APRIL 6, 2019

SUPPORT VECTOR MACHINE FOR CLASSIFICATION OF SPAM EMAIL MESSAGES

NATURAL LANGUAGE PROCESSING

SAMYUEL DANYO

https://github.com/SamyuelDanyo/

Introduction

In this report I present my implementation of the **S**upport **V**ector **M**achine (**SVM**) classification algorithm. In addition, two pattern recognition methods: **P**rincipal **C**omponent **A**nalysis (**PCA**) & **L**inear **D**iscriminant **A**nalysis (**LDA**), are utilized for transforming the data-space into, a better formed, feature space. The three methods are full-custom implementations in Python. Additionally, visualizations, classification results and analysis of the SPAM E-mail Dataset, are presented.

The goal of the project is to study the effectiveness of SVM for classification, as well as, the effects of the classifier's configuration & the input data structure on performance.

The dataset is comprised of 4601 samples, each with 57 features. Out of them 1813(39.4%) are labeled as spam, leaving 2248(60.6%) as non-spam. The full description of the SPAM E-mail dataset and clarification on what is assumed as spam can be seen in [1].

The dataset is randomized and divided into three sub-sets – train with 2000(43.5%) samples, test with 1536(33.4%) samples & validation with 1065(23.1%) samples. Three preprocessing techniques are applied to the dataset in order to condition it better for classification: standardization, binarization & log-transform.

For classification analysis of the model fit, a few different metrics are observed: accuracy, overall error rate(1-accuracy), false positives error rate and false negatives error rate.

While the main metrics for evaluating the performance of the methods are the resultant accuracy/overall error rate, for the specific case of the SPAM E-mail dataset – a deeper insight can be drawn from the false positives/negatives error rate. As mentioned by [1], the false positives (classifying a non-spam email as spam) is very undesirable as this can lead to the loss of important correspondence. Hence, when I discuss the performance of the model fits, a special attention is given to the false positives rate with the aim of minimizing it.

Additionally, the effect of data preprocessing & feature space transform are studied. Various configurations between the preprocessing functions: standardization; normalization; binarization; log transform; & the feature extraction techniques (data transformation): PCA; LDA; (with different dimensionalities), were examined. Three configurations were selected to be presented in this report: binarization + No transform (BNY), standardization + 15-D PCA (STD_PCA), log + 2-D LDA (LOG_LDA).

Furthermore, the effect that some hyperparameters have on the SVM model fit are studied: type of kernel (linear, polynomial & gaussian); type of margin (hard vs soft for polynomial kernel); the degree of the polynomial kernel; the value of the penalty parameter C (cost of violating constraints) for polynomial kernel.

The methods are implemented mainly through using NumPy for matrices manipulation and calculations. The CVXOPT package is used for solving the dual quadratic problem for SVM. Shared helper functions are implemented for preprocessing the dataset, visualizing the LDA feature space, building classification confusion tables and extracting the evaluation metrics. Further method-specific helper functions are implemented for wrapped training, testing, displaying results and complete fitting using matplotlib for plotting graphs and some other basic helper packages.

PCA for Feature Extraction

Design

The transform aims to find the directions (components) that maximize the variance of the input (training) dataset. PCA projects the input data space onto a sub-feature space with minimum information loss.

Implementation

The Algorithm Class

The transform is implemented as a class with one field {*Components*} for holding the principal components (eigen vectors), ranked by their meaningfulness (eigenvalues), and four methods:

- `__init__(self):`
  - For creating the object & declaring the field.
- `_check_eigen(self, eigen_val, eigen_vec, cov_mat):`
  - As an internal use only (as indicated by the preceding '_') function to check if the eigenvalues/eigenvectors equation **Σv=λv** is satisfied.
- `extract(self, X):`
  - For extracting the data (X) components.
- `project(self, X, dims=1):`
  - For creating the new feature space by projecting the input *X* into <*dims*>-dimensional subspace, based on the extracted components.

