UCRL-TR-227939

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Multi-Layer Perceptrons and Support Vector Machines for Detection Problems with Low False Alarm Requirements: an Eight-Month Progress Report

Barry Chen, Tracy Hickling, Milovan Krnjajic, William Hanley, Grace Clark, John Nitao, David Knapp, Larry Hiller, Marshall Mugge

February 12, 2007

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Multi-Layer Perceptrons and Support Vector Machines for Detection Problems with Low False Alarm Requirements: an Eight-Month Progress Report

June 1, 2006

Barry Chen, Tracy Hickling, Milovan Krnjajić, Bill Hanley, Grace Clark, John Nitao, David Knapp, Larry Hiller, and Marshall Mugge

Table of Contents

1. Receiver Operating Characteristics (ROC) and Related Performance Metrics *(Status 02-09-2006)*

In this project, the basic problem is to automatically separate test samples into one of two categories: clean or corrupt. This type of classification problem is known as a two-class classification problem or detection problem. In what follows, we refer to clean examples as negative examples and corrupt examples as positive examples.

In a detection problem, a classifier decision on any one sample can be grouped into one of four decision categories: true negative, true positive, false negative and false positive. These four categories are illustrated by Table 1.

		Actual Category of Sample	
		Negative	Positive
Classifier's Decision	Negative	*True Negative*	*False Negative*
	Positive	*False Positive*	*True Positive*

Table 1 – Four decision categories of a classifier applied to a two-class problem.

True negatives and true positives are cases where the classifier has made the correct decision. False positives are cases where the classifier decides positive when the true nature of the sample was negative, and false negatives are cases where the classifier decides negative when the sample was actually positive. To evaluate the performance of a classifier, we run the classifier on all the samples of a data set and then count all the instances of true negatives, true positives, false negatives, and false positives. All of the performance metrics in this report are then formed from a combination of these four basic decision categories.

There are four performance rates of great significance in this report: true negative rate, true positive rate, false negative rate, and false positive rate. They are given by the following equations:

$$(1.1)\ \text{true negative rate} = \frac{\text{total number of true negatives}}{\text{total number of true negatives} + \text{total number of false positives}}$$

$$(1.2)\ \text{true positive rate} = \frac{\text{total number of true positives}}{\text{total number of true positives} + \text{total number of false negatives}}$$

$$(1.3)\ \text{false negative rate} = \frac{\text{total number of false negatives}}{\text{total number of true positives} + \text{total number of false negatives}}$$

$$(1.4)\ \text{false positive rate} = \frac{\text{total number of false positives}}{\text{total number of true negatives} + \text{total number of false positives}}$$

Note that false negative rate is also equal to 1 – true positive rate; likewise, false positive rate is equal to 1 – true negative rate.

The accuracy and precision of a classifier are given by:

$$(1.5)\ \text{accuracy} = \frac{\text{total number of true negatives and true positives}}{\text{total number of samples}}$$

$$(1.6)\ \text{precision} = \frac{\text{total number of true positives}}{\text{total number of true positives} + \text{total number of false positives}}$$

Accuracy measures the proportion of correct classifications with respect to all the data samples, and precision measures the proportion of correct classification with respect to all the data for which the classifier decides positive. Please note that unless otherwise specified all accuracy and precision numbers reported below come from operating the classifier at its default decision threshold (for MLPs, the threshold is 0.5, and for SVMs, the threshold is 0.0).

The classifier performance goal for this project is a high true positive rate with a false positive rate of less than 10^{-4}. Neither accuracy nor precision adequately measures the performance goals set by this project. To adequately measure how well a classifier is performing with respect to the performance goal, we will be calculating a metric based on the receiver operating characteristic (ROC) of the classifier. A ROC is simply a list (or lists) of false positive rates and corresponding true positive rates of a classifier (or a set of classifiers). Because most classifiers output a classification metric, e.g., a posterior probability of positive class, one can generate a ROC by varying the decision threshold on this classification metric and computing the true positive and false positive rates for each decision threshold. For example, a decision threshold of 0.5 means that test samples with posterior probabilities greater or equal to 0.5 will be classified as positive samples. Greater decision thresholds will result in fewer true positives and false positives, while smaller decision thresholds lead to more true positives and false positives. ROCs are usually visualized graphically with an X-Y plot where the x-axis corresponds to the false positive rate and the y-axis corresponds to the true positive rate. Figure 1 is a plot of example ROC curves from a SVM tested on data from five different signal levels.

To measure the degree to which a classifier achieves the performance goal of high true positive rates at low false positive rates, we compute the area under the ROC curve in the region of low false positive rates – 0 to 10^{-3} false positive rate. The standard area under the ROC curve (AUROC), measures the average true positive rate of a classifier over the entire range of false positive rates. It is equivalent to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example [6]. We have adapted the AUROC so that we compute the area under the ROC curve only over the region of interest for this project, i.e., low false positive rates between 0 and 10^{-3}. We refer to this adapted metric as AUROC10^{-3}. In what follows, we will report normalized AUROC10^{-3} which is the area under the ROC curve between false positive rates 0 and 10^{-3} divided by 10^{-3}. We have written a C program called ROCMain that computes ROC curves and AUROC10^{-3}.

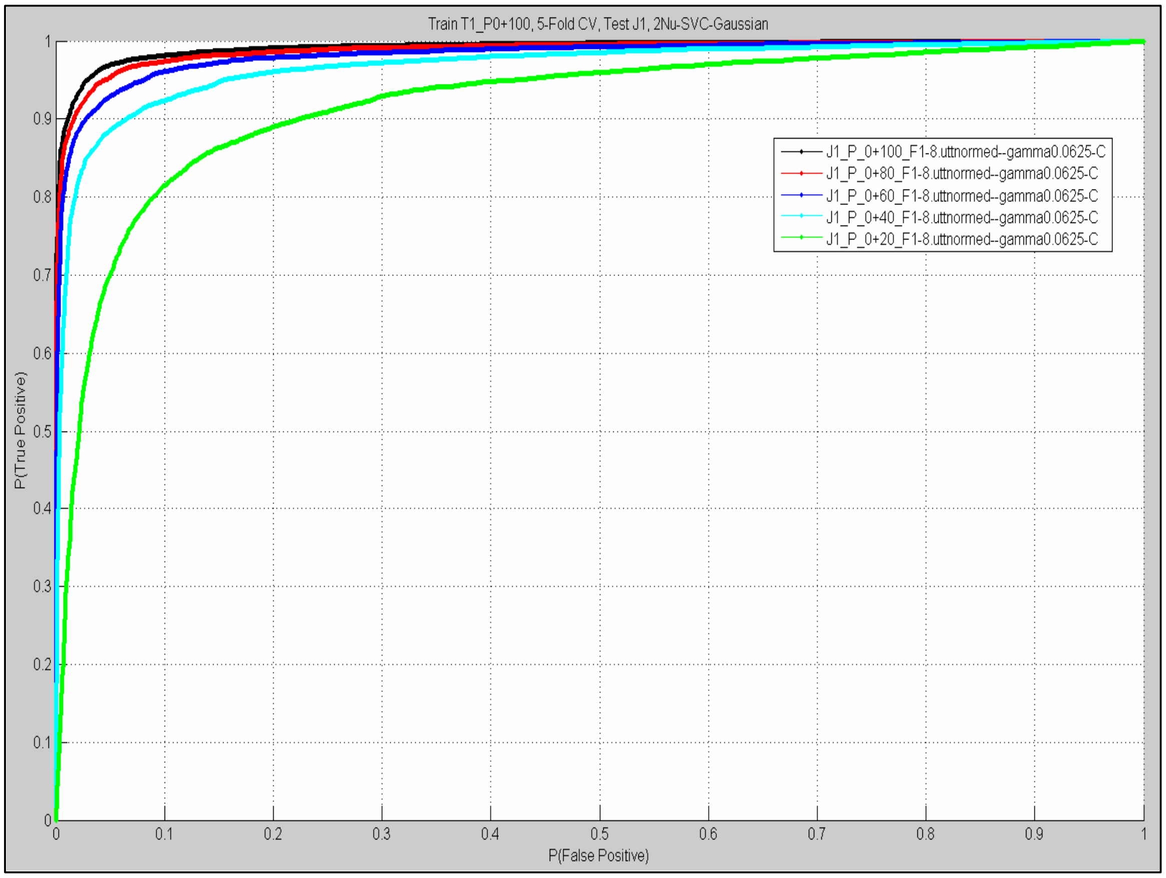


Figure 1 - Receiver Operator Characteristic (ROC) Curves showing the true positive rates versus false positive rates for a SVM classifier tested on data at various signal strengths.

2. A Brief Introduction to Cost-Sensitive Multi-Layer Perceptrons (MLPs) *(Status 1-31-2006)*

General Description
A Multi-Layer Perceptron or MLP is a neural network classifier that can approximate any posterior probability given enough training data and learnable parameters. It takes a vector of numbers, which are usually measurements of some meaningful features of the classes, and outputs a vector of class posterior probabilities.

How They Work
The MLPs in this report have two layers of computational nodes and are fully interconnected as shown in Figure 2. Each of the nodes encodes a hyperplane separation in the space of their inputs. The first layer, called the "hidden layer" is responsible for learning the initial separation boundaries in the input feature space. The second layer, called the "output layer", combines boundaries learned by the hidden layer, so that the resulting decision boundaries can be non-linear functions. A cartoon of the input space for a two class classification problem and the separators learned by a MLP is shown in Figure 3.

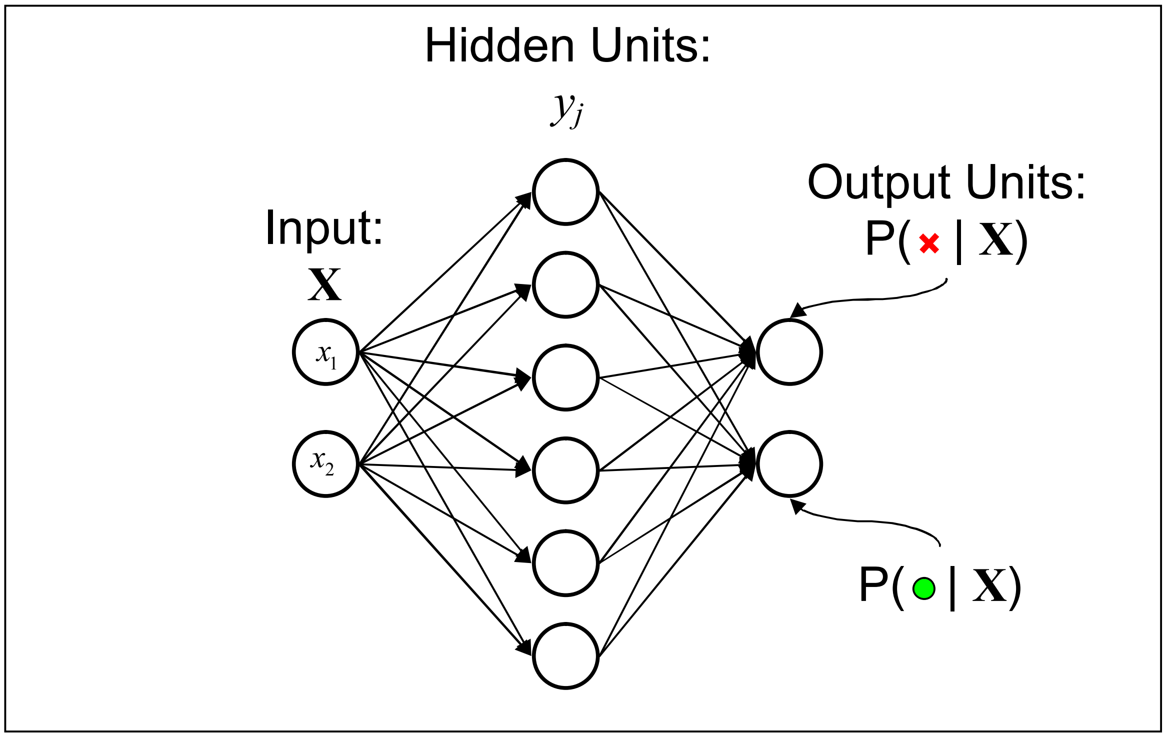


Figure 2 - The architecture of a Multi-Layer Perceptron (MLP).

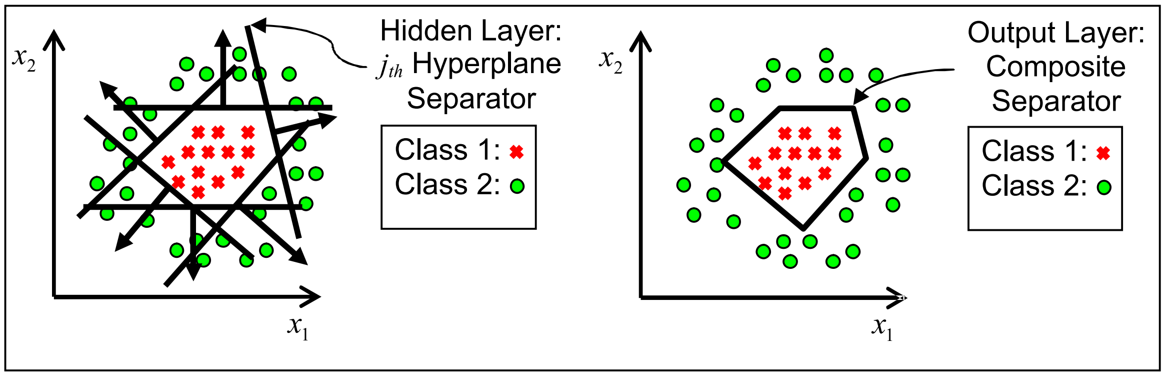


Figure 3 - Cartoon representation of the separators learned by the hidden and output layers of a MLP in the input space.

Mathematically, the output of the j_{th} hidden layer unit (hidden unit) is given by:

$$(2.1)\quad y_j = \frac{1}{1+e^{-(\mathbf{W}_j^T\mathbf{X}+B_j)}},$$

where $\mathbf{X}$ is the input vector, $\mathbf{W}_j^T$ is the transpose of the j_{th} hidden unit's learnable weights and B_j is the hidden unit's bias. Graphically, this function is a hyperplane ramp in the space of the inputs.

The posterior probability of class i is the output of the i_{th} output layer unit. It is given by:

$$(2.2)\quad \mathrm{P}(\text{class i}\,|\,\mathbf{X}) = \frac{e^{z_i}}{\sum\limits_{z_j \in \text{output units}} e^{z_j}},$$

$$(2.3)\quad z_i = \mathbf{O}_i^T\mathbf{Y} + C,$$

where Y is the vector of hidden layer outputs, $\mathbf{O}_i^T$ is the transpose of the i_{th} output unit's learnable weights and C is the output unit's bias.

During the training phase, a MLP is supplied with a training set of samples, each of which consists of an input vector and a target class label. The goal of the training process is to estimate the learnable parameters of the MLP (the hidden unit weights and bias and the output unit weights and bias) such that the MLP makes as few errors on this training set as possible. This is accomplished via the Error Back-Propagation algorithm [10] which is a gradient descent algorithm. The idea of this algorithm is to take an initial guess at the learnable parameters, compute the gradient of the error function with respect to the learnable parameter, and then update the learnable parameters in the opposite direction of the gradient which amounts to updating the parameters such that the error is reduced. Like other gradient descent algorithms, Error Back-Propagation suffers from the problem of local minima where the final learnable parameters give a locally minimal error rate which may be suboptimal.

To mitigate this problem, the MLPs in this report make larger updates to the parameters in the beginning, and then based on the performance on a cross-validation set, the update step sizes are reduced. More specifically, the user specifies an initial learning rate (update step size) that is used until performance on the cross-validation set does not improve by 0.5% absolute at which point the learning rate is halved after each epoch[1]. Finally, training is stopped as soon as the performance on the cross-validation set does not improve again by 0.5% absolute.

The error function used for training affects the decision boundary that the MLP ultimately learns. The standard error function used by MLPs described above is the cross-entropy error function given by:

$$(2.4)\ E = -\sum_{n \in samples} \sum_{k \in classes} t_{n,k} \ln\left(\frac{\mathrm{P}_{n,k}}{t_{n,k}}\right)$$

$t_{n,k}$ is the target value for the k_{th} class of the n_{th} sample and $\mathrm{P}_{n,k}$ is the output of the k_{th} output unit for the n_{th} sample and $\mathrm{P}_{n,k} \in [0,1]$. The closer the output of the MLP is to the target, the smaller this error will be. In a detection framework, where there are only two classes,

$$(2.5)\ t_{n,k} \in 0,1 \text{ and the target vector } t_n \in \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

When a MLP of the type described above is trained using this cross-entropy error function, the outputs of the MLP can be shown to approximate the posterior probabilities of the classes given the input vector [9].

There are two types of errors a MLP applied to a detection problem can make for any sample: false positive and false negative errors. Assuming that the second element of the target or output vector represents the positive class, then the following is an example of a false negative error: the MLP output vector for the n_{th} sample is $P_n = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$, and the target vector is $t_n = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Colloquially, this example shows that the MLP wishes to classify the sample as negative, while the sample is actually positive. On the other hand, if $P_n = \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$ and $t_n = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, then the MLP says the sample is positive when it actually was negative, which is a false positive error. Calculating the cross-entropy error for these two examples leads to a key observation: the error value for both the false positive and false negative example is 0.105. This means that the conventional cross-entropy error function penalizes both types of errors equally. Given the requirements of this project for achieving ultra-low false positive rates, it would be better if the error function penalizes the false positive errors more heavily than the false negative errors.

For this project, we developed a new cost-sensitive error function that allows the user to specify the penalties for each kind of error. We call this new error function the class-weighted cross-entropy error function, and it is given by:

[1] An epoch is one complete round of parameter updates over the entire set of training data.

$$(2.6)\quad EC = -\sum_{n\in samples}\sum_{k\in classes}\alpha_k t_{n,k}\ln\left(\frac{\mathrm{P}_{n,k}}{t_{n,k}}\right)$$

Where α_k is the penalty factor for errors on the k_{th} class. The larger the penalty factor for a certain class, the more weight an error made on that class will have. For example, setting $\alpha_0 = 1.0$ and $\alpha_1 = 0.1$ penalizes false positive errors 10 times more heavily than false negative errors. By penalizing false positive errors more heavily using this class-weighted error criterion, the decision boundaries that are ultimately learned will make fewer false positive errors than decision boundaries trained using the original error criterion. Finally, note that when using the class-weighted cross-entropy error function, the outputs of the MLPs may no longer be approximates of the class posterior probabilities. Please refer to [1] for further details about MLPs.

3. A Practical Guide for Applying Cost-Sensitive MLPs to a Classification Problem

The application of any classifier to a new data set consists of four main processes: data preparation, data preprocessing, training, and testing. The goal of data preparation is to put the input data into a form that the classifier can read and subdivide the data into sets for training, cross-validating, and testing. Once the data has been prepared, the preprocessing stage performs any feature normalization or transformations that might help the classifier perform better. The next step is to train the learnable parameters of the classifier on the training data. This involves a grid search over the user-specified training parameters of the classifier to find the settings that lead to the best performance on a cross-validation set. Using these best settings, a final training is performed resulting in a trained classifier. Finally, the trained classifier's performance on a test set can be measured. Figure 4 shows the flow diagram of these four processes, and Figure 5 displays a detailed flow diagram of the training process.

In what follows, we take a closer look at each of these processes and discuss details relevant for successfully applying cost-sensitive MLPs to a classification problem.

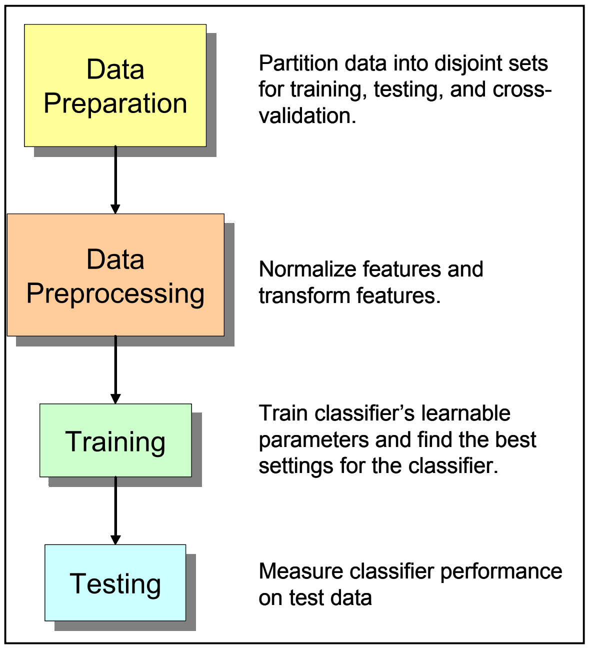

Data Preparation
Partition data into disjoint sets for training, testing, and cross-validation.
Data Preprocessing
Normalize features and transform features.
Training
Train classifier's learnable parameters and find the best settings for the classifier.
Testing
Measure classifier performance on test data

Figure 4 – Flow diagram of the four main processes involved in applying a classifier to a problem.

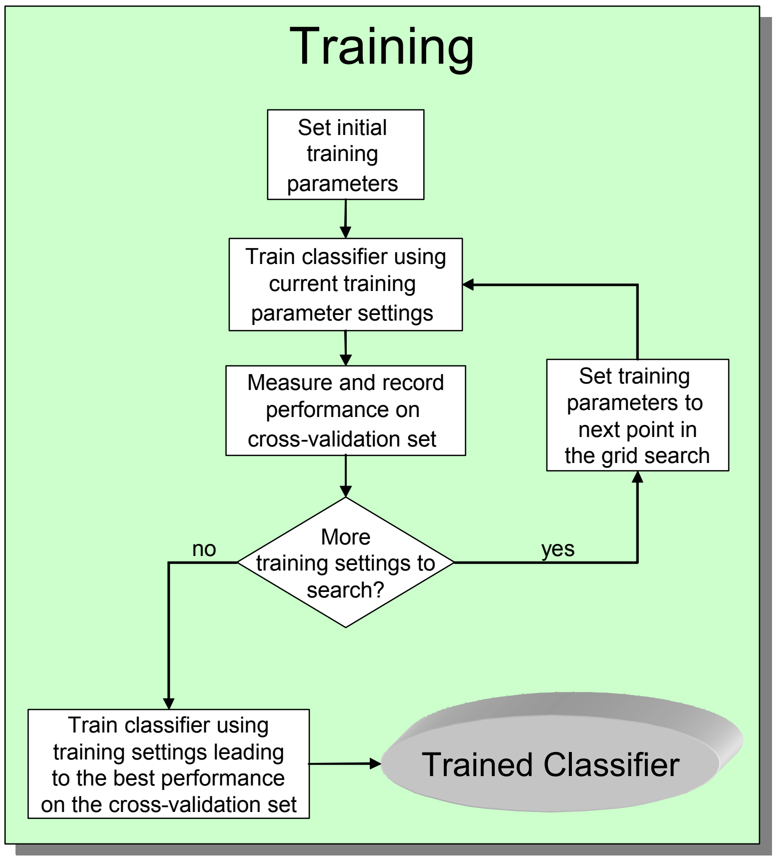

Training
Set initial training parameters
Train classifier using current training parameter settings
Measure and record performance on cross-validation set
More training settings to search?
no
yes
Set training parameters to next point in the grid search
Train classifier using training settings leading to the best performance on the cross-validation set
Trained Classifier

Figure 5 – Flow diagram of the training process in detail.

