13. (Cherkassky, Vladimir, “Inductive Principles for Learning from Data”) [↑](#footnote-ref-20)
21. Bebis, George, Lecture on “Support Vector Machines,” page 7, 2015. http://www.cse.unr.edu/~bebis/CS479/ [↑](#footnote-ref-21)
22. Klaus-Robert Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernard Scholkoph, “An Introduction to Kernel-Based Learning Algorithms,” IEEE Transactions on Neural Networkds, Vol. 12, No. 2, March 2001. [↑](#footnote-ref-22)
23. Corinna Cortes and Vladimir Vapnik, “Support-Vector Networks,” Machine Learning, 20, pages 273-297, 1995. [↑](#footnote-ref-23)
24. Bebis, George, Lecture on “Support Vector Machines,” page 25, 2015. http://www.cse.unr.edu/~bebis/CS479/ [↑](#footnote-ref-24)
25. http://en.wikipedia.org/wiki/Kernel\_method [↑](#footnote-ref-25)