LDA for Feature Extraction & Visualization

Design

The transform aims to find the directions that maximize the variance (separation) between classes & minimize the variance (difference) inside the classes (represented by maximizing the Fisher's ratio: ***max((m1-m2)**2 / (v1+v2))***. LDA projects an input data space onto a sub-feature space with minimum class-discriminatory information loss.

Implementation

The Algorithm Class

The transform is implemented as a class with one field {*Directions*} for holding the extracted direction (eigen vectors), ranked by their meaningfulness (eigenvalues), and four methods:

- `__init__(self):`
  - For creating the object & declaring the field.
- `_check_eigen(self, eigen_val, eigen_vec, cov_mat):`
  - As an internal use only (as indicated by the preceding '_') function to check if the eigenvalues/eigenvectors equation **Σv=λv** is satisfied.
- `extract(self, X, T):`
  - For extracting the data (X, dependent on the class *T*) directions.
- `project(self, X, dims=1):`
  - For creating the new feature space by projecting the input *X* into <*dims*>-dimensional subspace, based on the extracted directions.

The Helper Functions

The LDA algorithm class has one helper function:

- `visualize_LDA(X, T, set_n = 'Test'):`
  - For visualizing the feature space.

SVM for Classification

Design

The SVM classifier aims to find the optimal hyperplane decision boundary (separation) between two classes {-1,1}. It approaches the task by maximizing the margin (distance) between the data and the hyperplane (the hyperplane can be shifted to make sure it separates the margin equally). The vectors going through the points between which is the margin are called support vectors. Maximizing the margin is proven to yield the optimal separation (leads to lower probability of misclassification) by Cristianini and Shawe-Taylor. Formulated as optimization problem, where we try to maximize the normalized (geometric) margin, while keeping the absolute (functional) margin the same (=1), leading to minimization of the weights (hyperplane parameters). The problem can be summed up as:

min(1/2W.T*W)
subject to: d*(W.T*X+b)>=1 (d = class)
find: w,b

Such primal optimization problem can be approached, using Lagrange Theorem, aiming to find the equality and inequality Lagrange multipliers. The optimization problem can be further remodeled, using the Kuhn-Tucker Theorem, to transform it from primal to dual problem, where a single parameter (the Lagrange multipliers) yields the solution. The dual problem is:

min(W=a*d*SV (SV=X for X which a=>0) or max(ai -1/2ai*aj*di*dj*Xi.T*Xj),
subject to: a*d=0
a=>0
Find: a

In order to solve the problem, dual quadratic programming is required, which yields the Lagrange multipliers and hence the hyperplane parameters.

The obvious problem though is that this is a linear separation. Meaning it will work for linearly separable classes. In order to improve the performance on overlapping data, the soft margin is introduced. The concept allows for data-points to be inside the margin (or on the wrong side of it), by introducing the slack variables, which added together and scaled by 'C' (the cost of violating the constraints) are the error penalty. The optimization problem aims not only to maximize the margin but also minimize the error:

min(1/2W.T*W + C*slack)
subject to: d*(W.T*X+b)>=1 -slack
slack>=0, hence 0<=a<= C (as part of the dual problem)

The third SVM concept is the kernel transform (known as the Kernel trick). It is based on Cover's Theorem, which states that: "Probability that classes are linearly separable increases when data points in input space are nonlinearly mapped to a higher dimensional feature space.". In order to deal with non-linear data (which is not separable in the data space), the kernel transform K(), maps the input data into a feature space, before extracting the support vectors.

Implementation

The Classifier Class

The SVM classifier is implemented as a class with nine fields {kernel, p, margin_type, C, W, b, a, SV, SV_T} for holding the classifier configuration, fitted model; and six methods:

- `__init__(self, kernel=linear_kernel, margin_type = 'hard', C=None, p=None):`
  - For creating the object & declaring the fields.
- `_get_gram_matrix(self, X):`
  - *For calculating the feature space gram matrix.*
- `_check_gram_matrix(self, GM):`
  - For checking if the gram matrix satisfies the Mercer's Condition (if the kernel is valid).
- `fit(self, X, T):`
  - For extracting the support vectors and finding the optimal hyperplane.
- `project(self, X):`
  - For projecting the input *X*, based on the extracted model.
- `predict(self, X):`
  - For applying the discriminative function to return predictions based on the SVM projection.