<u>Data preparation</u>

We extended the International Computer Science Institute's quicknet software [8] to perform cost-sensitive training in this project. Quicknet has two main MLP programs, one for training called qnstrn, and one for testing called qnsfwd. Both of these programs read in data from special binary files called "pfiles". Originally, developed for speech recognition applications, each data sample in a pfile consists of a sentence (also known as utterance) index, a frame index, followed by either a series of numbers representing the feature measurements or a label for the class that this sample belongs to. Every data sample is unambiguously indexed by its sentence index and frame index.

In the first step of the data preparation stage, we have perl scripts that read in ascii files containing the clean and corrupt samples and converts them into two pfiles. The first pfile contains the data sample's feature vectors, and the second pfile contains the data sample's class labels. The perl script createPfiles.pl is an example script for creating a feature and label pfile from ascii feature files like the ones provided in this project.

The second step of the data preparation stage involves separating the data samples into training, cross-validation, and test sets. In MLP training, the learnable parameters are continuously modified to give better and better performance on the training set, and the learning rate and stopping point of training is determined by its performance on a separate cross-validation set. After training is finished, we measure the MLP's performance on the test set. The MLP's training set and cross-validation set usually come from data originally designated for training the classifiers or else data that closely resembles future test data. For example, to build a cross-validation set from training data, randomly select 10% of this original training data for the cross-validation set, and then randomize the ordering of the remaining 90% to form the MLP training set. In the SVM sections, this same process was used, but we also experimented with an n-fold cross-validation technique and found little difference between n-fold cross-validation and randomly picking 10% of the data for cross-validation.

Preprocessing

The goal of the preprocessing stage is to transform the data in a way that makes the work of the classifier easier. The most common type of data transformation is normalization where the range that the feature values is shrunk or expanded. A standard normalization technique, which works well for MLPs, transforms each feature component to have zero mean and unit variance over the data set. This normalization technique works by first computing the mean and standard deviation of each feature component over the entire data set, and then subtracting the mean from each sample and dividing by the standard deviation. Even though it is theoretically true that MLPs could learn this normalization, in practice pre-normalizing the inputs to the MLPs lead to better performance since none of the learnable parameters have to be devoted to learning this normalization.

Besides normalization, there are other transformations that may make the original data points more separated in space. Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA) are examples of such feature transformations that people often use to pre-process their input data for the classifiers. In this project, we do not explore the application of these types of transformations.

One key point to remember is to be consistent with the transformations across data sets, i.e., apply the same transformation used for the training set on the cross-validation and test sets.

Training

In the training stage, we are looking for the best user-specified training parameters to use for training the MLP. In other words, we want to find the settings for parameters that will lead to MLPs that will most likely perform the best on the testing set. This is generally accomplished by training MLPs, one for each parameter setting, on the training data, and then picking the parameters for the MLP that gives the highest AUROC10^{-3} on a cross-validation set.

We use qnstrn as the training program for our MLPs. The user-specified parameters of significance for qnstrn MLPs are: the total number of hidden units, the class-weights, initial learning rate, and initial random seed. The total number of hidden units controls the power and flexibility of the MLP: more hidden units mean more trainable parameters which may result in more complicated decision boundaries. In practice, a good rule of thumb is to use 10 trainable parameters for every sample in the training set. The total number of trainable parameters, *numTrainableParameters*, in a two-layer MLP with *numInputFeatures* input features, *numHiddenUnits* hidden units, and *numOutputUnits* output units is calculated by:

$$(3.1)\quad numTrainableParameters \equiv$$
$$(numInputFeatures + 1)\cdot numHiddenUnits + (numHiddenUnits + 1)\cdot numOutputUnits$$

In most of the MLP experiments below, we simply set *numHiddenUnits* such that *numTrainableParameters* is about a tenth of the number of training data samples.

Equation 2.6 is the error function used for performing cost-sensitive MLP training. As discussed above, there are special α_k variables used for weighting the error term arising from the k_{th} class. In practice, we often search a limited range of these class-weights, so that we bias the training toward decision boundaries giving fewer false positives. A good set to search over for $\begin{bmatrix}\alpha_0\\ \alpha_1\end{bmatrix}$ is $\begin{bmatrix}1.0\\ 1.0\end{bmatrix},\begin{bmatrix}1.0\\ 0.1\end{bmatrix},\begin{bmatrix}1.0\\ 0.01\end{bmatrix}$.

The last two MLP training parameters to search over are the initial random seed and initial learning rate. The initial random seed is used for specifying the starting point in the gradient descent on the error functions, while the initial learning rate controls the size of the update steps that the algorithm takes on the error surface. In more detail, the initial random seed controls the random initial settings for all the learnable parameters in the MLP. It may be important in applications with relatively small amounts of training data (i.e., less than 1 million), to train many different MLPs each with a different initial random seed (the seed for selecting random initial settings for all the learnable parameters) since different starting points in the gradient descent algorithm can often lead to different local minima solutions. In many of the MLP experiments below, we use 25-50 different initial random seeds. The range to search over for the initial learning rate depends from problem to problem, so it is best to first search a range of initial learning rates that have a large range (e.g., $\log_{10}$(-4,-3,-2,-1,0,1)), and then search over smaller and finer ranges centered at the rate that gave the best performance in the previous wider search.

Testing

In the testing stage, all we simply do is run the MLP on the test data and measure its performance. More specifically, we run qnsfwd, the MLP forward-pass program that takes input samples and computes the outputs of a trained MLP. Recall that the outputs of the MLP are posterior probabilities of the classes. For each test sample, we compute the posterior probability of the positive class and collect the corresponding truth label (i.e., whether the sample is actually a positive class or not). We then place the computed posterior probabilities into a text file and the corresponding truth labels in another file and then use these files as input to our ROC program (ROCMain) that computes the ROC curve and AUROC10^{-3}.

4. A Brief Introduction to Cost-Sensitive Support Vector Machines (SVMs) *(Status 04-04-2006)*

<u>General Description</u>
Support Vector Machines (SVMs), like MLPs, are classifiers that learn nonlinear separating boundaries between the classes. SVMs are created from a combination of two techniques: the maximal margin classifier and the kernel trick. The maximal margin classifier is simply the linear hyperplane separator that best separates the two classes[2] while maximizing the distance between the closest training example and the hyperplane. Figure 6 shows a graphical depiction of a maximal margin classifier separating two classes.

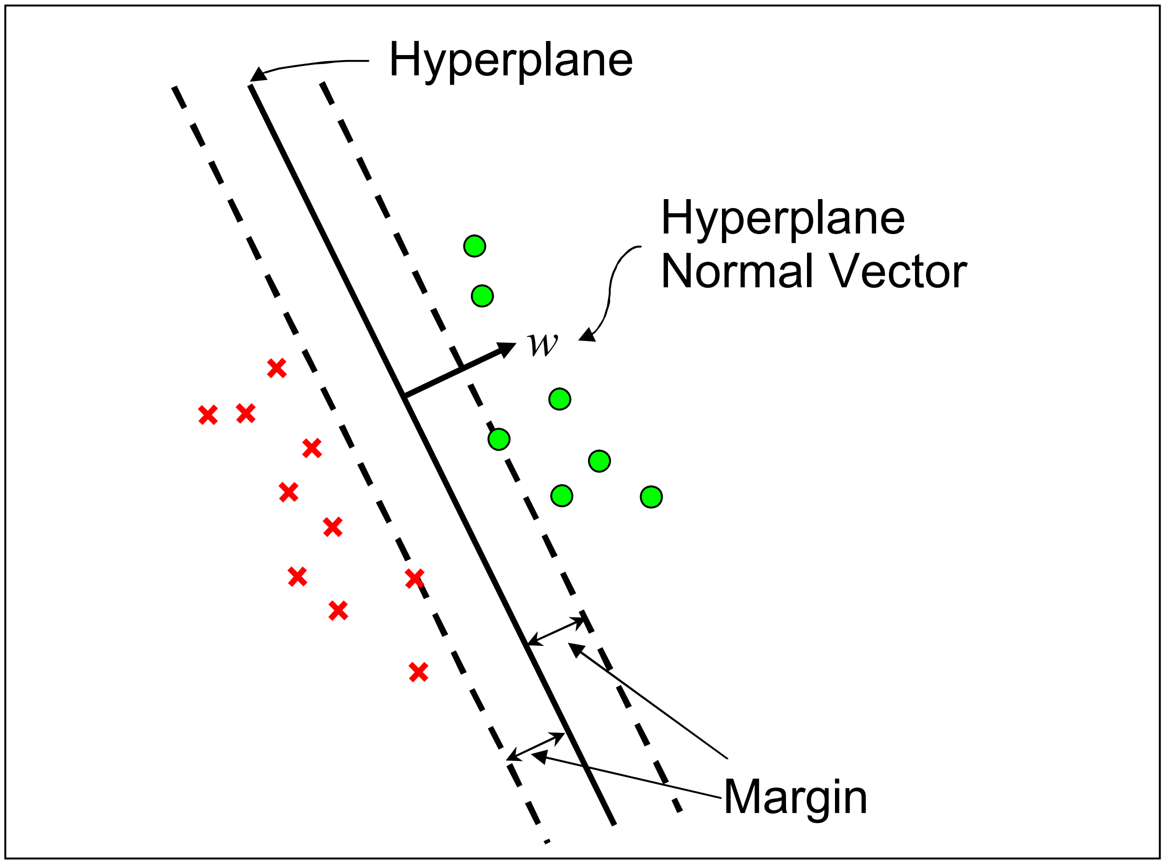


Figure 6 - The hyperplane separator learned by a maximal margin classifier.

The kernel trick is a powerful technique that is used to turn the linear decision boundary into a nonlinear one. This trick involves using a kernel function to perform dot product operations in “feature” space while operating on input space. More specifically, a kernel is a function operating on the input vectors in input space that represents the dot product of two input vectors mapped to a “feature” space which often has a much higher

[2] Typically SVMs solving multi-class classification problems consist of a set of two-class SVMs, one for every pair of two classes.

dimensionality than the original input space. Mathematically, a kernel function is given by: $k(x_1,x_2)=\langle\Phi(x_1),\Phi(x_2)\rangle$, where x_i are the input vectors, $\langle\cdot,\cdot\rangle$ is the dot product operator, and $\Phi(\cdot)$ is the mapping from input space to "feature" space.
The problem of computing the maximal margin hyperplane is a quadratic optimization problem whose constraints are expressed as dot products. By using a kernel for the dot products found in the maximal margin optimization problem, a maximal margin hyperplane is formed in the higher dimensional "feature" space which when projected back to the original input space becomes a nonlinear separation boundary. Figure 7 depicts the kernel trick applied to maximal margin classification. For a more extensive tutorial on SVMs, please see [2].

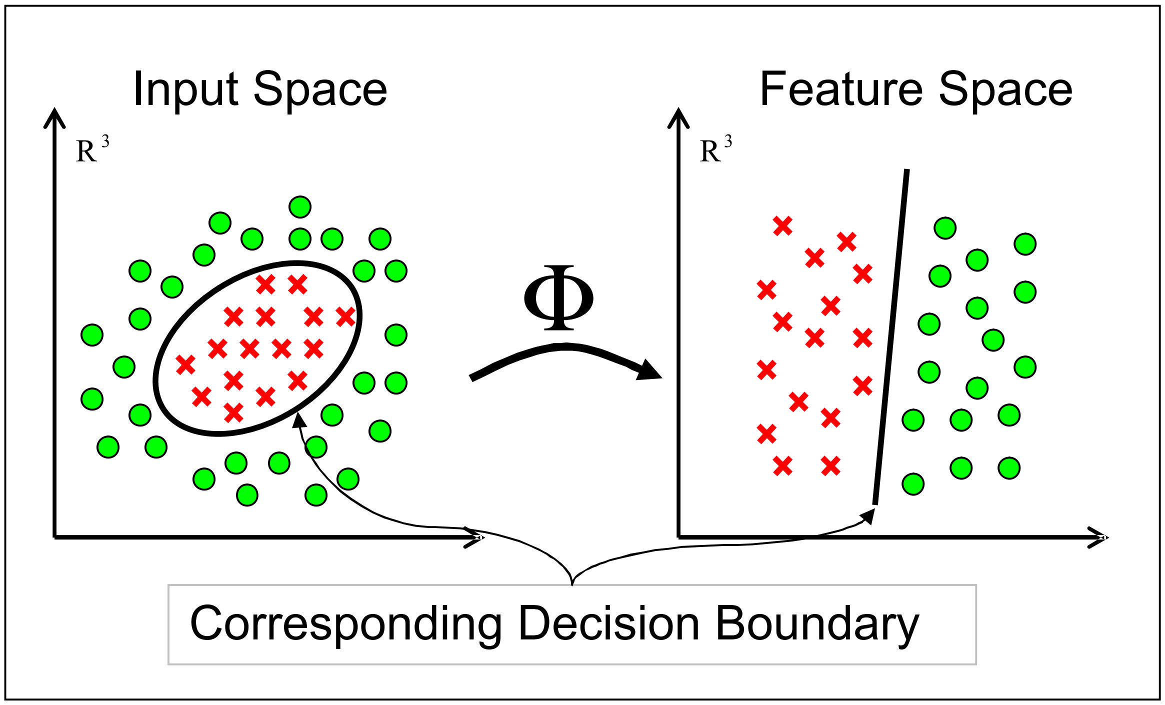


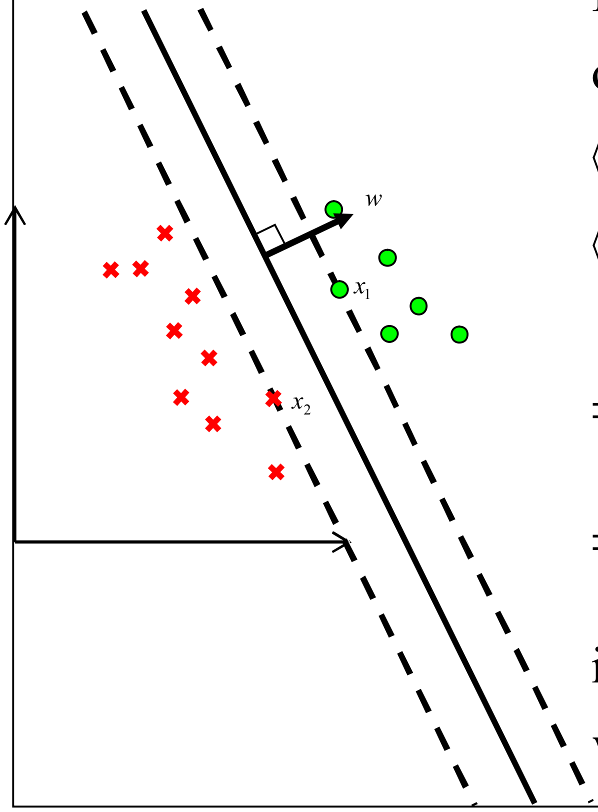
Figure 7 - The "kernel trick" allows one to learn a linear decision boundary in feature space that corresponds to a nonlinear one in input space.

Further Details on Maximal Margin Classifiers and the Kernel Trick
In this subsection we discuss some of the mathematics underlying the maximal margin classifiers which leads to the development of a cost-sensitive SVM. During the training phase, this type of SVM can penalize different errors more heavily so that the final result can be biased toward making as few false positive errors as possible. This subsection ends with several salient comments concerning kernel functions.

Figure 8 shows the geometry of the problem of finding the hyperplane that best separates two classes while maximizing the margin between the two classes. The margin is defined to be the distance to the closest point to the hyperplane. For a separable data set, the optimization problem can be expressed by the following:

$$(4.1)\quad \begin{array}{l}\text{minimize } \frac{1}{2}\|w\|^2, \\ \text{subject to } y_i(\langle x_i, w\rangle + b) \geq 1 \quad \forall\, i = 1,\ldots,m\end{array}$$

Where $||w||$ is the norm of the normal vector of the hyperplane, y_i is the i_{th} training sample's class label ($y_i \in -1,+1$), x_i is the i_{th} training sample's input vector, $\langle\cdot,\cdot\rangle$ is the dot product, b is the hyperplane's bias, and m is the total number of training samples. In layman's terms minimizing $\|w\|^2$ is equivalent to maximizing the margin (see Figure 8). The constraints specify that all training examples lie on the correct side of the hyperplane, i.e., all positive classes on one side and all negative classes on the other.



Let x_1 and x_2 be points on the margin.

$$\langle w, x_1\rangle + b = 1$$

$$\langle w, x_2\rangle + b = -1$$

$$\Rightarrow \langle w, (x_1 - x_2)\rangle = 2$$

$$\Rightarrow \left\langle \frac{w}{\|w\|}, (x_1 - x_2)\right\rangle = \frac{2}{\|w\|}$$

is the distance from x_1 to x_2 which is twice the margin.

Figure 8 - The geometry of the maximal margin classifier.

For a non-separable data set, there will be some training examples that lie on the wrong side of the hyperplane or that lie within the margin on the correct side of the hyperplane. For these samples, we introduce some "slack variables" which measure how far these samples lie from the margin of the correct side of the hyperplane. Slack variables are 0 for samples that are classified correctly and lie outside the margin; otherwise, slack variables measure the distance we have to move the sample in order to be classified correctly and lie on the margin. We denote the i_{th} slack variables by ξ_i , and the new optimization problem becomes:

$$\text{minimize } \frac{1}{2}\|w\|^2 + C\sum_i \xi_i,$$

(4.2) subject to

$$\xi_i \geq 0, \forall\, i = 1,\ldots,m$$
$$y_i(\langle x_i, w\rangle + b) \geq 1 - \xi_i, \forall\, i = 1,\ldots,m$$

As before minimizing $\|w\|^2$ maximizes the margin, but this time we would also like to minimize the sum of the slack variables or margin error. C is the parameter that controls the tradeoff between maximizing the margin and minimizing the errors. Larger C values result in hyperplanes that minimize the margin errors, while smaller C values lead to hyperplanes with larger margins. This type of SVM is call the C-Support Vector Classifier (C-SVC).

The SVMs discussed thus far have not addressed the issue of cost-sensitive training, i.e., biasing the training to make fewer false positive or fewer false negative errors. A cost-sensitive adaptation to SVMs is discussed in [4] and [5]. They present the 2Nu-Support Vector Classifier (2Nu-SVC) which uses the following optimization problem for learning the hyperplane separator:

$$\underset{w,b,\xi,\rho}{\text{minimize}} \ \frac{1}{2}\|w\|^2 - \nu\rho + \frac{\gamma}{m}\sum_{i\in I_+}\xi_i + \frac{1-\gamma}{m}\sum_{i\in I_-}\xi_i,$$

subject to

(4.3)
$$\xi_i \geq 0, \forall\, i = 1,\ldots,m$$
$$y_i(\langle x_i, w\rangle + b) \geq \rho - \xi_i, \forall\, i = 1,\ldots,m$$
$$\rho \geq 0$$

where I_+ and I_- are the positive and negative class training samples respectively, γ is the penalty factor for false negative errors, ν is a user specified parameter that controls the margin and error tradeoff, and ρ is an auxiliary variable that is solved for during the optimization proportional to the final margin. [4] and [5] point out that by reparameterizing ν and γ with:

$$\nu = \frac{2\nu_+\nu_-m_+m_-}{(\nu_+m_+ + \nu_-m_-)m}, \quad \gamma = \frac{\nu_-m_-}{\nu_+m_+ + \nu_-m_-} = \frac{\nu m}{2\nu_+m_+}$$

(4.4) or,

$$\nu_+ = \frac{\nu m}{2\gamma m_+}, \quad \nu_- = \frac{\nu m}{2(1-\gamma)m_-},$$

the user can work with parameters with nice theoretical interpretations. It can be shown that ν_+ (ν_-) is an upper bound on the fraction of margin errors on the training set from the positive (negative) class and a lower bound on the fraction of support vectors from the positive (negative) class. Moreover, both ν_+ and ν_- must lie between 0 and 1 for the solution to the optimization problem to be valid, which also makes the parameter search much cleaner. In practice a user just has to perform a search for ν_+ and ν_- on an evenly spaced grid spanning the square area between $\nu_+=0, \nu_-=0$ and $\nu_+=1, \nu_-=1$. In an application that requires low false alarm rates, ν_- should be set to smaller values since ν_- controls the upper bound on number of false alarms in the training set.

We conclude this section with several quick comments about kernel functions. There are many kernel functions that a user can choose to use. In fact, researchers are constantly developing new ones, but this does not mean that the existing kernel functions are going to give poor results. The most commonly used kernel function is the Gaussian kernel function. Its name comes from the form that the kernel takes and not from any requirement that the data be Gaussian. The feature space that a Gaussian kernel takes a dot product in is theoretically infinite dimensional, which makes the Gaussian kernel quite powerful. Also, because the Gaussian kernel has only one parameter to vary, it is also a simple kernel to work with. The Gaussian kernel is given by:

$$(4.5) \quad k(a,b) = \exp\left(\frac{-\|a-b\|^2}{\sigma^2}\right)$$

where a and b are input vectors and σ is the Gaussian kernel's width parameter that is chosen by the user. For the savvy user, experimenting with other kernel function may potentially lead to better performance, but in this work we work solely with Gaussian kernels.

5. A Practical Guide for Applying Cost-Sensitive SVMs to a Classification Problem

A successful application of SVMs for a classification problem involves the same four processing steps of a successful application of MLPs as discussed in Section 3. They are: data preparation, data preprocessing, training, and testing. In this section we discuss specifically how SVMs relate to these four processing stages.

Data Preparation

In this project we have been using the LIBSVM [3] software package for training and testing our SVM models (svm-train for training and svm-predict for testing). LIBSVM requires the input data file to be in a space-delimited format such that the first element in a line is the label (-1 or 1) of the sample followed by the sample's feature measurements preceded by the feature index number and a colon, e.g. "-1 0:0.34384 1:1.37839 2:0.098765".

In the MLP case, we divided the training data into a train and cross-validation set, where the cross-validation set was used for the grid search on the MLP parameters. For SVM training, we will also be using cross-validation sets to test our SVM parameter settings. Because we are using LIBSVM, we have the option of creating a single cross-validation set (as in the MLP case), or we may choose to perform n-fold cross-validation. svm-train has a built-in n-fold cross-validation option. This means that it will randomly partition the input training set into n equally populated sets and then perform training on all n-1 set combinations using the remaining set as the cross-validation set. At the end of n-fold cross-validation training, the average performance score (AUROC10^{-3}) on each of the n cross-validation sets is reported.

Data Preprocessing

As in the MLP case, normalizing the features is critical for the successful application of SVMs to a classification problem. We typically perform the same type of feature normalization used for MLPs, i.e., zero mean and unit variance normalization computed within a data set.

Training

Like the training phase for MLPs, the training phase for SVMs consists of two separate steps: the first step is to find the best SVM training parameter settings via cross-validation, and the second step is to train the SVM on the training set using the best parameter settings.

For C-SVCs the training parameters of significance are C, the parameter controlling the tradeoff between margin maximization and error minimization, and σ, the width of the Gaussian kernel. The first step is to find settings for C and σ that gives the best performance as measured by AUROC10^{-3} on a cross-validation set. As mentioned above, with LIBSVM the user can choose to use a single cross-validation set (like the MLP case) or to use n-fold cross-validation. We have found that the difference in performance between the two choices of cross-validation is small. To find the best settings for C and

σ, perform a grid search over values of C and σ. At each grid point, train a SVM with the values of C and σ corresponding to the values at the grid point and measure the performance of the resulting SVM model on the cross-validation set. A useful strategy for choosing the grid values is to start with a large range and set the distance between grid points logarithmically. For example, search over the set of C values $\log_2(-5)$, $\log_2(-4)$, $\log_2(-3)$, … , $\log_2(10)$. Larger C values result in looser decision boundaries, while smaller C values result in tighter decision boundaries. Likewise, smaller σ values lead to tighter decision boundaries, while larger σ values lead to looser ones. After finding the best values of C and σ, train the C-SVC on the entire training set using these values. See [7] and [11] for more details on practically training SVMs.

The parameters of significance for 2Nu-SVCs are ν_+ (upper bound on false negative margin errors), ν_- (upper bound on false positive margin errors), and σ (the width of the Gaussian kernel). Finding the best settings for these parameters involves a 3-d grid search, which makes the 2Nu-SVC training phase more time consuming than the C-SVC training phase which involves a 2-d grid search. As pointed out in [4] and [5] the advantage of parameterizing the 2Nu-SVC using ν_+ and ν_- is that the user simply needs to search for the best setting of ν_+ and ν_- over a uniform unit grid $[0,1]\times[0,1]$. Using coarser spacing between grid points at the beginning and then zooming in to finer spacing at the best operating point in the coarser level is a good strategy for performing this grid search. Finally, search over values of σ in the same way described above for C-SVCs. After finding the best values of ν_+, ν_-, and σ, train the 2Nu-SVC on the entire training set using these values.

Testing

The testing phase for SVMs, simply involves computing the side of the decision boundary as well as the distance to the decision boundary of each sample in the test set. The side tells us what class the SVM has classified a sample, while the distance is a measure of how sure the SVM believes the classification to be true. We have modified svm-predict to output this distance information. Using this distance information, we can plot ROC curves and calculate AUROC10^{-3} using our ROCMain program.

6. Experiments Using MLPs and SVMs on the Classification Data Sets

In the following subsections, we will summarize the various experiments performed using MLPs and SVMs on the classification data sets by summarizing the main findings and describing the experiment via the experimental parameters table. Also, each subsection contains references to the relevant status report which the reader can refer to for more figures and details.

Each experiment can be uniquely described by the set of experimental parameters and their settings. For MLPs, this set of experimental parameters includes:

Training set	What data is used to train the MLP?
Cross-validation set	What data is used for controlling the learning rate schedule and for determining the optimal initial learning and initial random seed?
Testing set	What data is used to test the MLP?
Features used	Which features are used as input (e.g., F1,F5,&F8)?
Feature normalization	What kind of normalization is applied to the input features?
Cross-validation performance criteria	Which performance measure is used to rank and control the learning rate schedule for MLPs during cross-validation (e.g., accuracy, precision, AUROC)?
Testing performance criteria	Which performance measure is used during testing (e.g., accuracy, precision, AUROC)?
Class-weights	What importance weighting is given to negative and positive classes respectively (e.g., 10,1 weights negative class 10 time more than positive class)?
Learning rate search range	What range of learning rates is used for the grid search?
Number of initial random seeds	How many times is MLP training repeated using a different initial starting point?
Number of hidden units or trainable parameters	How many hidden units or trainable weights are in the MLP?

For SVMs, the set of experimental parameters consists of:

Training set	What data is used to train the SVM?
Cross-validation set	What data is used for determining the optimal Gaussian kernel width, C, or ν_- and ν_+?
Testing set	What data is used to test the SVM?
Features used	Which features are used as input (e.g., F1,F5,&F8)?
Feature normalization	What kind of normalization is applied to the input features?
Cross-validation performance criteria	Which performance measure is used to rank the SVMs trained with different training settings during cross-validation (e.g., accuracy, precision, AUROC)?
Testing performance criteria	Which performance measure is used during testing (e.g., accuracy, precision, AUROC)?
Kernel type	What type of kernel is used (all SVM experiments below use the Gaussian kernel)?
Gaussian kernel width search range	What range of Gaussian widths, σ, is used for the grid search?
C search range	(For non-cost-sensitive SVM training only) What range of C is used for the grid search? Note that this is only the initial search range since successive grid searches focus the search on maxima of the previous grid search.
ν_- and ν_+ search range	(For cost-sensitive SVM training only) What range of ν_- and ν_+ is used for the grid search? Note that this is only the initial search range since successive grid searches focus the search on maxima of the previous grid search.

Experiment 1: Accuracy and Precision of MLPs on G2 and K35 Sets Using F1-8 With or Without A (Status 11-15-2005)

Summary:
In this experiment we apply MLPs on six different pairings of G2 and K35 as training and test sets using features 1-8 with and without the A-feature. The main findings are:

- Accuracy and precision are higher for same-type training/testing (e.g., train on G2 and test on G2) than different-type training/testing.
- The A-feature seems to help in same-type training/testing and hurt in different-type training/testing.
- The number of hidden units in the MLPs does not affect performance greatly.

Experimental Settings:
The cross-validation sets used in this experiment are the same as the test set, so the resulting performance is an optimistic estimate of what we can expect in a fair test where the cross-validation set is disjoint from the test set.
Training and testing sets:

- G2+G2_P100
  - Train and test on clean and corrupt G2 (50% training, 50%test)
- G2train-K35test
  - Train on clean and corrupt G2, test on clean and corrupt K35 (both P50 and P100)
- K35+K35_P100
  - Train on clean and P100 K35 (50% training, 50%test)
- K35+K35_P50
  - Train on clean and P50 K35 (50% training, 50%test)
- K35+K35all
  - Train on clean and P50 & P100 K35 (50% training, 50%test)
- K35train-G2test
  - Train on clean and corrupt (both P50 and P100) K35, test on clean and corrupt G2

Features used	F1-F8 or F1-F8+A
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	Accuracy or precision
Testing performance criteria	Same as cross-validation performance criteria: Accuracy or precision
Class-weights	Not used (i.e., equal class-weights)
Learning rate search range	From 0.001 to 2.5 in increments of 0.005
Number of initial random seeds	One
Number of hidden units	10, 20, 40, 80, 160, 320, vs. 640

Tables and Graphs:

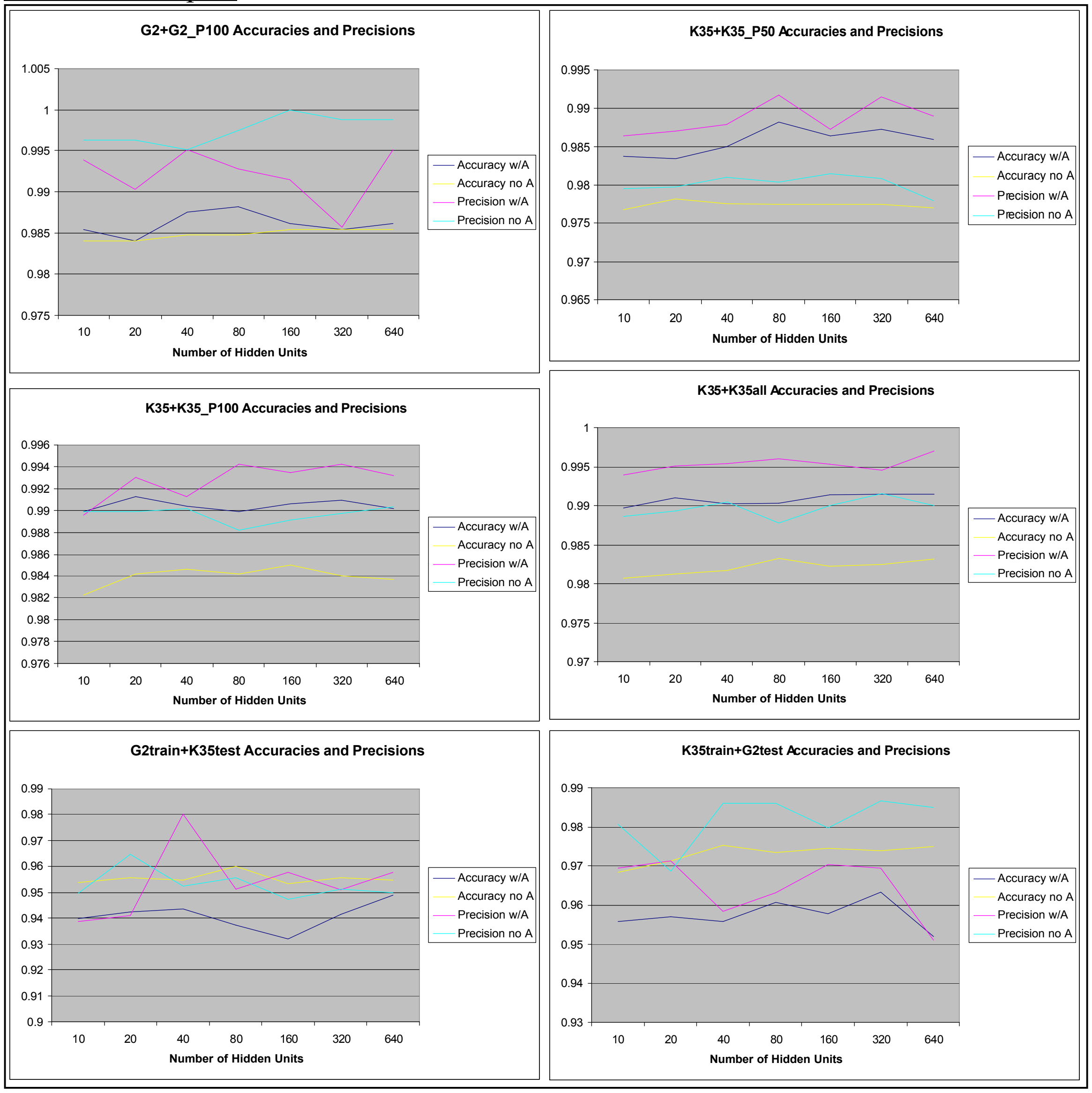


Figure 9 – Graphs of accuracy and precision versus the number of MLP hidden units for the six different training and testing conditions.

Experiment 2: Accuracy and Precision of MLPs Trained on G2 and Tested on K35 Using All the Possible Feature Combinations

(Status 12-08-2005 & Status 12-22-2005)

Summary:

In this experiment we investigate how different input feature combinations affect the accuracy and precision of the MLPs that use them. This study is restricted to the case of training on G2 data, cross-validating on a random half of all K35 data, and testing on the remaining half of the K35 data. Here are the main findings:

- The best feature combination in terms of accuracy and precision for MLPs is not F1-F8. For accuracy, the best is F1,F2,F5,F6 and for precision, the best is F2,F4,F5,F7.
- Using accuracy and precision as performance criteria for cross-validation and testing does not lead to a high true positive rate in the low false positive rate regions (< 10e-3).

Experimental Settings:

Training set	G2 and G2_P_100
Cross-validation set	Random half of K35, K35_P_50, and K35_P_100
Testing set	Remaining half of K35, K35_P_50, and K35_P_100
Features used	All 511 possible combinations of F1-F8, and A
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	Accuracy or precision
Testing performance criteria	Same as cross-validation performance criteria: accuracy or precision
Class-weights	Not used (i.e., equal class-weights)
Learning rate search range	0.05 to 1.7 with increments of 0.05
Number of initial random seeds	One
Number trainable parameters	About 300

Tables and Graphs:

features	TP	TN	FP	FN	Accuracy	Precision
F2,F4,F5,F7	0.868	0.9914	0.0086	0.132	0.9108	0.9948
F1,F5	0.8494	0.9902	0.0098	0.1506	0.8982	0.9939
F1,F4,F5,F6,F7	0.8283	0.99	0.01	0.1717	0.8844	0.9936
F1,F4,F5,F7,F8	0.825	0.99	0.01	0.175	0.8823	0.9936
F1,F2,F4,F5	0.7971	0.9904	0.0096	0.2029	0.8641	0.9936
F8	1	0	1	0	0.6531	0.6531
F7	1	0	1	0	0.6531	0.6531
F6	1	0	1	0	0.6531	0.6531
F6,F7,F8	1	0	1	0	0.6531	0.6531
F7,F8	1	0	1	0	0.6531	0.6531
F6,F7	1	0	1	0	0.6531	0.6531
F6,F8	1	0	1	0	0.6531	0.6531

Table 2– Top and bottom five feature combinations chosen and ranked by precision.

features	TP	TN	FP	FN	Accuracy	Precision
F1,F2,F5,F6	0.9694	0.9702	0.0298	0.0306	0.9696	0.9839
F1,F2,F5,F8	0.9768	0.9517	0.0483	0.0232	0.9681	0.9744
F2,F5,F8	0.9667	0.9644	0.0356	0.0333	0.9659	0.9808
F2,F5	0.9685	0.96	0.04	0.0315	0.9655	0.9785
F2,F5,F7	0.9668	0.9618	0.0382	0.0332	0.9651	0.9794
F8	1	0	1	0	0.6531	0.6531
F7	1	0	1	0	0.6531	0.6531
F6	1	0	1	0	0.6531	0.6531
F6,F7,F8	1	0	1	0	0.6531	0.6531
F7,F8	1	0	1	0	0.6531	0.6531
F6,F7	1	0	1	0	0.6531	0.6531
F6,F8	1	0	1	0	0.6531	0.6531

Table 3 - Top and bottom five feature combinations chosen and ranked by accuracy.

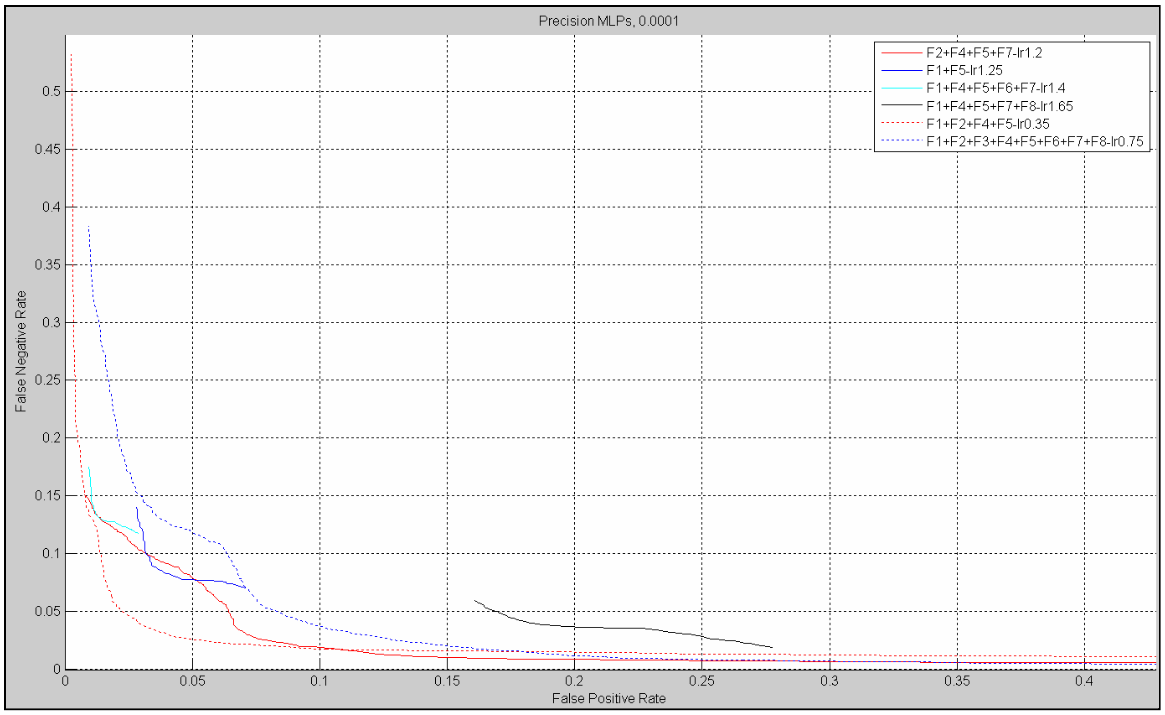


Figure 10 - False negative rate versus false positive rate curves of MLPs trained on G2 data with the best precision tested on K35 data. The best precision MLP (F1+F2+F4+F5-lr0.35) achieves about a 30% true positive rate at a false positive rate of 10e-3.

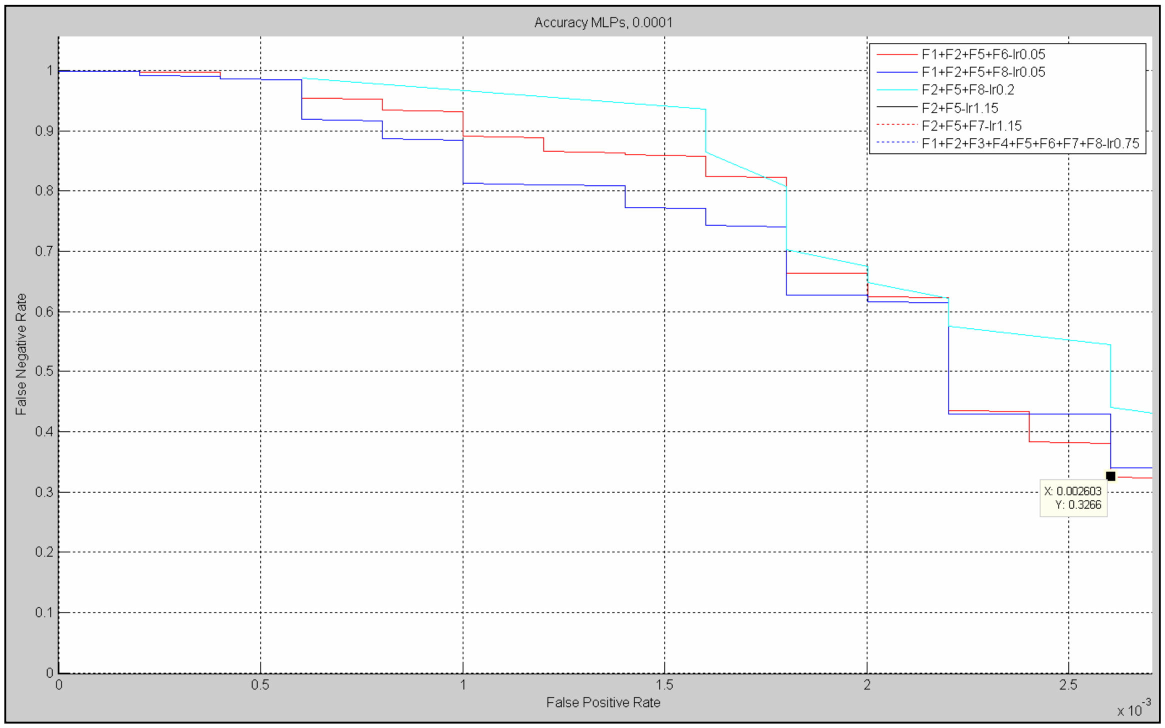


Figure 11 - False negative rate versus false positive rate curves of MLPs trained on G2 data with the best accuracy tested on K35 data. Note that x-axis is scaled by 10^{-3} and that true positive rates (1-false negative rate) are less than 20% for false positive rates less than 10e-3.

Experiment 3: Accuracy and Precision of Cost-Sensitive MLPs Trained on G2 and Tested on K35 Using F1-8 and F1, F2, and F5

(Status 01-31-2006)

Summary:

In this experiment we compare the accuracy and precision of MLPs trained with and without class weightings. More specifically, we compare MLPs trained using equal class-weights with MLPs trained using class-weights of 1.0 and 0.1 which penalizes false positive errors ten times more than false negative errors. As in experiments 1 and 2, the best initial learning rate for the MLPs is chosen by picking the one that leads to the highest accuracy or precision on the cross-validation set. The main findings are:

- Using a class-weighting that penalizes false positives ten times more than false negatives consistently reduces the false positive rate when using the 0.5 decision threshold.
- However, operating characteristic curves show that unequal class-weighting only helps in certain regions.
- The problem is that picking the MLP that has the best accuracy or precision on the cross-validation set does not usually lead to the MLP that has the lowest false alarm rate and simultaneously the highest true positive rate.

Experimental Settings:

Training set	G2 and G2_P_100
Cross-validation set	Random half of K35, K35_P_50, and K35_P_100
Testing set	Remaining half of K35, K35_P_50, and K35_P_100
Features used	F1-F8 and F1+F2+F5 (one of the better feature combinations from experiment 2)
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	Accuracy or precision
Testing performance criteria	Same as cross-validation performance criteria: accuracy or precision
Class-weights	Equal class-weights (1,1) and 10x false alarm penalizing (1.0, 0.1)
Learning rate search range	0.05 to 1.7 with increments of 0.05
Number of initial random seeds	One
Number of trainable parameters	About 300

Tables and Graphs:

	Initial Learning Rate	False Pos	False Neg
1,2,3,4,5,6,7,8-normed-byacc	0.85	0.10633	0.01244
1,2,3,4,5,6,7,8-normed-byprec	0.8	0.04105	0.1121
1,2,3,4,5,6,7,8-normed-classweight-byacc	1.35	0.05226	0.10306
1,2,3,4,5,6,7,8-normed-classweight-byprec	1.4	0.02663	0.13359

Table 4 - The false positive rate and false negative rate (at decision threshold = 0.5) of MLPs trained with equal class-weights (1,2,3,4,5,6,7,8-normed-byacc and 1,2,3,4,5,6,7,8-normed-byprec) and MLPs trained with false alarm penalizing class-weights (1,2,3,4,5,6,7,8-normed-classweight-byacc and 1,2,3,4,5,6,7,8-normed-classweight-byprec). Note that "byacc" refers to the fact that the MLP with the highest accuracy on the cross-validation set was chosen, while "byprec" means that the MLP with the highest precision was chosen.