The Helper Functions

The SVM algorithm class has six helper functions:

- `show_results_SVM(svm, X, accuracy_train, accuracy_test, false_neg_train ,false_pos_train, false_neg_test, false_pos_test, fit_time):`
  - For displaying the fit profiling.
- `fit_SVM(X_train, T_train, X_test, T_test, kernel = linear_kernel, margin_type = 'hard', C=None, p = None, transform = 'None', pp_dims = 3, plot_lda = False):`
  - A Wrapper Function which encapsulates SVM fitting, testing, data gathering, visualizing & data display.
- `plot_non_lin_SVM(X_train, T_train, SVM):`
  - *For* visualizing the 2,3-D Non-Linear Kernel SVM fit margin.
- `linear_kernel(x1, x2):`
- `polynomial_kernel(x, y, p=3):`
- `gaussian_kernel(x, y, sigma=5.0):`

Experimental Setup

All experiments (without evaluation) are performed for each of the three preprocessing/transform configurations (binarization + No transform, standardization + 15-D PCA, log + 2-D LDA) of the dataset.

- Train & Test | Hard Margin SVM With Linear Kernel – the train set is used to fit SVM model. The test set is used to test its classification performance.
- Train & Test | Hard Margin SVM With Polynomial Kernel – the train set is used to fit SVM model. The test set is used to test its classification performance. The experiment is performed for p ∈ {2, 3, 4, 5}.

- ❖ Train & Test | Soft Margin SVM With Polynomial Kernel – the train set is used to fit SVM model. The test set is used to test its classification performance. The experiment is performed for $p \in \{1, 2, 3, 4, 5\}$ for $C \in \{0.1, 0.6, 1.1, 2.1\}$.
- ❖ EVALUATION | Soft Margin (C=1.1) SVM With Gaussian Kernel – the train set is used to fit SVM model. The evaluation set is used to test its classification performance.

Results and Discussion

Binary Dataset

When looking below at table 1, displaying the BNY dataset SVM fit results:

- ❖ Hard Margin with Linear Kernel: the SVM fit has poor performance of ~60% accuracy. This is to be expected as the linear kernel is not able to transform the non-linear features of the dataset and hence even the optimal hyperplane cannot provide good separation.
- ❖ Hard Margin with Polynomial Kernel: the non-linear kernel is able to extract a better-separated feature space, resulting in top test accuracy of 88.74% with only 2.08% false positives for p=4. While the accuracy grows with increasing p for 2-4, p=5 provides the worst fit. This might be caused by: either increased instability due to the higher degree of the function, or due to the higher non-linearity, which might be proving hard to optimize and hence, parts of the information might be lost.
- ❖ Soft Margin with Polynomial Kernel: as expected the soft margin, provides the best fit. The achieved maximum test accuracy is 94.01% with 2.93% false positives error rate for p=2. Similarly, to the point made above, p=5 seems to be unstable and hence results in a worse performance. Another interesting thing to observe is that with rising p, the model tends to overfit more. This is due to the better training representation, which the model learns very well but by doing that it loses on generalization. For p=3, the training accuracy goes as high as 99.10% with 0% false positives but the test accuracy suffers at 91.86%. The best fit is achieved for the softest margin with C=0.1. It is important to note that for p=2, C=2.1, the training accuracy is higher than that for p=2, C=0.1. However, when a look at the test accuracy is taken, the opposite is observed. This is due to hard margins fitting better to the training data (getting the optimal fit with least errors), while softer margins allow more errors and hence can generalize better.