	Initial Learning Rate	False Pos	False Neg
1,2,5-normed-byacc	0.05	0.02663	0.04159
1,2,5-normed-byprec	1.4	0.01382	0.09987
1,2,5-normed-classweight-byacc	1.4	0.01522	0.08615
1,2,5-normed-classweight-byprec	0.3	0.00721	0.12636

Table 5 - The false positive rate and false negative rate (at decision threshold = 0.5) of MLPs trained with equal class-weights (1,2,5-normed-byacc and 1,2,5-normed-byprec) and MLPs trained with false alarm penalizing class-weights (1,2,5-normed-classweight-byacc and 1,2,5-normed-classweight-byprec). Note that "byacc" refers to the fact that the MLP with the highest accuracy on the cross-validation set was chosen, while "byprec" means that the MLP with the highest precision was chosen.

Experiment 4: Area Under the ROC (AUROC) of MLPs and Cost-Sensitive MLPs Trained on G2 and Tested on K35 Using F1-8
(Status 02-21-2006)

Summary:
In this experiment, we use the area under the receiver operating characteristic curve from false alarm rates 0 to 10^{-3} (AUROC10^{-3}) as the performance criteria during cross-validation and testing. We compare the performance on K35 data of MLPs trained on G2 data using equal class-weights with MLPs trained on G2 using false alarm penalizing class-weights. The main findings show:

- MLPs trained with false alarm penalizing class-weights consistently achieve higher true positive rates at lower false alarm rates than those trained with equal class-weights.
- Using AUROC10^{-3} as the performance criteria during cross-validation leads to better performing MLPs than MLPs chosen using accuracy.

Experimental Settings:

Training set	G2 and G2_P_100
Cross-validation set	Random half of K35, K35_P_50, and K35_P_100
Testing set	Remaining half of K35, K35_P_50, and K35_P_100
Features used	F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	Accuracy or AUROC10^{-3}
Testing performance criteria	Same as cross-validation performance criteria: accuracy or precision
Class-weights	Equal class-weights (1,1) and false alarm penalizing class-weights (1.0, 0.1) , (1.0, 0.01), (10, 1), (100, 1)
Learning rate search range	Depends on class-weights (1,1) – from 0.05 to 1.0 with increments of 0.1 (1.0, 0.1) – from 0.5 to 3 with increments 0.25 (1.0, 0.01) – from 1.0 to 10.0 with increments of 1.0 (10, 1) – from 0.01 to .1 with increments of 0.01 (100, 1) – from 0.01 to .1 with increments of 0.01
Number of initial random seeds	100
Number of trainable parameters	About 300

Tables and Graphs:

Accuracy-Based Cross-Validation		
Class Weights	Best Accuracy	Best AUROC10-3
1,1	95.50%	3.31E-04
1,0.1	95.37%	5.00E-04
1,0.01	91.63%	5.65E-04
10,1	91.87%	3.75E-04
100,1	91.91%	5.20E-04

AUROC-Based Cross-Validation		
Class Weights	Best Accuracy	Best AUROC10-3
1,1	95.12%	3.93E-04
1,0.1	92.94%	6.31E-04
1,0.01	91.57%	6.33E-04
10,1	91.66%	4.26E-04
100,1	91.91%	6.52E-04

Table 6 - Tables of accuracy and AUROC10^{-3} on the testing half of all K35 data for MLPs trained using various class-weights on G2. The left table shows the results when using accuracy as the cross-validation performance criteria, while the right table shows the results when using AUROC10^{-3} as the cross-validation performance criteria. Using AUROC10^{-3} in cross-validation leads to higher AUROC10^{-3} in testing.

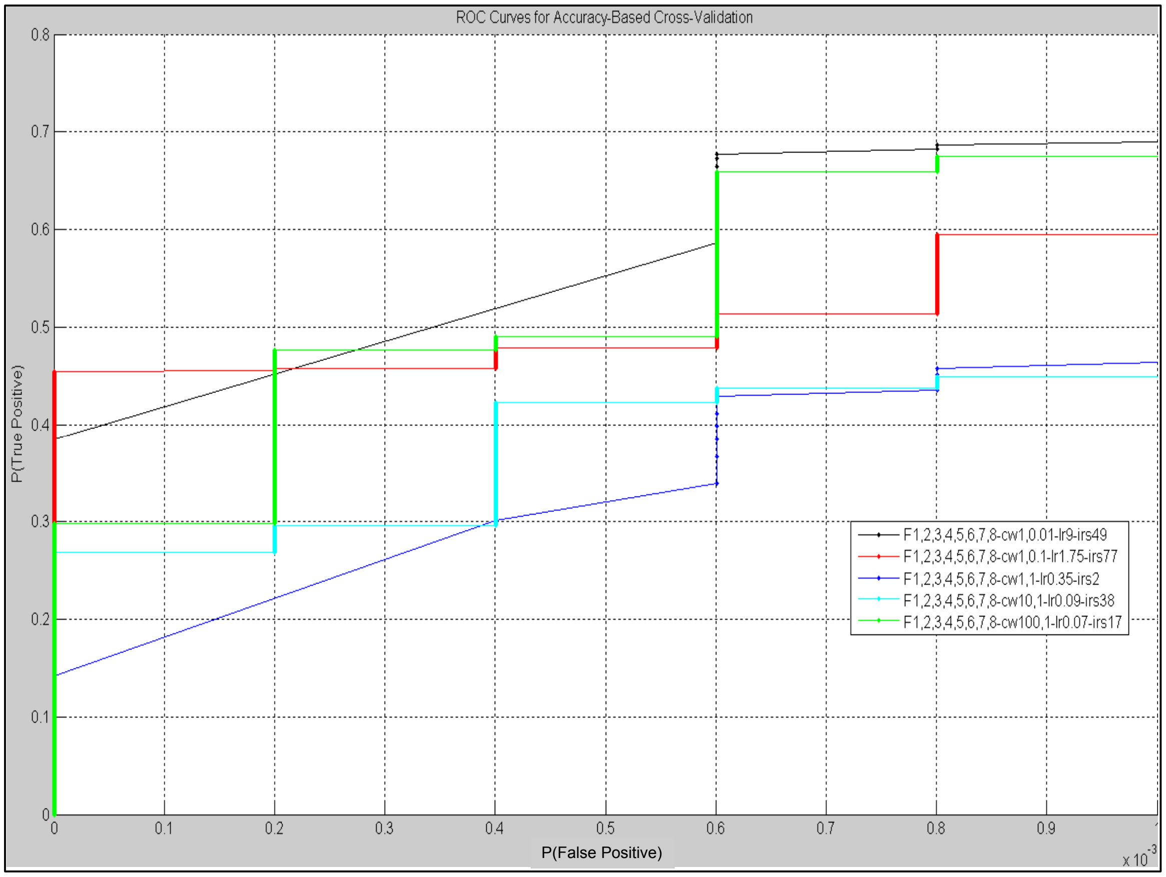


Figure 12 – ROC curves on half of all K35 data for MLPs trained on G2 using different class-weights (e.g., "cw1,0.01" refers to setting the class-weights as (1, 0.01)) and *accuracy* as the cross-validation performance criteria. All MLPs using false alarm penalizing class-weights have higher AUROC10^{-3} than the MLP with equal class-weights. Note that the x-axis is scaled by 10^{-3}.

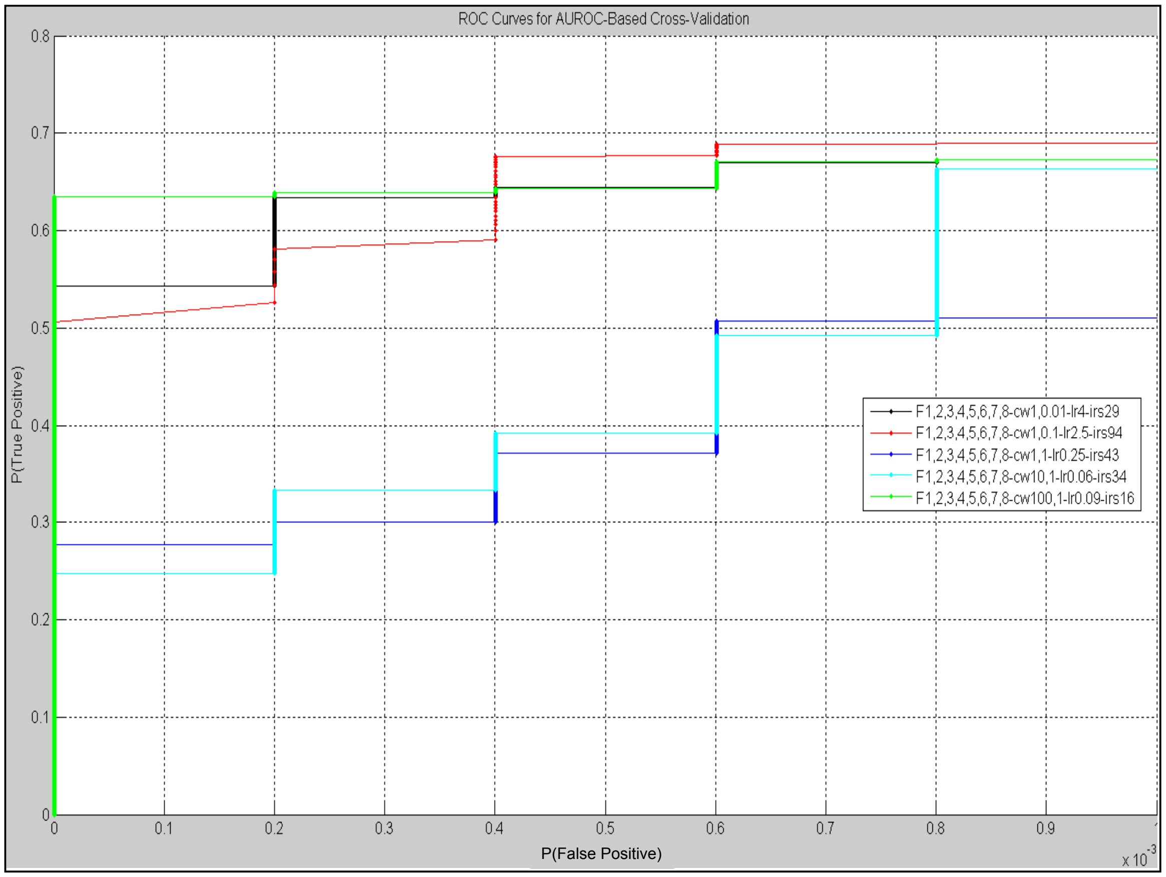


Figure 13 - ROC curves on half of all K35 data for MLPs trained on G2 using different class-weights (e.g., "cw1,0.01" refers to setting the class-weights as (1, 0.01)) and *AUROC10^{-3}* as the cross-validation performance criteria. All MLPs using false alarm penalizing class-weights have higher AUROC10^{-3} than the MLP with equal class-weights. Note that the x-axis is scaled by 10^{-3}.

Experiment 5: AUROC of MLPs and Cost-Sensitive MLPs Trained on T1 Sets and Tested on J1 Sets Using F1-8 and the Effect of Cross-Validation Data on Performance *(Status 02-28-2006)*

Summary:

In this experiment, we investigate how the choice in cross-validation data affects the performance (AUROC10^{-3}) on J1 data of MLPs trained using equal and false alarm-penalizing class-weights on T1 data. The main findings are:

- Cross-validation data that more closely resemble test data lead to MLPs that perform better on the test data.
  - Using a T1 cross-validation set leads to MLPs that perform very well on T1, but they do not perform as well as MLPs that use J1 cross-validation sets when tested on J1.
  - Increasing the amount of T1 data or J1 data in the cross-validation set does not always lead to better performance on J1 test data.
- There is some evidence that training on harder data (P_40) leads to MLPs that perform better on easy data (P_100) than MLPs trained on easy data (P_100).

Experimental Settings:

Training/Cross-Validation sets:

1. CV with 2000 samples from T1 & T1_P_100, Train with remaining[3] T1 & T1_P_100 data
2. CV with 2000 samples from J1 & J1_P_100, Train with T1 & T1_P_100
3. CV with 9700 samples from T1 & T1_P_100, Train with remaining[2] T1 & T1_P_100 data
4. CV with 9700 samples from J1 & J1_P_100, Train with T1 & T1_P_100.
5. Train with T1 & T1_P_40, CV with 2000 samples from J1 & J1_P_40.

Testing set	Remaining[2] J and J1_P_100 data, J1_P_80, J1_P_60, J1_P_40, and J1_P_20.
Features used	F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	AUROC10^{-3}
Testing performance criteria	AUROC10^{-3}
Class-weights	Equal class-weights (1,1) and false alarm penalizing class-weights (1.0, 0.1) , (1.0, 0.01), (10, 1), (100, 1)
Learning rate search range	Depends on class-weights (1,1) – from 0.05 to 1.0 with increments of 0.1 (1.0, 0.1) – from 0.5 to 3 with increments 0.25 (1.0, 0.01) – from 0.5 to 6 with increments of 0.5 (10, 1) – from 0.005 to .1 with increments of 0.01 (100, 1) – from 0.005 to .1 with increments of 0.01
Number of initial random seeds	50
Number of trainable parameters	About 300

[3] Note that in all cases, the cross-validation set is disjoint from the test set, which means no sample occurs in both the cross-validation set and test set.

Tables and Graphs:

	ClassWeights	CV	J1_P_100	J1_P_80	J1_P_60	J1_P_40	J1_P_20
Train=T1_P0&P100 CV=T1_P0&P100 (2000 Samples)	1,1	97.44%	37.40%	21.53%	8.17%	2.35%	0.33%
	1,0.1	97.24%	56.83%	46.13%	31.86%	13.04%	0.73%
	10,1	97.32%	39.93%	26.43%	8.27%	1.22%	0.06%
	1,0.01	97.02%	7.11%	1.99%	0.25%	0.04%	0.04%
	100,1	97.22%	49.50%	33.01%	13.40%	1.87%	0.22%

	ClassWeights	CV	J1_P_100	J1_P_80	J1_P_60	J1_P_40	J1_P_20
Train=T1_P0&P100 CV=J1_P0&P100 (2000 Samples)	1,1	77.69%	54.98%	40.85%	19.38%	4.11%	0.41%
	1,0.1	82.02%	67.99%	64.02%	48.02%	23.15%	1.64%
	10,1	80.87%	52.96%	40.47%	22.62%	2.68%	0.35%
	1,0.01	81.81%	63.93%	56.50%	41.16%	12.40%	0.52%
	100,1	82.55%	58.87%	51.61%	34.60%	6.95%	0.60%

Table 7 - AUROC10^{-3} percentages of MLPs trained using various class-weights. The upper table reports numbers from MLPs trained with cross-validation data from 2000 samples of T1, while the lower table reports numbers from MLPs trained with cross-validation data from 2000 samples of J1. AUROC10^{-3} percentages are higher on J1 testing data when the MLP uses a J1 cross-validation set.

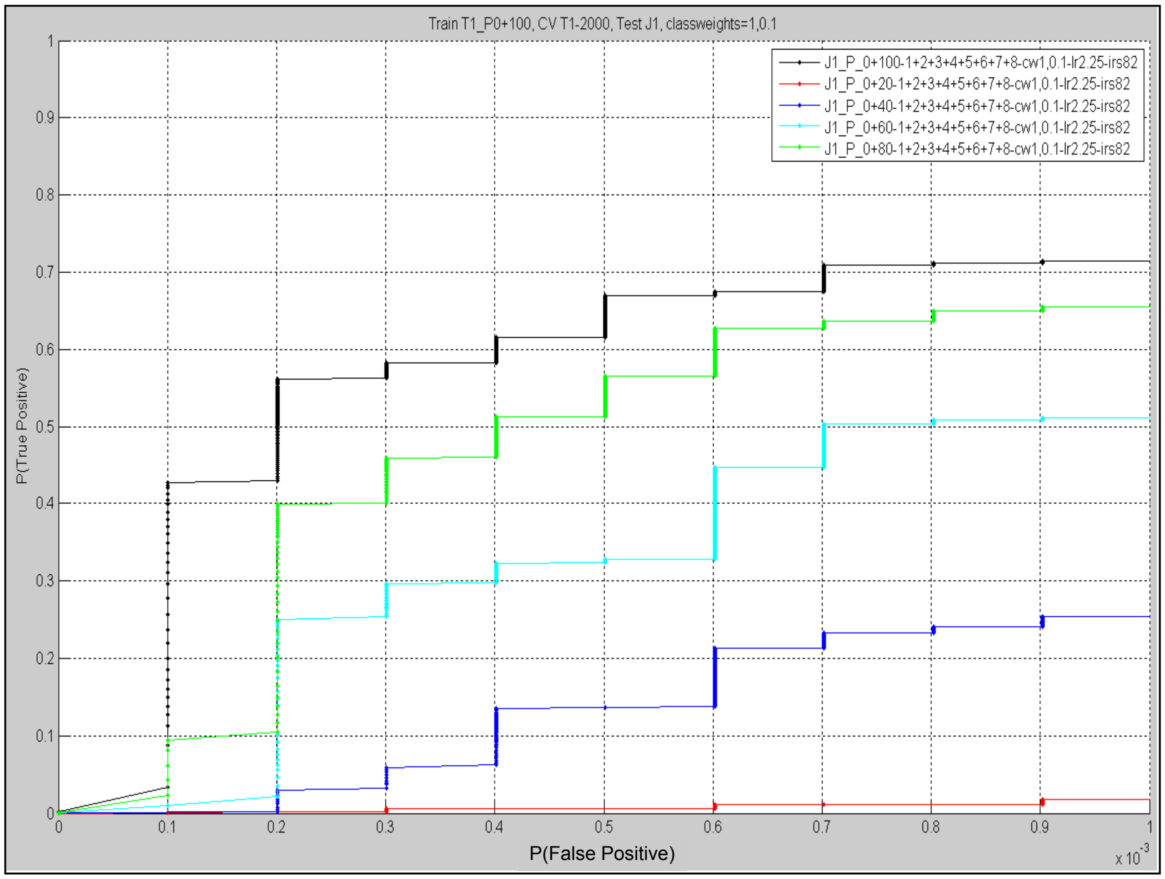


Figure 14 – ROC curves of an MLP cross-validated on 2000 samples of T1 & T1_P_100, trained on the remaining samples of T1 & T1_P_100, and tested on J1 data of different signal strengths. Note that the x-axis is scaled by 10^{-3}.

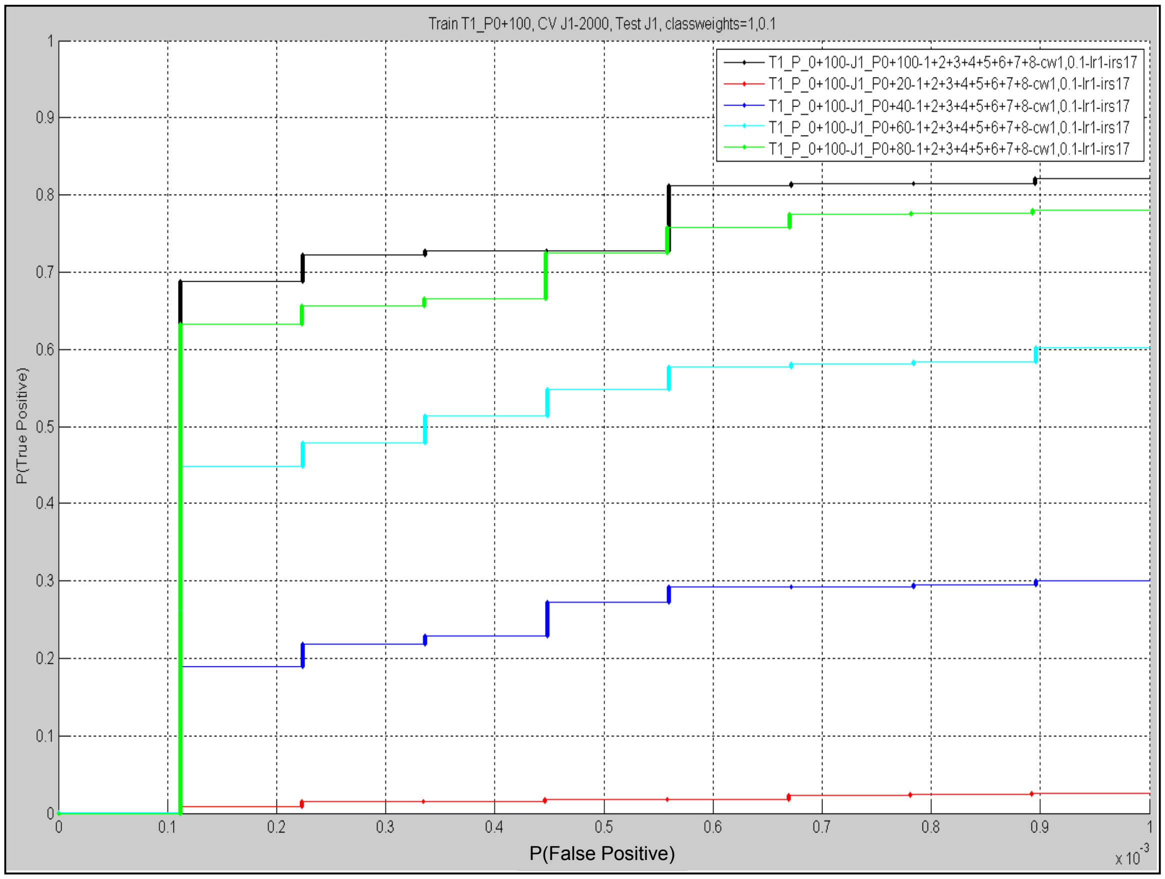


Figure 15 - ROC curves of an MLP cross-validated on 2000 samples of J1 & J1_P_100, trained on T1 & T1_P_100, and tested on remaining J1 data of different signal strengths. Note that because these ROC curves show better performance than those in Figure 14, cross-validating using J1 data leads to better performance on J1 test data. Note that the x-axis is scaled by 10^{-3}.

	ClassWeights	CV	J1_P_100	J1_P_80	J1_P_60	J1_P_40	J1_P_20
Train=T1_P0&P100 CV=T1_P0&P100 (7900 Samples)	1,1	89.33%	9.00E-05	0.00%	0.00%	2.00E-05	3.00E-05
	1,0.1	93.13%	34.12%	17.48%	5.69%	0.41%	0.07%
	10,1	89.59%	21.64%	5.10%	1.21%	0.37%	0.13%
	1,0.01	88.92%	55.71%	52.74%	41.87%	13.20%	0.72%
	100,1	88.10%	62.93%	55.70%	39.72%	7.47%	0.47%

	ClassWeights	CV	J1_P_100	J1_P_80	J1_P_60	J1_P_40	J1_P_20
Train=T1_P0&P100 CV=J1_P0&P100 (9700 Samples)	1,1	72.24%	66.95%	64.83%	49.36%	15.18%	1.46%
	1,0.1	73.29%	69.20%	66.67%	55.01%	28.89%	1.84%
	10,1	73.63%	50.73%	54.99%	36.53%	7.11%	0.42%
	1,0.01	75.28%	55.71%	58.92%	42.65%	12.08%	0.80%
	100,1	75.69%	65.04%	63.52%	52.16%	28.22%	2.98%

Table 8 - AUROC10^{-3} percentages of MLPs trained using various class-weights. The upper table reports numbers from MLPs trained with cross-validation data from 7900 samples (about 50%) of T1, while the lower table reports numbers from MLPs trained with cross-validation data from 9700 samples (about 50%) of J1. The AUROC10^{-3} percentages are not always higher than those of smaller CV sets in Table 7.