Table 1: Binarized Dataset SVM Fit. Best Performance Per Type in Gold.

Dataset >>> Preprocessing: Binarization | Transform: None								
SVM Type	**Train Accuracy | False - | False + (%)**				**Test Accuracy | False - | False + (%)**			
Hard Margin With Linear Kernel	59.60 | 40.40| 0.00				61.91 | 38.09 | 0.00			
Hard Margin With Polynomial Kernel	**p = 2**	**p = 3**	**p = 4**	**p = 5**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
	87.65 7.95 4.23	85.15 12.25 2.60	88.25 10.10 1.65	79.25 4.40 16.25	88.74 7.03 4.23	86.20 11.00 2.80	88.74 9.18 2.08	78.78 4.75 16.47
Soft Margin With Polynomial Kernel	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**
p = 1	93.15 4.05 2.80	93.45 3.80 2.75	93.50 3.75 2.75	93.45 3.85 2.70	91.73 4.10 4.17	92.06 4.17 3.78	92.06 4.30 3.65	92.19 4.04 3.78
p = 2	97.45 1.80 0.75	98.70 1.20 0.10	98.95 1.00 0.05	99.05 0.95 0.00	94.01 3.06 2.93	92.51 3.65 3.84	92.06 3.91 4.04	91.99 3.84 4.17
p = 3	99.10 0.90 0.00	99.15 0.70 0.15	99.15 0.70 0.15	99.15 0.70 0.15	91.86 3.84 4.30	91.99 3.91 4.10	91.99 3.91 4.10	91.99 3.91 4.10
p = 4	95.45 4.30 0.25	95.50 4.25 0.25	95.50 4.25 0.25	95.50 4.25 0.25	89.65 6.45 3.91	89.65 6.45 3.91	89.65 6.45 3.91	89.58 6.51 3.91
p = 5	75.75 18.70 5.55	75.45 18.90 5.65	75.45 18.90 5.65	75.45 18.90 5.65	73.11 19.99 6.90	72.98 20.12 6.90	72.98 20.12 6.90	73.05 20.05 6.90

Standardized + PCA-transformed Dataset

As seen in table 2, the second preprocessing/transformation configuration (STD_PCA) yields similar observations as the binarization dataset. Instability can be observed for higher p, and overfitting for harder margins (bigger C).

While for the linear kernel the preprocessing of the dataset seems to make no difference (due to the already mentioned non-linearity), for the hard margin, polynomial kernel the PCA significantly underperforms the binarization. I believe this is due to the PCA transforming the feature space in targeting dimensions which hold as much information as possible. This means the boundaries between classes might be blurred and hence a hard margin performs poorly. On the flip side, it can be seen that, when soft margin is applied, the accuracy improves immensely. While it still slightly underperforms the binarization, it can be seen that with only 15 dimensions it performs closely, which could prove to be useful for big, multidimensional datasets since it reduces the size of the quadratic optimization problem.

Table 2: Standardize + PCA Dataset SVM Fit. Best Performance Per Type in Gold.