	ClassWeights	CV	J1_P_100	J1_P_80	J1_P_60	J1_P_40	J1_P_20
Train=T1_P0&P40 CV=J1_P0&P40 (2000 Samples)	1,1	46.16%	54.63%	47.26%	36.41%	15.51%	1.23%
	1,0.1	55.96%	55.53%	50.71%	42.02%	20.16%	0.79%
	10,1	47.99%	61.59%	49.80%	39.10%	14.85%	1.84%
	1,0.01	53.10%	70.65%	58.45%	42.76%	23.20%	1.37%
	100,1	54.02%	67.10%	61.15%	46.45%	16.68%	1.89%

Table 9 - AUROC10^{-3} percentages of MLPs trained on T1 & T1_P_100 and cross-validated on 2000 samples from J1 & J1_P_40 using various class-weights. For class-weights = (10,1), (1,0.01), and (100,1), these AUROC10^{-3} percentages are much better than those of bottom table in Table 7, which provides some evidence to support the hypothesis that training on a harder task (P40) leads to improvements on the easier task (P100).

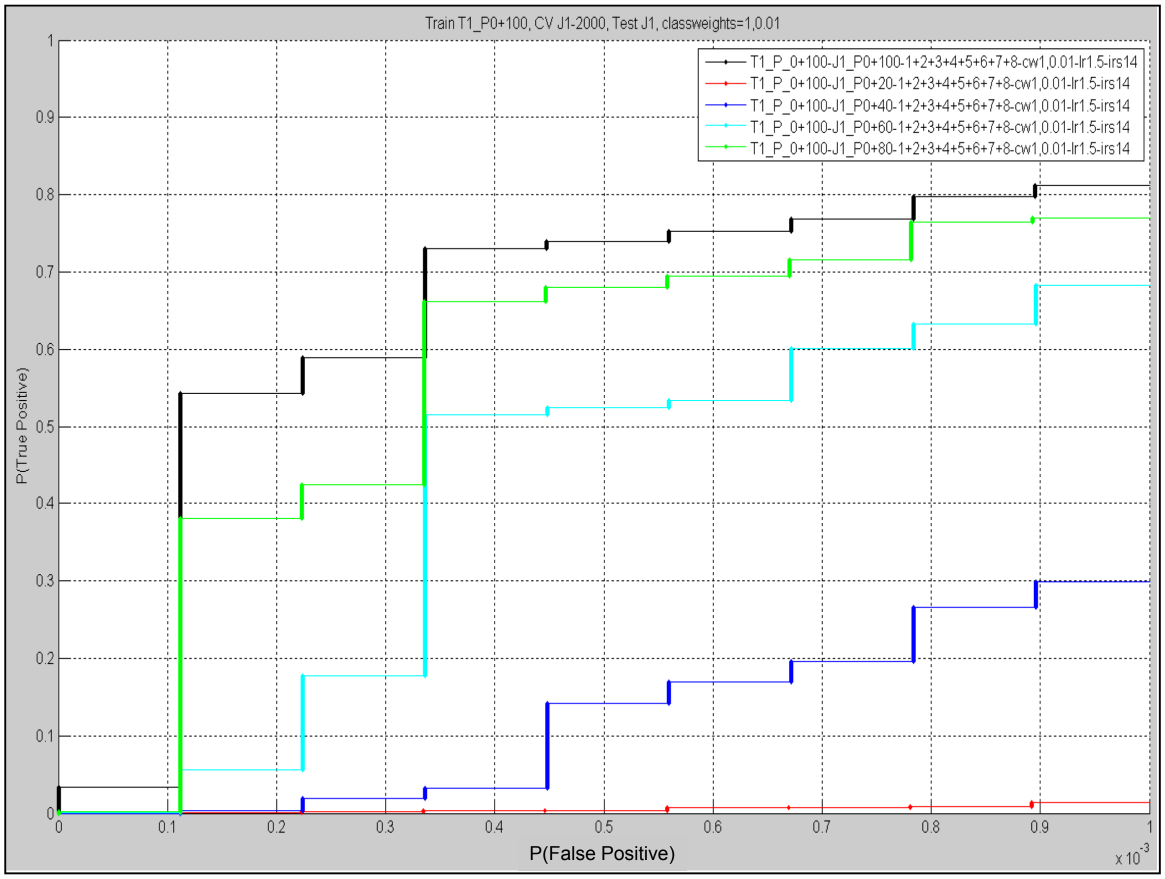


Figure 16 - ROC curves of an MLP cross-validated on 2000 samples of J1 & J1_P_100, trained on *T1 & T1_P_100*, and tested on remaining J1 data of different signal strengths. Note that the x-axis is scaled by 10^{-3}.

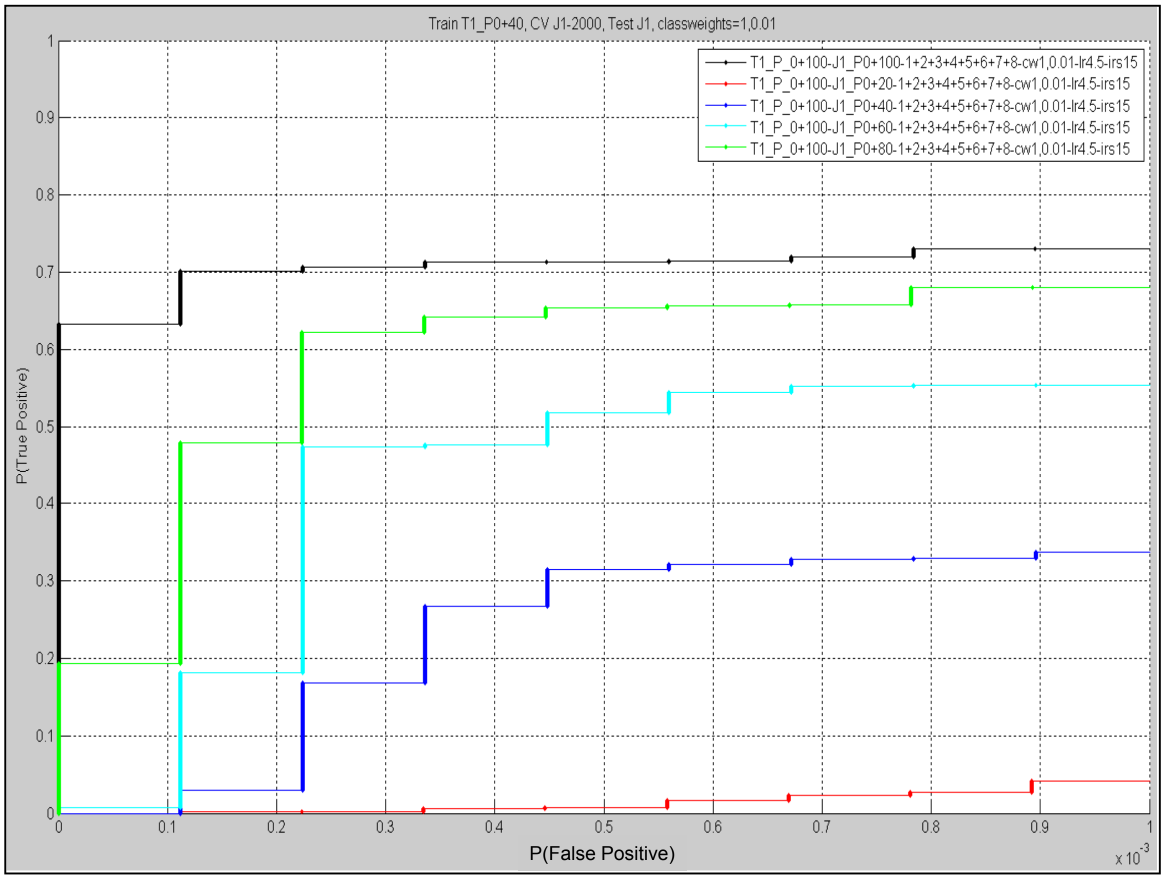


Figure 17 - ROC curves of an MLP cross-validated on 2000 samples of J1 & J1_P_100, trained on *T1 & T1_P_40*, and tested on remaining J1 data of different signal strengths. Comparing these curves to the ones in Figure 16, shows that training on harder data (e.g., T1_P_40) can sometimes lead to better performance on easier data (e.g., J1_P_100 and J1_P_80). Note that the x-axis is scaled by 10^{-3}.

Experiment 6: AUROC of SVMs Trained on T1 Sets and Tested on J1 Sets Using F1-8 and the Effect of Cross-Validation Data on Performance *(Status 04-04-2006 & Status 04-11-2006)*

Summary:

In this experiment, we vary the data used for cross-validation and investigate the effect this has on final test performance of SVMs. The main findings are:

- As in the MLP case (Experiment 5), the AUROC10^{-3} percentages on J1 data are better for SVMs trained using J1 data for cross-validation.
  - Increasing the amount of T1 data for cross-validation hurts performance on J1 test data.
- 10-fold cross-validation leads to similar performance compared with cross-validation on a single cross-validation set consisting of a disjoint 10% collection of training data.

Experimental Settings:

Training set	Remaining T1 & T1_P_100
Cross-validation set	2000 samples of J1 & J1_P_100 ("J1-2000") 2000 samples of T1 & T1_P_100 ("T1-2000") 9700 samples of T1 & T1_P_100 ("T1-9700") 10-Fold Cross Validation (i.e., CV on each of the ten subsets of T1 & T1_P_100, and train on the remaining nine subsets. "10-Fold CV on T1")
Testing set	J1, J1_P_100, J1_P_80, J1_P_60, J1_P_40, J1_P_20
Features used	F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	AUROC10^{-3}
Testing performance criteria	AUROC10^{-3}
Kernel type	Gaussian
Gaussian kernel width search range	σ=1 to 10 with increments of 1
C search range	1 to 500 with increments of 25
ν_- and ν_+ search range	Not applicable

Tables and Graphs:

	P100	P80	P60	P40	P20	AVG
SVM (Gaussian) CV J1-2000	72.58%	69.01%	59.87%	39.05%	5.17%	49.14%
SVM (Gaussian) CV T1-2000	60.67%	51.74%	34.59%	12.51%	1.12%	32.12%
SVM (Gaussian) CV T1-9700	53.92%	48.96%	39.05%	19.13%	2.61%	32.73%
SVM (Gaussian) 10-Fold CV on T1	60.00%	50.07%	31.48%	11.57%	1.27%	30.88%

Table 10 – AUROC10^{-3} percentages for C-SVC using Gaussian kernels trained on T1 & T1_P_100 with various cross-validation sets.

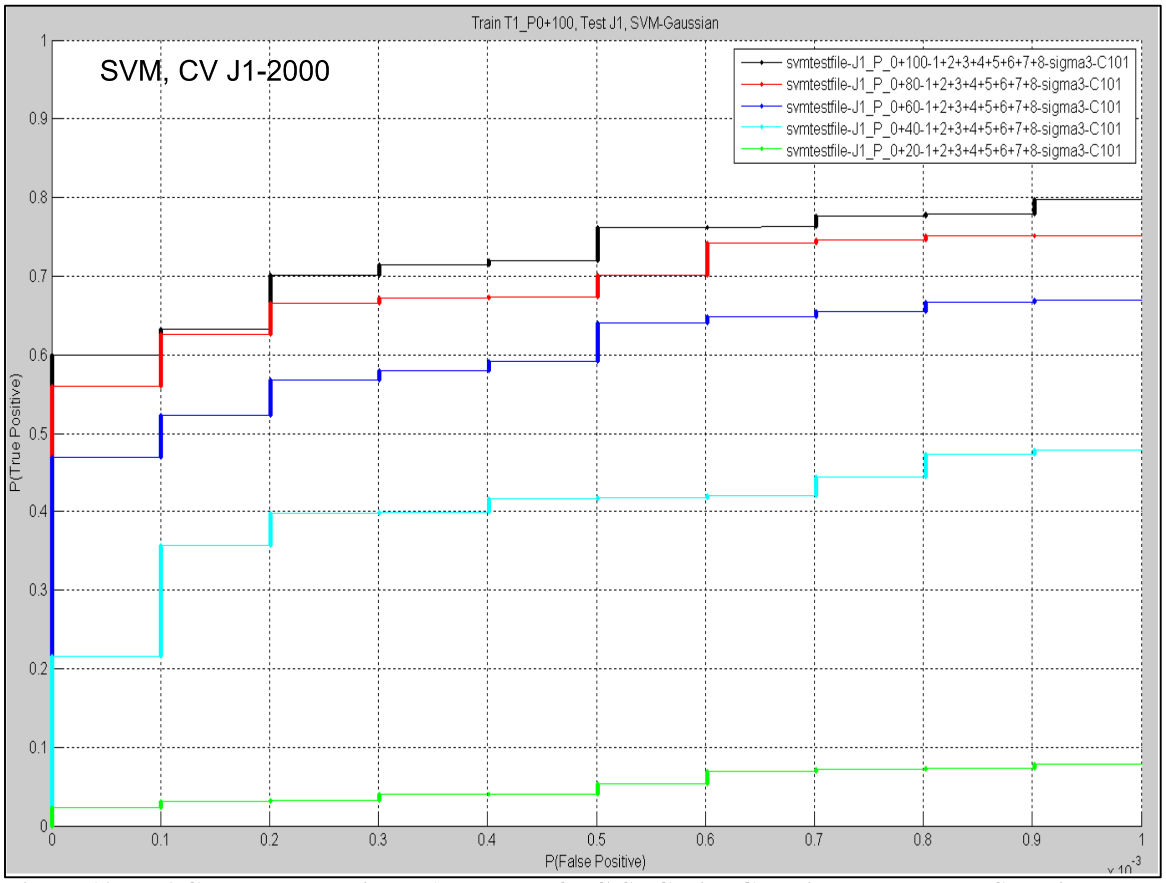


Figure 18 – ROC curves on various J1 test sets of a C-SVC with Gaussian kernel. The SVM is trained on T1 & T1_P_100 and cross-validated on 2000 samples of J1 & J1_P_100. Note that the x-axis is scaled by 10^{-3}.

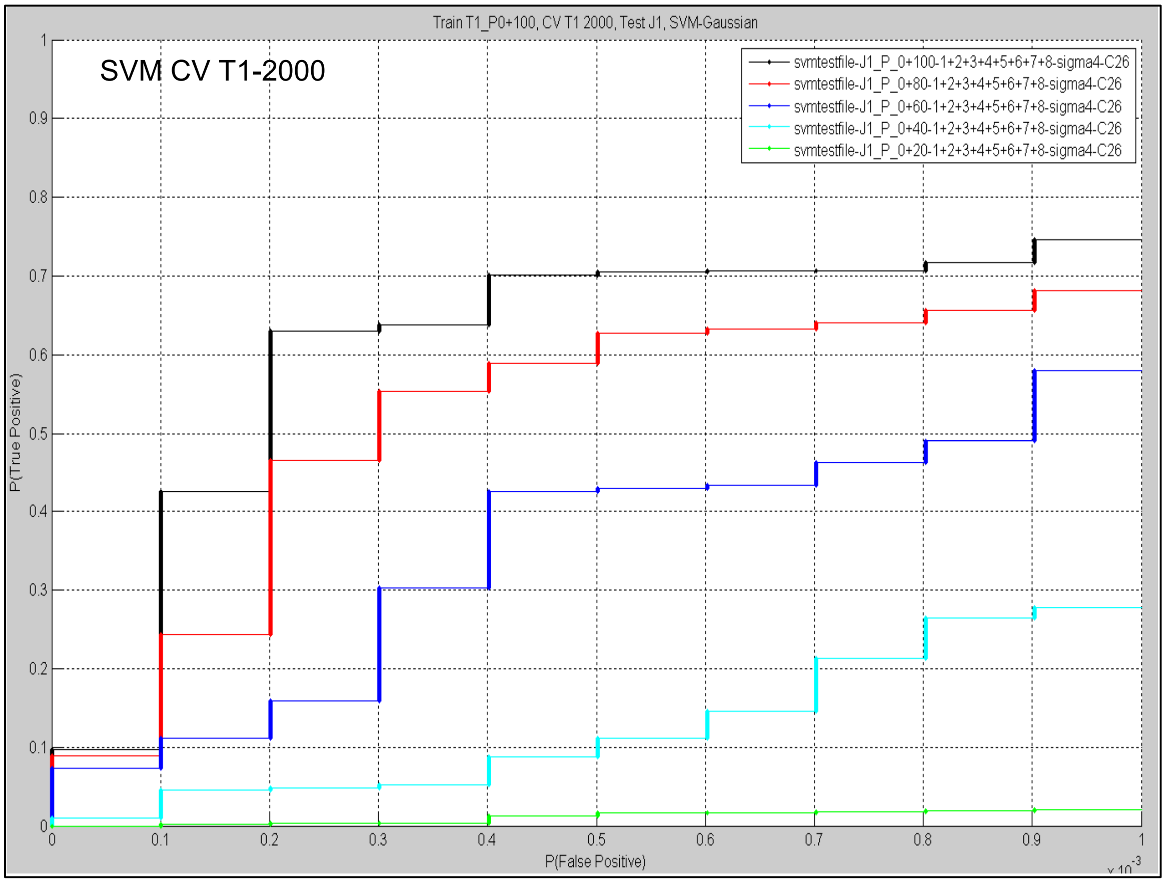


Figure 19 – ROC curves on various J1 test sets of a C-SVC with Gaussian kernel. The SVM is cross-validated on 2000 samples of T1 & T1_P_100 and trained on the remaining T1 & T1_P_100 data. Note that the x-axis is scaled by 10^{-3}.

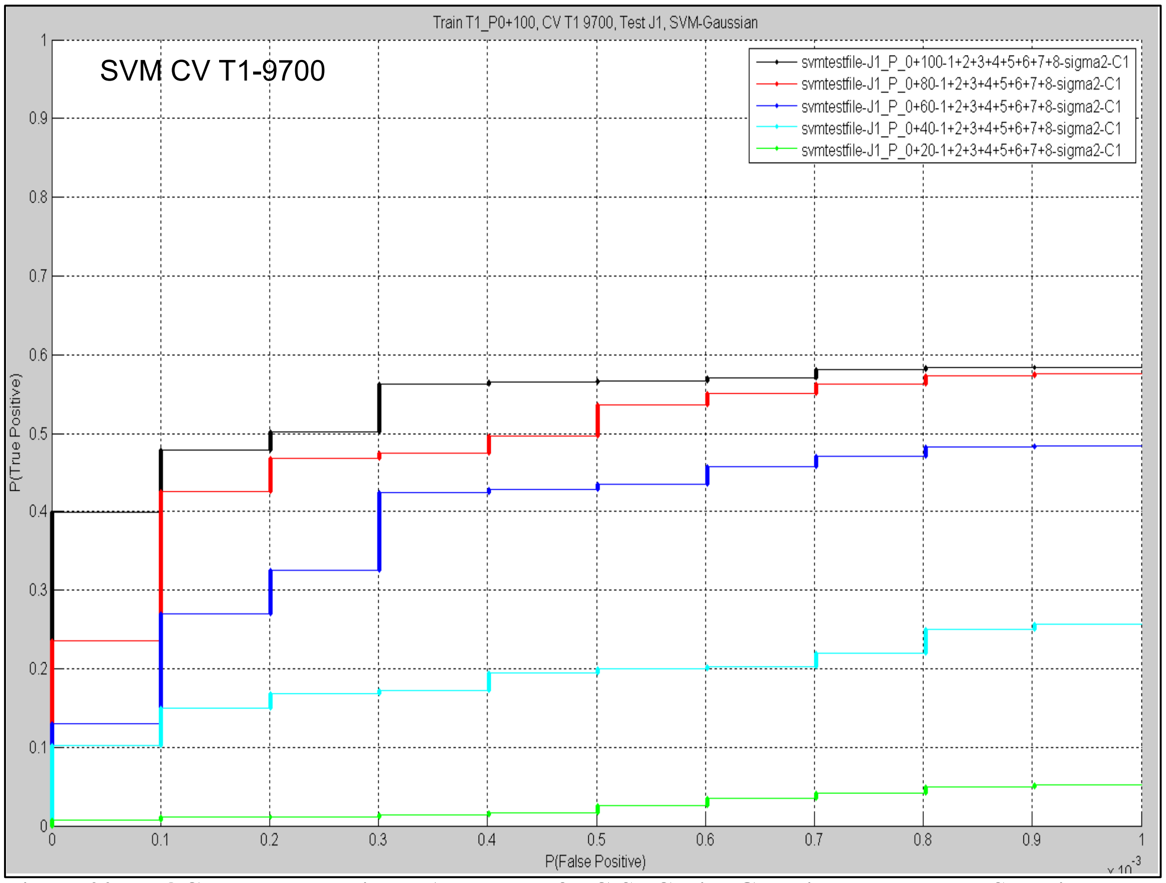


Figure 20 – ROC curves on various J1 test sets of a C-SVC with Gaussian kernel. The SVM is cross-validated on 9700 samples of T1 & T1_P_100 and trained on the remaining T1 & T1_P_100 data. Note that the x-axis is scaled by 10^{-3}.

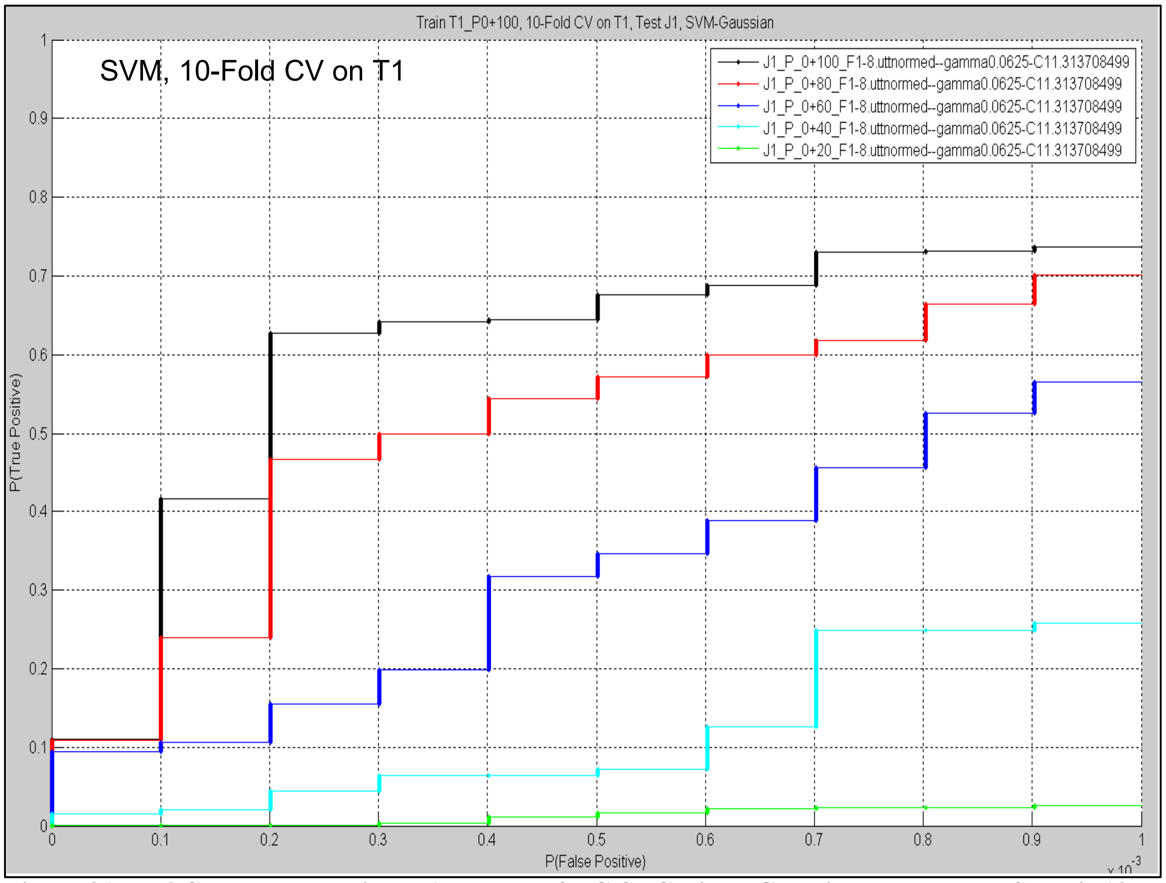


Figure 21 – ROC curves on various J1 test sets of a C-SVC with a Gaussian kernel. The SVM is 10-fold cross-validated using each of the 10% disjoint subsets of T1 & T1_P_100. The values of C and σ leading to the best average AUROC10^{-3} are used to train the final SVM on all of the T1 & T1_P_100 data. Note that the x-axis is scaled by 10^{-3}.

Experiment 7: AUROC of MLPs and SVMs Trained on T1_P_100 and Tested on J1 Sets Using All the Possible Feature Combinations *(Status 03-16-2006)*

Summary:

In this experiment, we investigate the how varying the input features affects the AUROC10^{-3} percentages in MLPs as well as SVMs. Both are trained using T1 & T1_P_100 data, cross-validated on J1 data, and tested on J1 data. Because the cross-validation sets contain J1 data, the resulting AUROC10^{-3} percentages on the J1 test set are higher than those where the cross-validation set does not contain any J1 data. The main findings are:

- SVM performance degrades less rapidly than MLP performance as signal strength decreases
- The best MLP feature combination uses 6 out of the 8 features, while the best SVM feature combination uses all 8 features.

Experimental Settings:

For MLPs:

Training set	T1 & T1_P_100
Cross-validation set	2000 Samples of J1 & J1_P_100
Testing set	J1, J1_P_100, J1_P_80, J1_P_60, J1_P_40, J1_P_20
Features used	All 255 possible combination of F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	AUROC10^{-3}
Testing performance criteria	AUROC10^{-3}
Class-weights	(1.0, 0.01)
Learning rate search range	From 0.5 to 6.0 with increments of 0.5
Number of initial random seeds	25
Number of trainable parameters	About 300

For SVMs:

Training set	T1 and T1_P_100
Cross-validation set	All of J1 & J1_P_100
Testing set	J1, J1_P_100, J1_P_80, J1_P_60, J1_P_40, J1_P_20,
Features used	All 255 possible combination of F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	AUROC10^{-3}

Testing performance criteria	AUROC10^{-3}
Kernel type	Gaussian
Gaussian kernel width search range	σ=1 to 5 with increments of 1
C search range	1 to 300 with increments of 25
ν_- and ν_+ search range	Not applicable

<u>Tables and Graphs:</u>

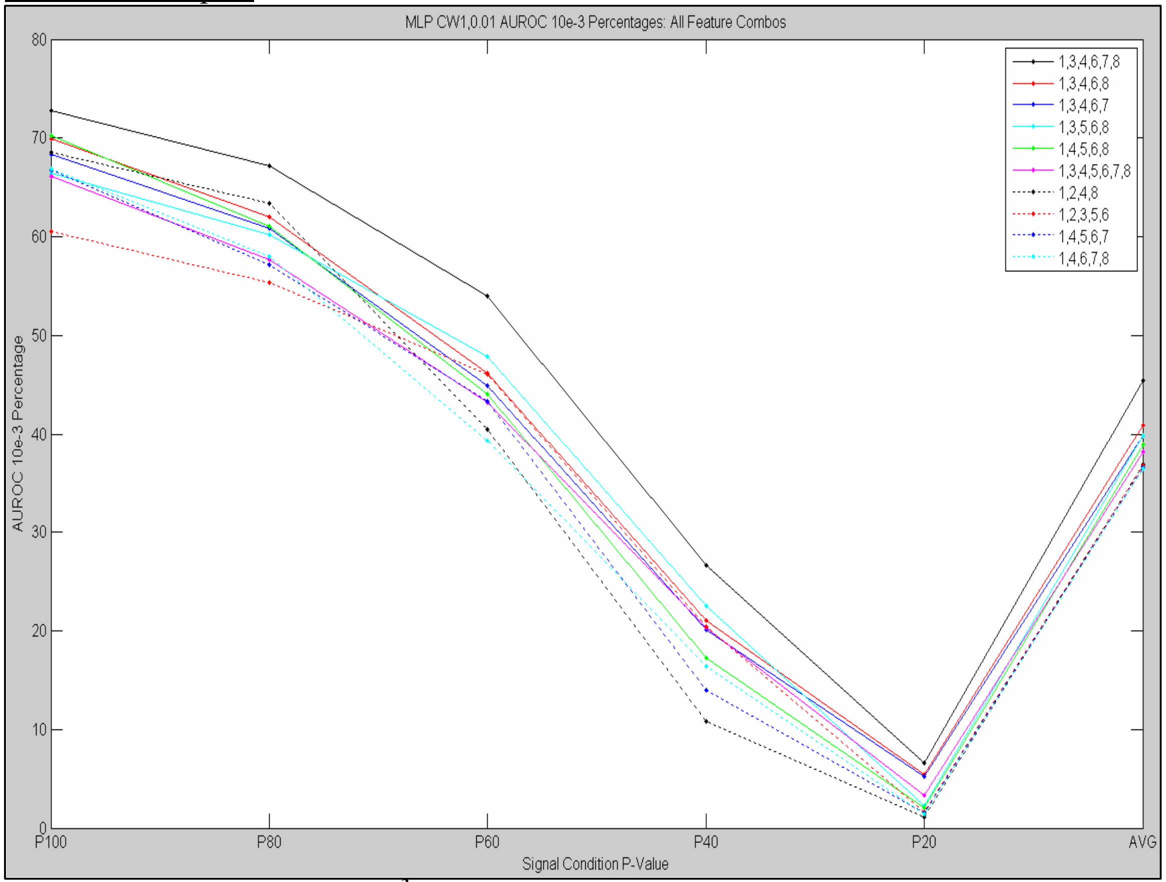


Figure 22 – The top-10 AUROC10^{-3} percentages versus J1 signal strength for MLPs trained with class-weights (1, 0.01) on T1 & T1_P_100.

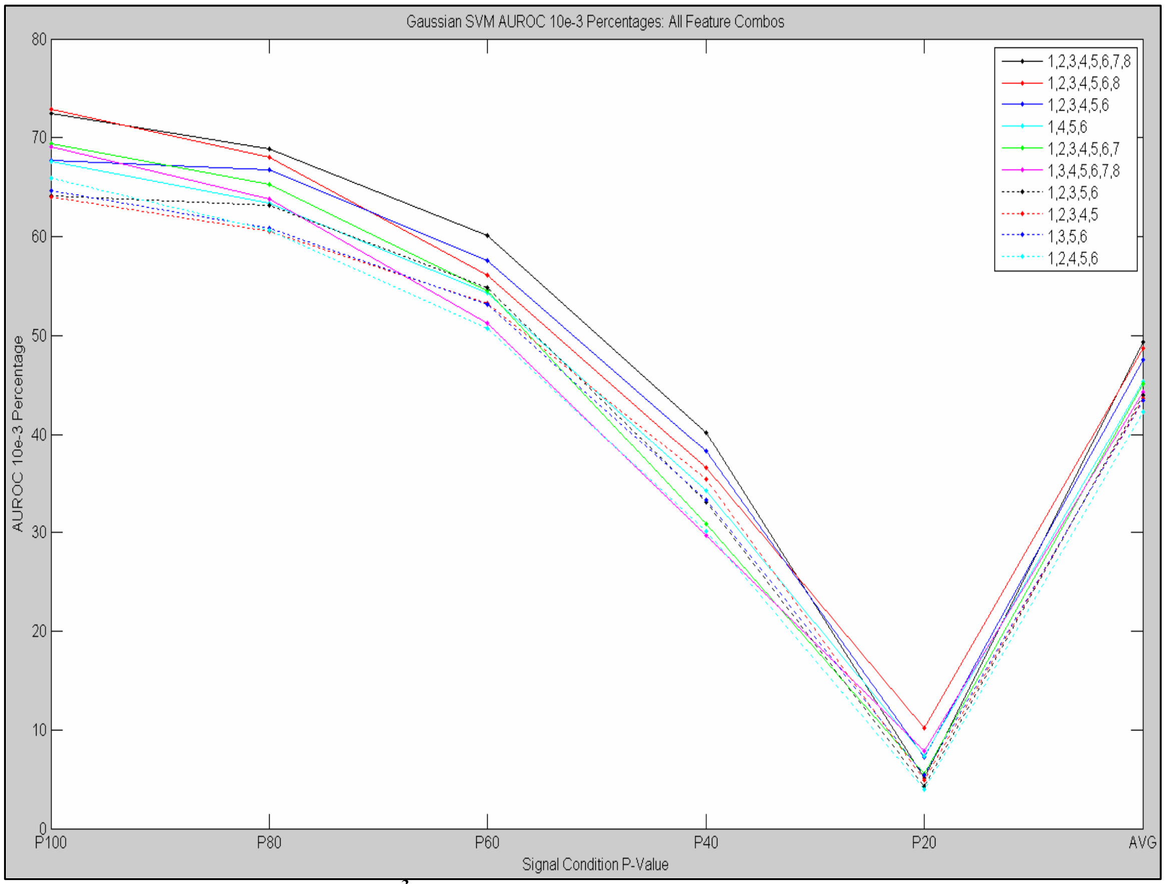


Figure 23 – The top-10 AUROC10^{-3} percentages versus J1 signal strength for SVMs trained on T1 & T1_P_100.

Refer to Status 3-16-2006 for more top-10 AUROC10^{-3} percentages versus J1 signal strength graphs of MLPs and SVMs.

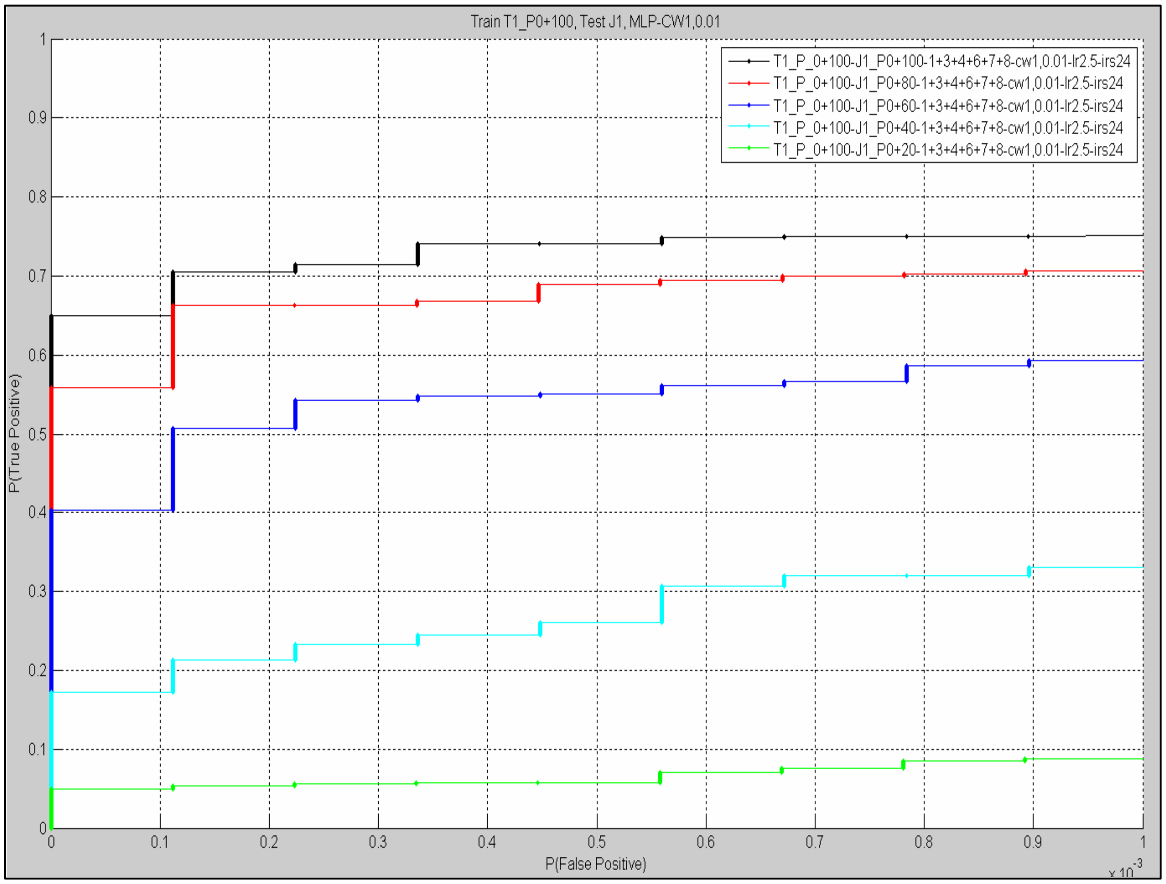


Figure 24 – ROC curves for the best performing MLP feature combination F1+F3+F4+F6+F7+F8. Note that the x-axis is scaled by 10^{-3}.

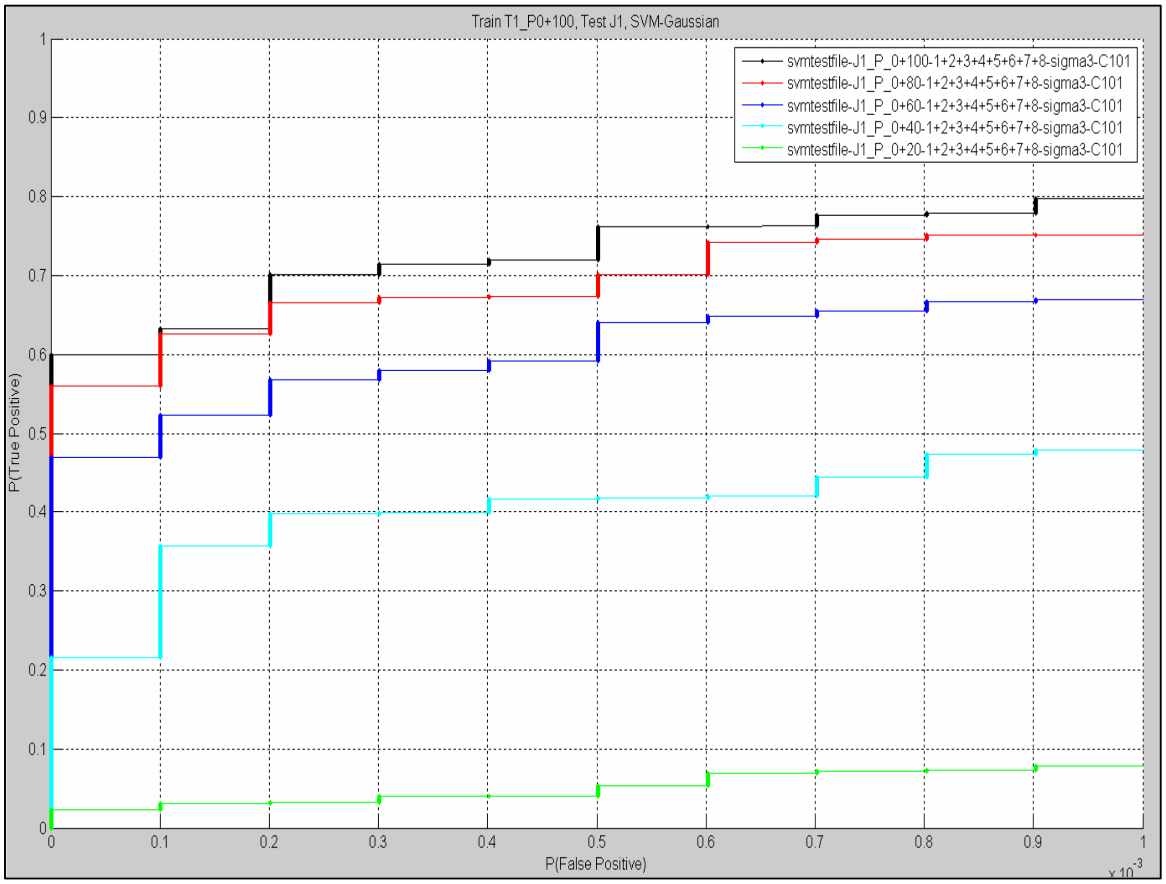


Figure 25 – ROC curves for the best performing SVM feature combination F1+F2+F3+F4+F5+F6+F7+F8. Note that the x-axis is scaled by 10^{-3}.

Experiment 8: AUROC of SVMs Trained on T0 and T1 Tested on J1 and J2 Using F1-8*(Status 04-18-2006 and Status 04-28-2006)*

Summary:

In this experiment, we investigate the effect that training on T0 or T1 has on the AUROC10^{-3} percentages measured on J1 and J2 test data. We also investigate the effect of feature normalization (zero mean unity variance normalization versus no normalization) has on performance. The main findings are:

- Zero mean and unity variance normalization leads to much better results.
- With normalization:
  - When testing on J1, training with T1 leads to better AUROC10^{-3} than with T2.
  - When testing on J2, training *at the P_100 level* with T1 leads to better AUROC10^{-3} than with T2, but training *at the P_40 level* with T2 is better than T1.
- Without normalization:
  - Training with T1 almost always leads to poorer AUROC10^{-3} than training with T2 except when training and testing at the highest signal levels (P_100 and P80).

Experimental Settings:

Training set	T1 & T1_P_100 T1 & T1_P_40 T0 & T0_P_100 T0 & T0_P_40
Cross-validation set	10-fold cross-validation using training set
Testing set	J1, J1_P_100, J1_P_80, J1_P_60, J1_P_40, J1_P_20 J2, J2_P_100, J2_P_80, J2_P_60, J2_P_40, J2_P_20
Features used	F1-F8
Feature normalization	Zero mean and unit variance for each feature or Unnormalized
Cross-validation performance criteria	AUROC10^{-3}
Testing performance criteria	AUROC10^{-3}
Kernel type	Gaussian
Gaussian kernel width search range	Depends on Normalization, Zero mean: $\log_2\gamma$=-7 to 1 with increments of $\log_2 1$ Unnormalized: $\log_2\gamma$=-7 to 4 with increments of $\log_2 1$ (where $\sigma = \frac{1}{\sqrt{\gamma}}$)
C search range	Depends on Normalization, Zero mean: $\log_2 C$=-2 to 10 with increments of $\log_2 1$ Unnormalized: $\log_2 C$=-5 to 10 with increments of $\log_2 1$
ν_- and ν_+ search range	Not applicable

Tables and Graphs:

AUROC 10e-3

	P100	P80	P60	P40	P20	AVG
Train T1_P100, Test J1	59.25%	49.40%	31.82%	12.13%	1.10%	30.74%
Train T0_P100, Test J1	33.59%	16.79%	5.82%	1.42%	0.17%	11.56%
Train T1_P100, Test J2	56.14%	51.56%	40.19%	19.82%	2.13%	33.97%
Train T0_P100, Test J2	11.44%	5.45%	2.78%	1.23%	0.38%	4.25%
Train T1_P40, Test J1	59.09%	57.46%	52.04%	37.49%	7.14%	42.65%
Train T0_P40, Test J1	61.96%	51.74%	35.72%	13.13%	1.33%	32.77%
Train T1_P40, Test J2	38.34%	34.46%	27.07%	18.10%	4.84%	24.56%
Train T0_P40, Test J2	48.56%	45.94%	39.30%	21.30%	1.20%	31.26%

Table 11 – Table of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J1 or J2 at various signal strengths. Features are normalized to zero mean and unity variance.

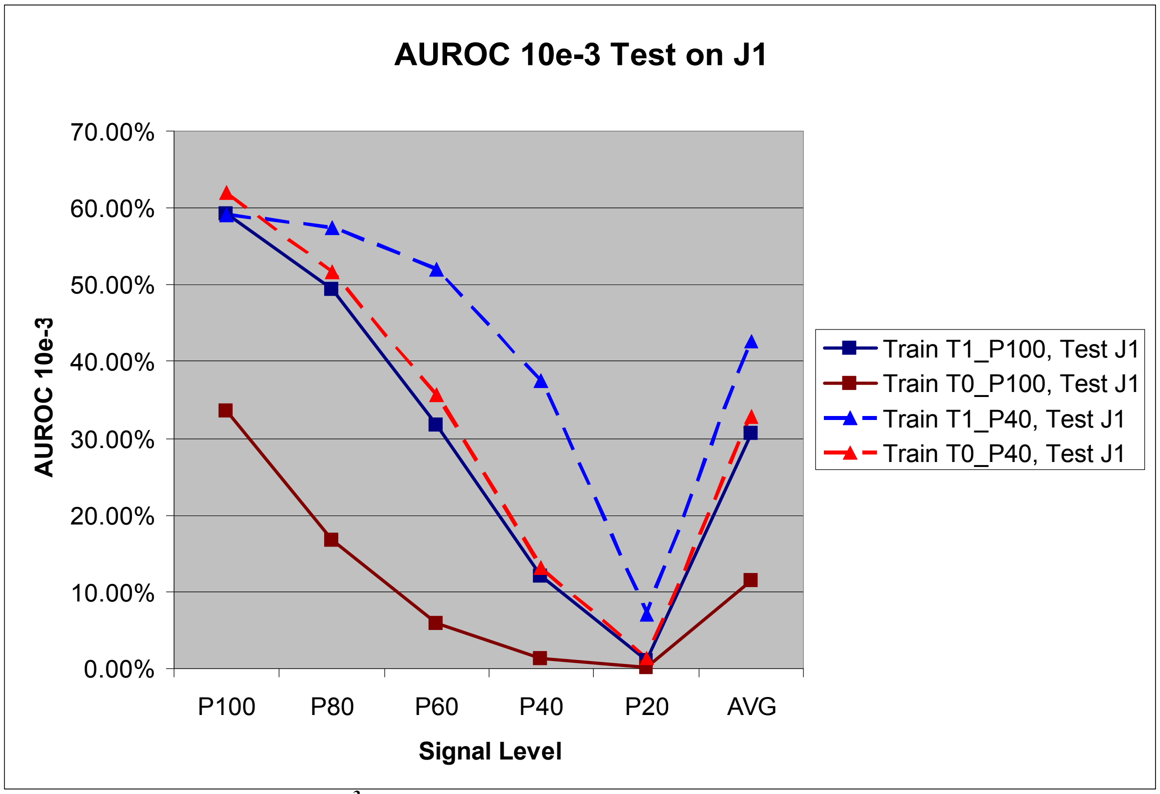


Figure 26 – Plot of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J1 at various signal strengths. Features are normalized to zero mean and unity variance.

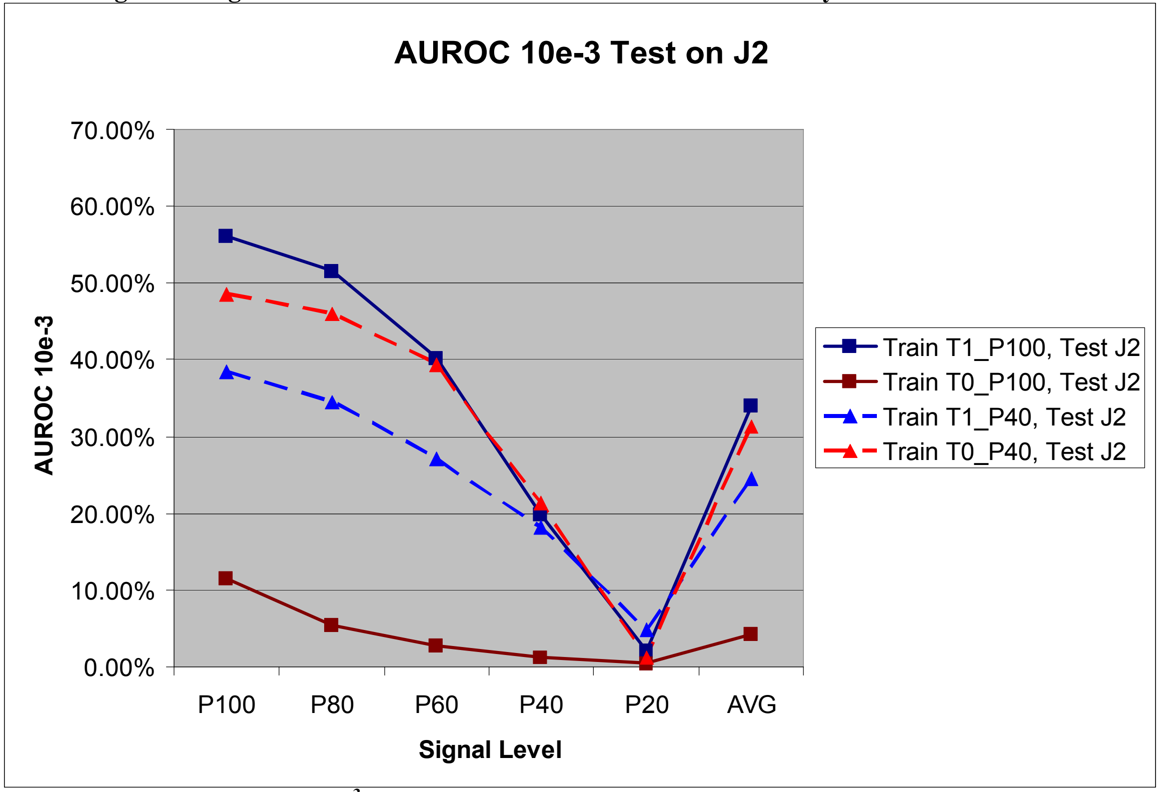


Figure 27 – Plot of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J2 at various signal strengths. Features are normalized to zero mean and unity variance.

	P100	P80	P60	P40	P20	AVG
Train T1_P100, Test J1	52.00%	45.58%	24.04%	3.71%	0.33%	25.13%
Train T0_P100, Test J1	49.77%	46.68%	31.04%	9.79%	1.33%	27.72%
Train T1_P100, Test J2	47.93%	41.37%	26.65%	4.45%	0.57%	24.19%
Train T0_P100, Test J2	39.80%	38.70%	28.97%	11.88%	2.07%	24.28%
Train T1_P40, Test J1	16.15%	30.72%	42.58%	35.10%	6.24%	26.16%
Train T0_P40, Test J1	29.17%	42.71%	46.56%	34.43%	8.29%	32.23%
Train T1_P40, Test J2	8.91%	17.73%	25.30%	19.10%	1.07%	14.42%
Train T0_P40, Test J2	18.57%	27.70%	30.33%	20.08%	1.72%	19.68%

Table 12 – Table of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J1 or J2 at various signal strengths. Features are not normalized.

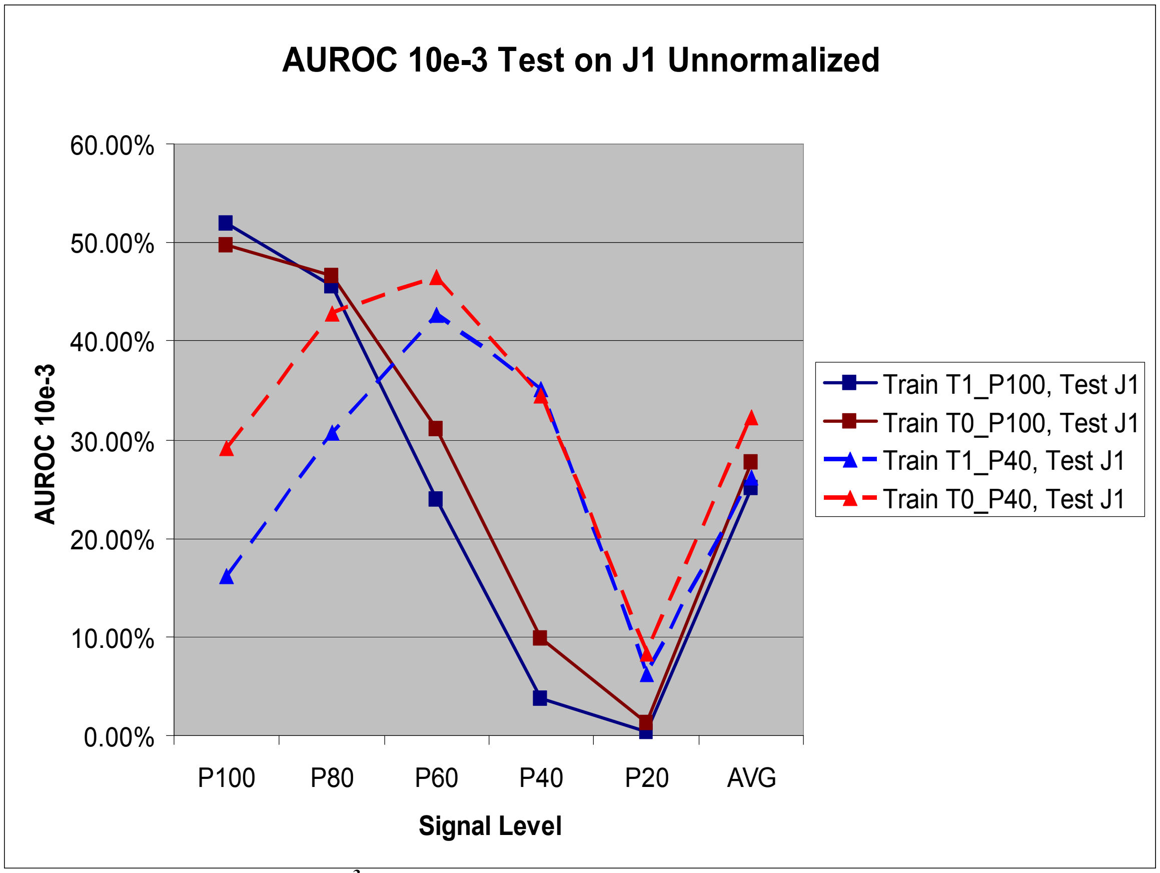


Figure 28 – Plot of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J1 at various signal strengths. Features are not normalized.

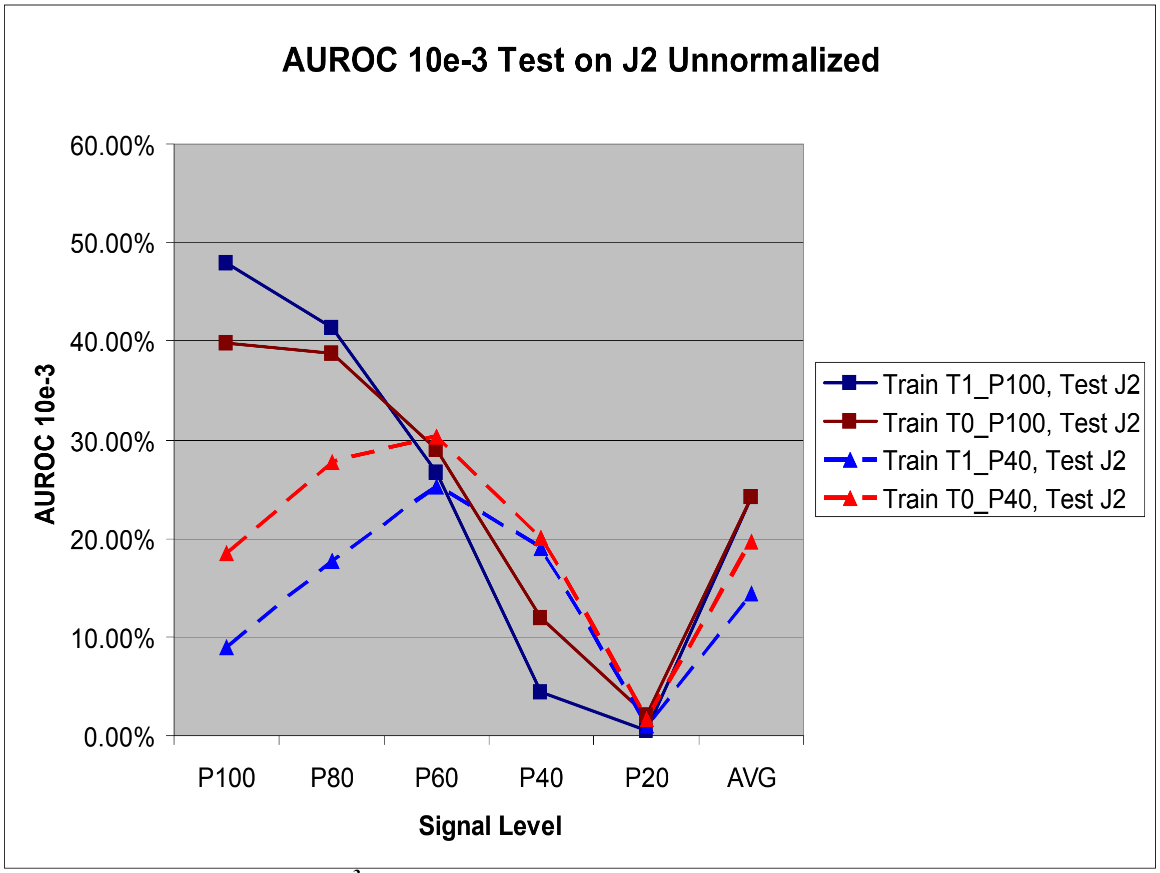


Figure 29 – Plot of AUROC10^{-3} percentages for C-SVCs trained on T1 or T2 data and tested on J1 at various signal strengths. Features are not normalized.

Experiment 9: AUROC of Cost-Sensitive SVMs Trained on T1 and T0 Tested on J1 and J2 ***(Status 05-10-2006 & Status 05-26-2006)***

Summary:

In this experiment, we compare the performance on J1 and J2 of the cost-sensitive 2Nu-SVC with the non-cost-sensitive C-SVC trained on T1 or T0 data at the P_100 level. The main findings are:

- 2Nu-SVC training is much more time consuming due to extra grid search dimension as well as the slower individual execution time compared to the C-SVC.
- 2Nu-SVCs achieve higher AUROC10^{-3} at all signal levels except for P_20 of J1 and J2 than comparable C-SVCs when trained on T1_P_100 data or T0_P_100 data.

Experimental Settings:

Training set	T1 & T1_P_100 T0 & T0_P_100
Cross-validation set	5-fold cross-validation using training set
Testing set	J1, J1_P_100, J1_P_80, J1_P_60, J1_P_40, J1_P_20 J2, J2_P_100, J2_P_80, J2_P_60, J2_P_40, J2_P_20
Features used	F1-F8
Feature normalization	Zero mean and unit variance for each feature
Cross-validation performance criteria	AUROC10^{-3}
Testing performance criteria	AUROC10^{-3}
Kernel type	Gaussian
Gaussian kernel width search range	$\log_2\gamma$=-5 to 1 with increments of $\log_2 3$ (where $\sigma = \frac{1}{\sqrt{\gamma}}$)
C search range	For the C-SVCs: $\log_2 C$=-2 to 10 with increments of $\log_2 1$
ν_- and ν_+ search range	For the 2Nu-SVCs: Uniform grid from 0.1 to 0.9 with increments of 0.1 for both ν_- and ν_+

Tables and Graphs:

	P100	P80	P60	P40	P20	AVG
C-SVC Train T1_P100, Test J1	49.40%	42.17%	28.58%	11.71%	0.94%	26.56%
2Nu-SVC Train T1_P100, Test J1	63.87%	57.28%	42.08%	12.50%	0.42%	35.23%
C-SVC Train T1_P100, Test J2	36.66%	24.58%	16.07%	6.80%	2.42%	17.30%
2Nu-SVC Train T1_P100, Test J2	52.92%	41.29%	25.53%	10.54%	1.89%	26.43%
C-SVC Train T0_P100, Test J1	33.59%	16.79%	5.82%	1.42%	0.17%	11.56%
2Nu-SVC Train T0_P100, Test J1	52.39%	39.90%	16.59%	1.50%	0.15%	22.11%
C-SVC Train T0_P100, Test J2	11.44%	5.45%	2.78%	1.23%	0.38%	4.25%
2Nu-SVC Train T0_P100, Test J2	45.06%	39.62%	26.74%	7.55%	1.00%	23.99%

Table 13 – AUROC10^{-3} percentages on J1 and J2 test sets at various signal strengths of C-SVCs and 2Nu-SVCs trained on T1 & T1_P_100 and T0 & T0_P_100 data. Each 2Nu-SVC outperforms its C-SVC counterpart for every signal strength except the P_20 level.

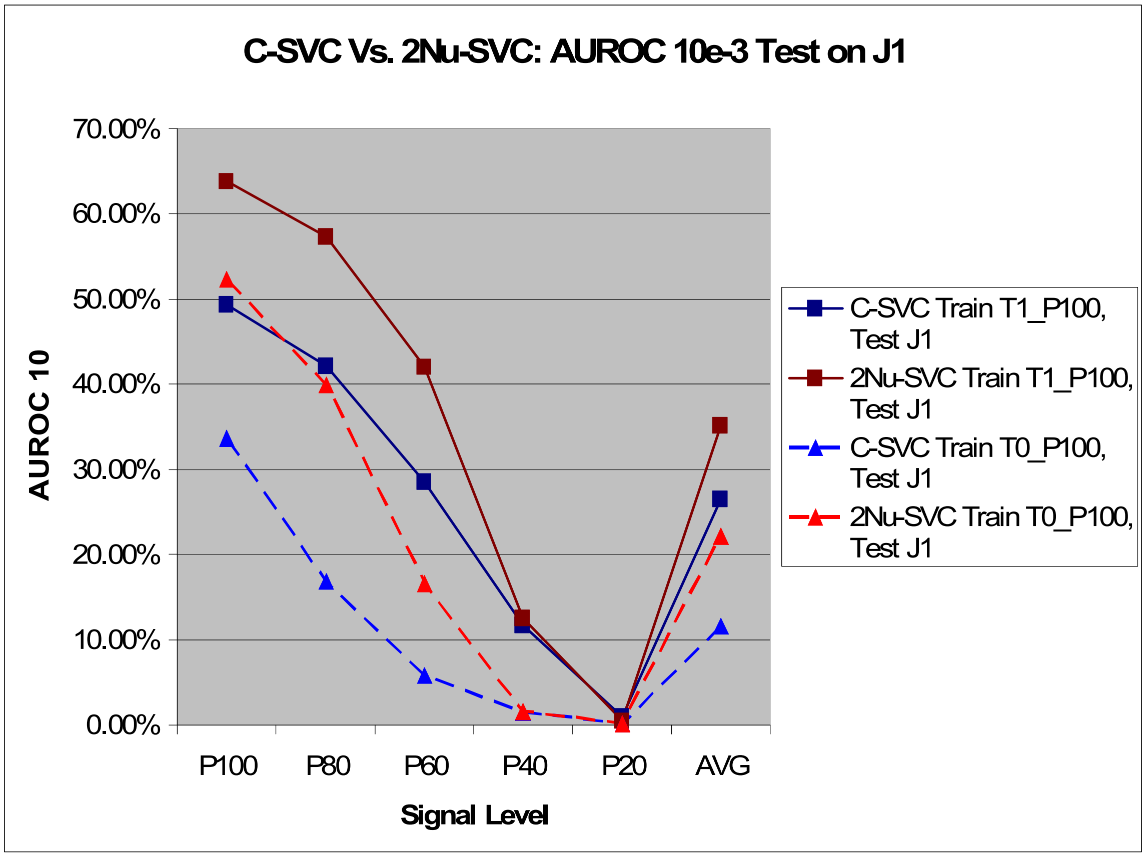


Figure 30 – Graph of AUROC10^{-3} percentages on the J1 test set at various signal strengths of C-SVCs and 2Nu-SVCs trained on T1 & T1_P_100 or T0 & T0_P_100 data. Each 2Nu-SVC outperforms its C-SVC counterpart for every signal strength except the P_20 level.

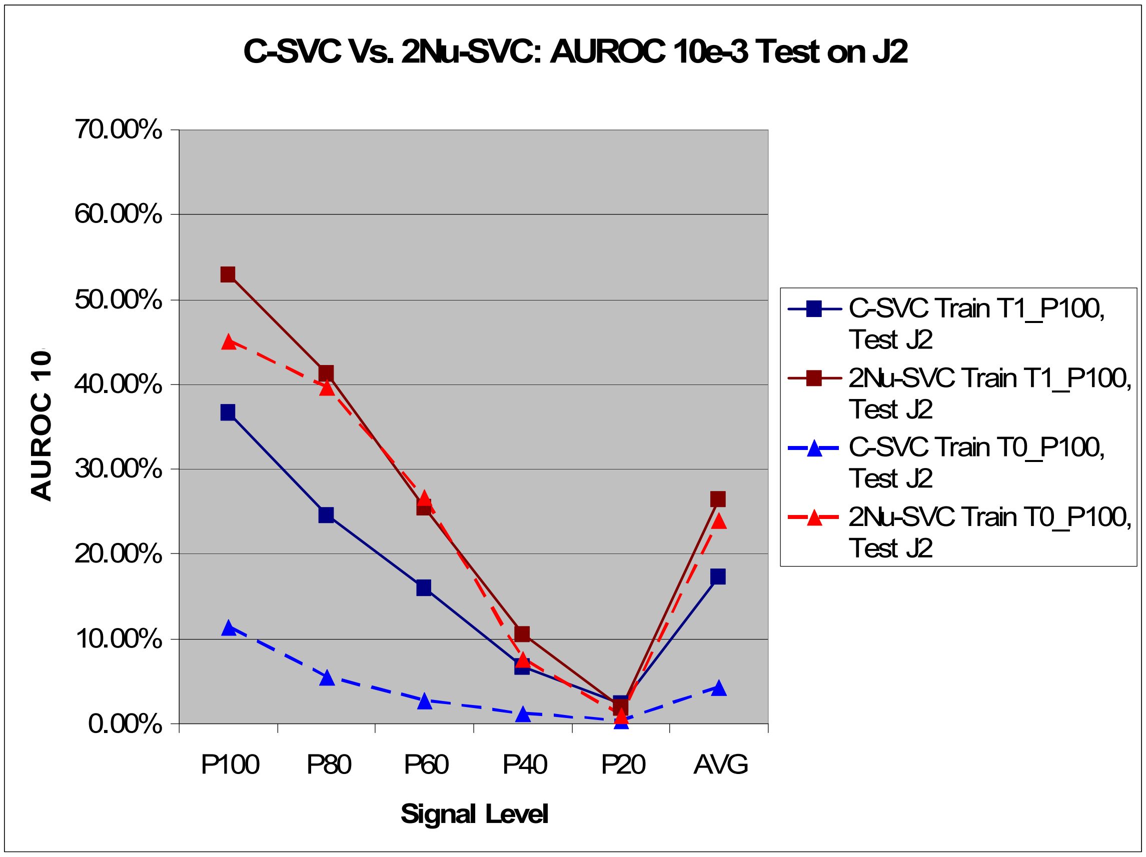


Figure 31 – Graph of AUROC10^{-3} percentages on the J2 test set at various signal strengths of C-SVCs and 2Nu-SVCs trained on T1 & T1_P_100 or T0 & T0_P_100 data. Each 2Nu-SVC outperforms its C-SVC counterpart for every signal strength except the P_20 level.

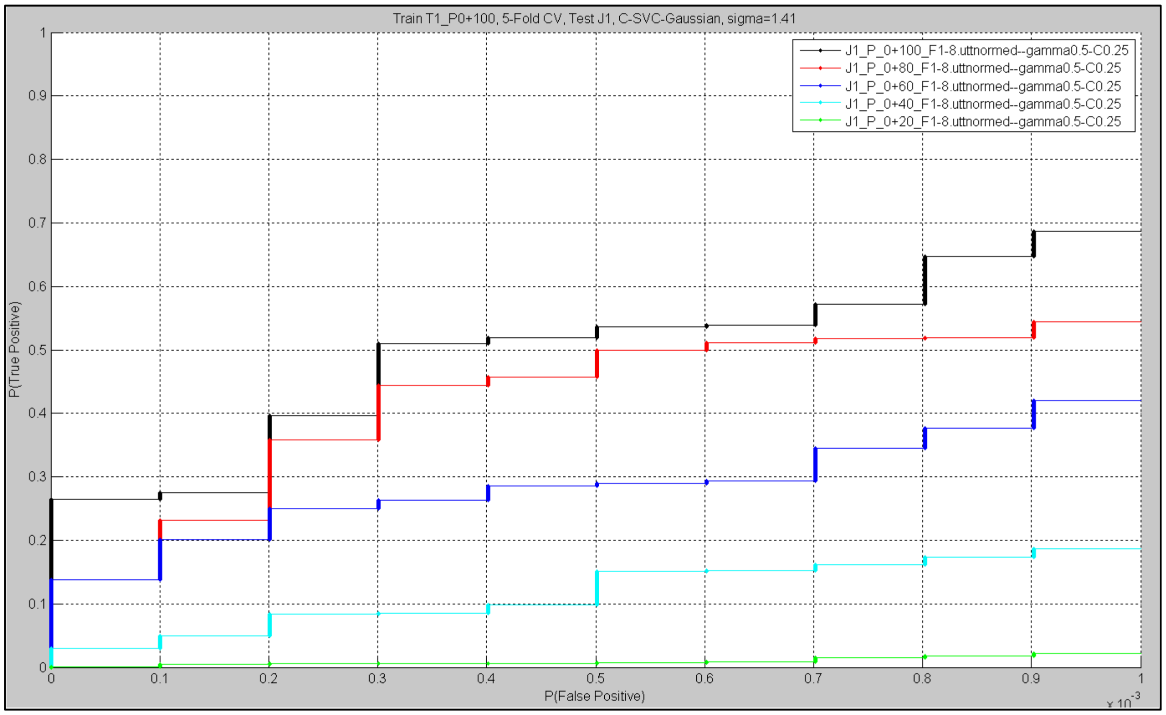


Figure 32 – ROC curves on J1 test data of the C-SVC (C=0.25 and σ=1.41) trained and 5-fold cross-validated on T1 & T1_P_100. Note that the x-axis is scaled by 10^{-3}.

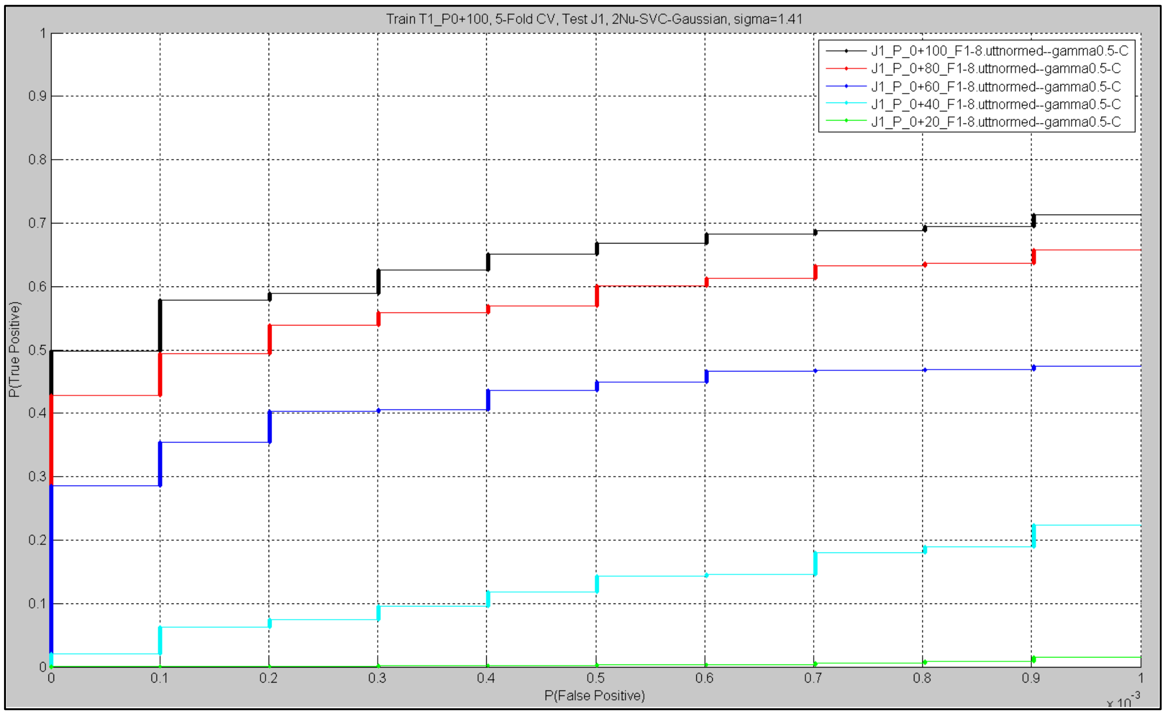


Figure 33 – ROC curves on J1 test data of the 2Nu-SVC (ν_-=0.021,ν_+=0.071, and σ=1.41,) trained and 5-fold cross-validated on T1 & T1_P_100. Note that the x-axis is scaled by 10^{-3}.

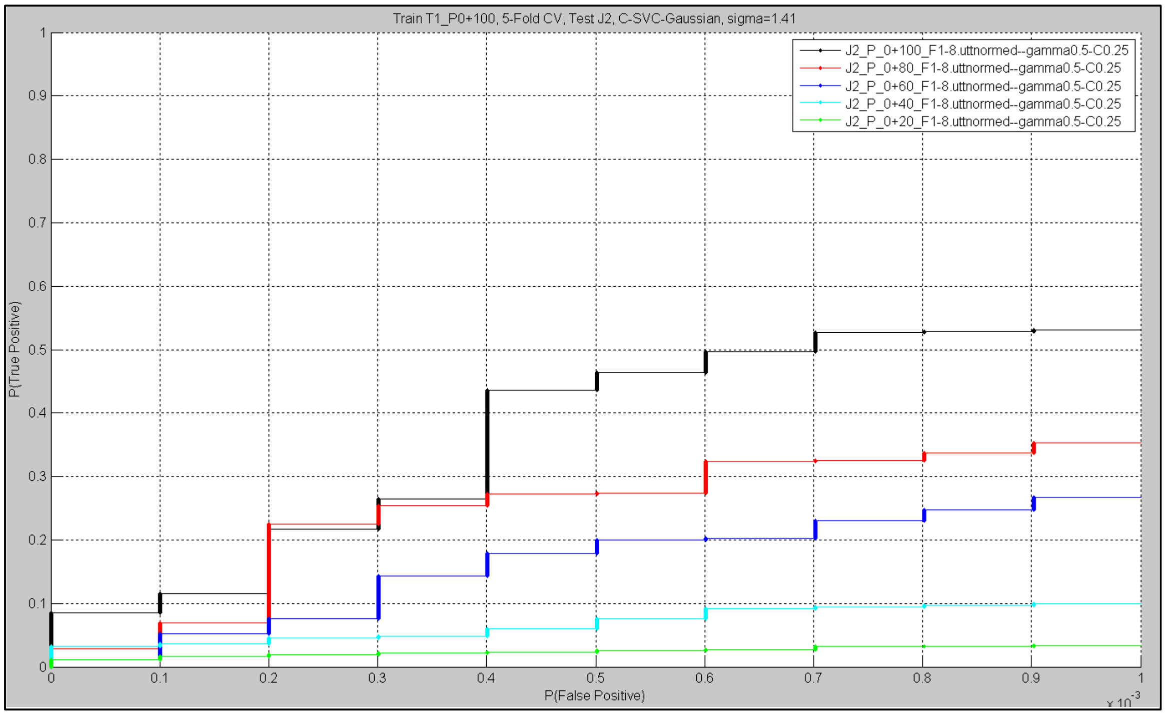


Figure 34 – ROC curves on J2 test data of the C-SVC (C=0.25 and σ=1.41) trained and 5-fold cross-validated on T1 & T1_P_100. Note that the x-axis is scaled by 10^{-3}.

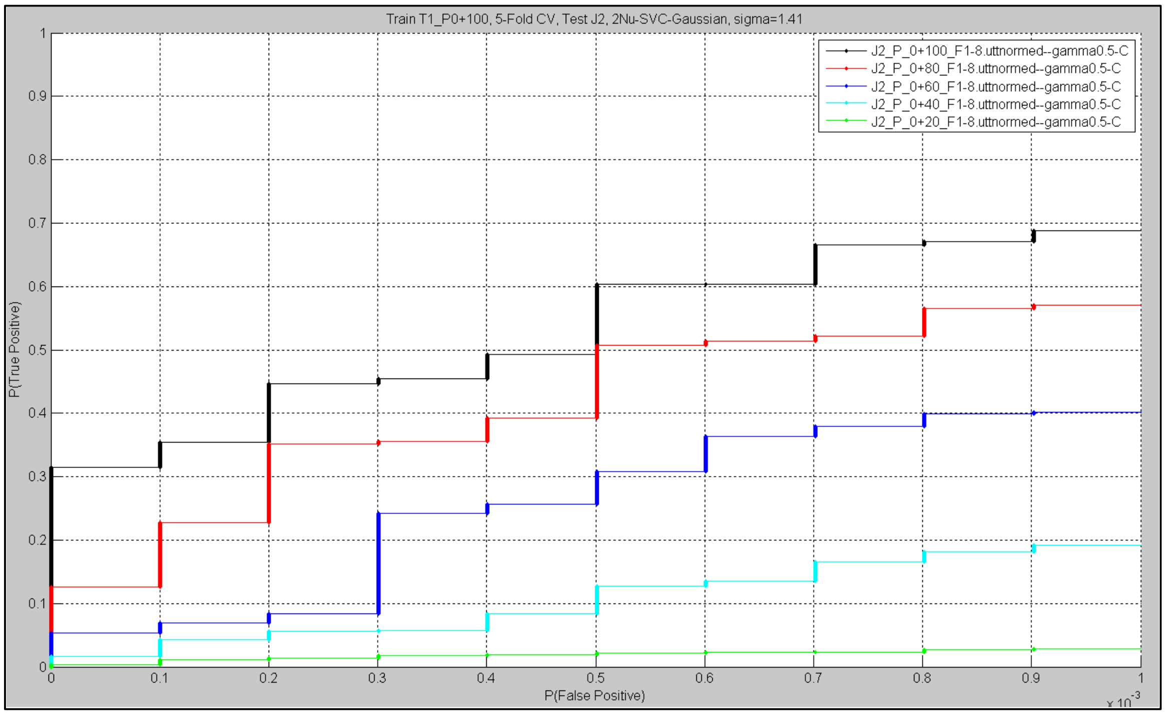


Figure 35 – ROC curves on J2 test data of the 2Nu-SVC (ν_-=0.021,ν_+=0.071, and σ=1.41,) trained and 5-fold cross-validated on T1 & T1_P_100. Note that the x-axis is scaled by 10^{-3}.

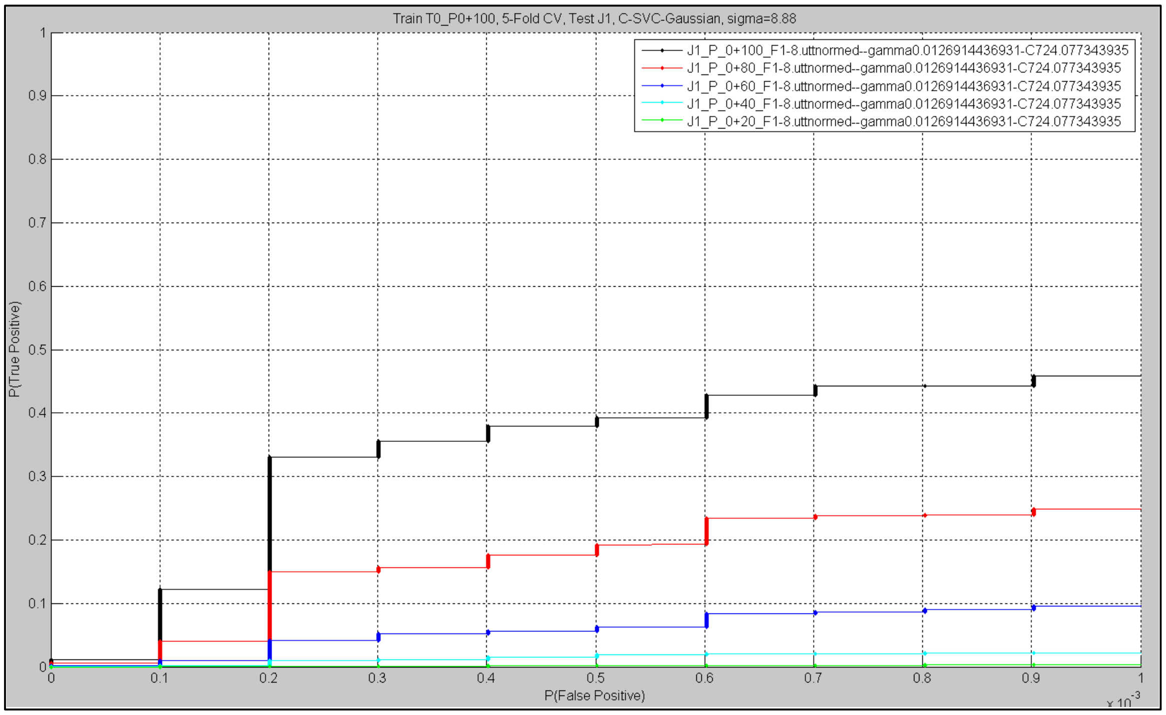


Figure 36 – ROC curves on J1 test data of the C-SVC (C=724.1 and σ=8.88) trained and 5-fold cross-validated on T0 & T0_P_100. Note that the x-axis is scaled by 10^{-3}.

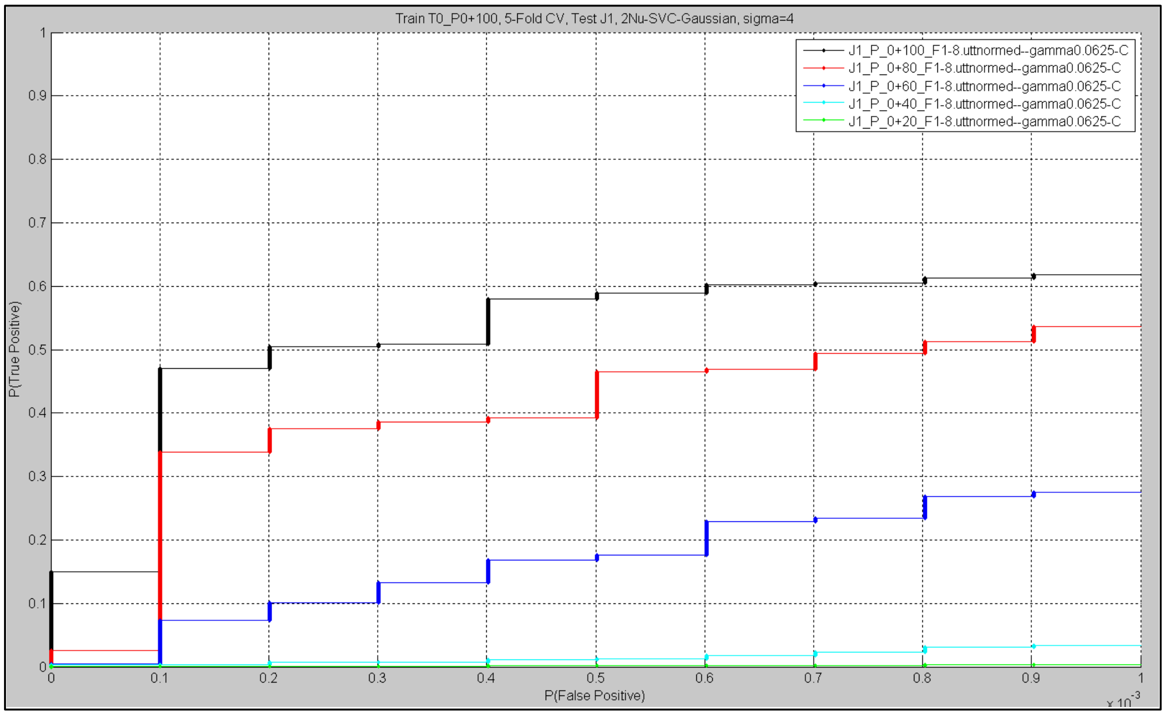


Figure 37 – ROC curves on J1 test data of the 2Nu-SVC (ν_-=0.009, ν_+=0.005, and σ=4,) trained and 5-fold cross-validated on T0 & T0_P_100. Note that the x-axis is scaled by 10^{-3}.

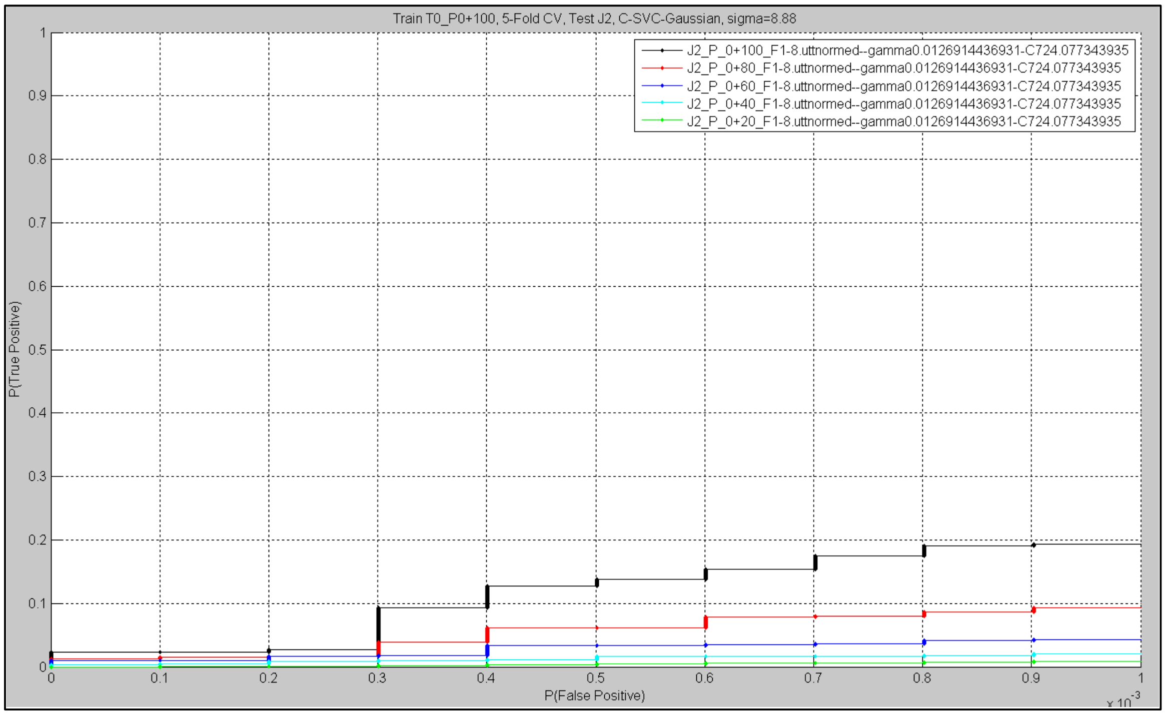


Figure 38 – ROC curves on J2 test data of the C-SVC (C=724.1 and σ=8.88) trained and 5-fold cross-validated on T0 & T0_P_100. Note that the x-axis is scaled by 10^{-3}.

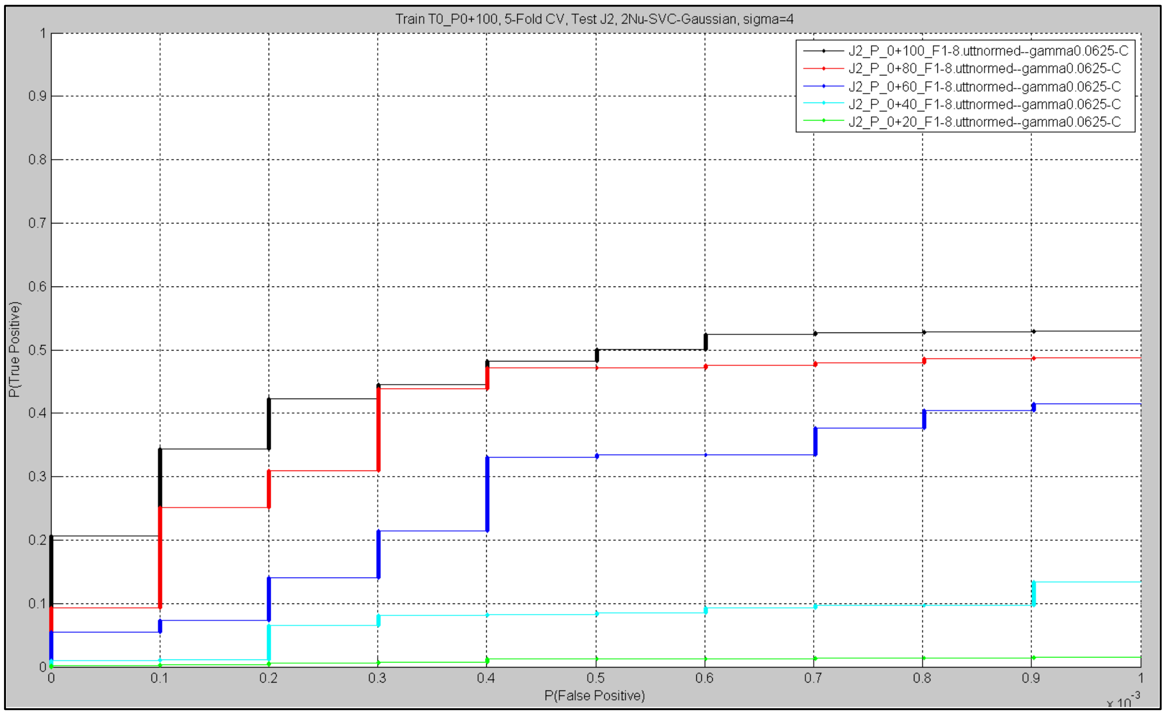


Figure 39 – ROC curves on J2 test data of the 2Nu-SVC (ν_-=0.009,ν_+=0.005, and σ=4,) trained and 5-fold cross-validated on T0 & T0_P_100. Note that the x-axis is scaled by 10^{-3}.

7. References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, New York, 1995.

[2] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, Kluwer Academic Publishers, 1998.

[3] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.

[4] M. A. Davenport, R. G. Baraniuk, and C. D. Scott, "Controlling false alarms with support vector machines," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Toulouse, France, 2006.

[5] M. A. Davenport, "The 2nu-SVM: A Cost-Sensitive Extension of the nu-SVM," *Rice University ECE Technical Report TREE 0504*, October 2005.

[6] T. Fawcett, "ROC Graphs: Notes and practical considerations for researchers," *Tech Report HPL-2003-4*, HP Laboratories, 2003.

[7] C.-W. Hsu, C.-C. Chang, C.-J. Lin, "A practical guide to support vector classification," 2003.

[8] D. Johnson, "QuickNet: a suite of software that facilitates the use of multi-layer perceptrons (MLPs) in statistical pattern recognition systems," Software available at http://www.icsi.berkeley.edu/Speech/qn.html.

[9] N. Morgan and H. Bourlard, "Continuous speech recognition," *IEEE Signal Processing Magazine*, vol. 12, no. 3, pp. 25-42, May 1995.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, 1986.

[11] B. Scholkopf and A. J. Smola, *Learning With Kernels*, MIT Press, Cambridge, Massachusetts, 2002.