Dataset >>> Preprocessing: Standardization | Transform: 15-D PCA								
SVM Type	**Train Accuracy | False - | False + (%)**				**Test Accuracy | False - | False + (%)**			
Hard Margin With Linear Kernel	59.60 | 40.40| 0.00				61.91 | 38.09 | 0.00			
Hard Margin With Polynomial Kernel	**p = 2**	**p = 3**	**p = 4**	**p = 5**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
	59.60 40.40 0.00	60.25 0.00 39.75	60.00 0.00 40.00	41.70 0.00 58.30	61.98 38.02 0.00	60.09 1.24 38.67	57.03 1.43 41.54	39.39 0.07 60.55
Soft Margin With Polynomial Kernel	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**
p = 1	90.45 5.40 4.15	90.75 5.15 4.10	90.70 5.20 4.10	90.65 5.20 4.15	90.82 4.23 4.95	90.82 4.10 5.08	90.89 4.04 5.08	90.82 4.10 5.08
p = 2	94.95 3.00 2.05	95.45 2.65 1.90	95.40 2.75 1.85	95.30 2.80 1.90	92.32 2.73 4.95	91.60 2.93 5.47	91.41 2.86 5.73	91.60 3.06 5.34
p = 3	97.35 1.65 1.00	98.09 1.00 0.10	98.85 0.95 0.20	99.20 0.65 0.15	91.02 3.39 5.60	90.56 3.26 6.18	89.45 3.71 6.84	88.22 3.54 8.27
p = 4	69.65 8.20 22.15	80.05 17.95 2.00	67.50 6.20 26.30	47.00 2.75 50.25	67.71 6.12 26.17	80.40 14.06 5.53	63.87 6.38 29.75	45.31 1.89 52.80
p = 5	86.70 7.70 5.60	86.25 8.60 5.15	84.20 10.15 5.75	88.35 11.00 6.65	74.61 8.33 17.06	76.69 10.03 13.28	75.52 11.20 13.28	72.66 11.07 16.28

Log + LDA-transformed Dataset

The third data preprocessing/transform configuration (LOG_LDA) provides interesting insights. While for p=5, instability is seen, overall the model is more robust compared to the other configurations. Overfitting is highly decreased and almost unnoticeable. The 2-D data performs on par with the binarized dataset and even better than the PCA-transformation, achieving 93.82% accuracy with 3.19% false positives error rate for soft margin SVM with polynomial kernel (p=4, C=0.1).

The robustness of the fit in combination with the extreme dimensionality reduction and "good" feature space could be very useful for utilizing SVM on high-dimensionality, big datasets thanks to quadratic programming size reduction and making the classifier easier to tune.

Table 3: Logarithmic + LDA Dataset SVM Fit. Best Performance Per Type in Gold.

Dataset >>> Preprocessing: Log | Transform: 2-D LDA								
SVM Type	**Train Accuracy | False - | False + (%)**				**Test Accuracy | False - | False + (%)**			
Hard Margin With Linear Kernel	59.60 | 40.40| 0.00				61.91 | 38.09 | 0.00			
Hard Margin With Polynomial Kernel	**p = 2**	**p = 3**	**p = 4**	**p = 5**	**p = 2**	**p = 3**	**p = 4**	**p = 5**
	59.60 40.40 0.00	59.60 40.40 0.00	59.60 40.40 0.00	59.80 40.20 0.00	61.91 38.09 0.00	61.91 38.09 0.00	61.91 38.09 0.00	62.04 37.96 0.00
Soft Margin With Polynomial Kernel	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**	**C = 0.1**	**C = 0.6**	**C = 1.1**	**C = 2.1**
p = 1	94.40 2.90 2.70	94.50 2.80 2.70	94.50 2.80 2.70	94.40 2.90 2.70	93.55 2.93 3.52	93.62 2.86 3.52	93.62 2.86 3.52	93.62 2.86 3.52
p = 2	94.50 3.05 2.45	94.40 2.85 2.75	94.30 2.80 2.90	94.25 2.75 3.00	93.68 3.06 3.26	93.55 2.93 3.52	93.42 2.86 3.71	93.23 2.80 3.97
p = 3	94.40 2.90 2.70	94.40 2.90 2.70	94.40 2.90 2.70	94.40 2.90 2.70	93.62 2.86 3.52	93.62 2.86 3.52	93.55 2.93 3.52	93.62 2.86 3.52
p = 4	94.45 3.00 2.55	94.55 2.95 2.50	94.45 3.10 2.45	94.45 3.10 2.45	93.82 2.99 3.19	93.62 2.99 3.39	93.68 3.12 3.19	93.49 3.32 3.19
p = 5	8.90 33.30 57.80	82.60 14.40 3.00	7.25 35.10 57.65	44.45 26.55 29.00	9.18 31.05 59.77	84.18 12.11 3.71	7.68 32.88 59.44	43.68 26.37 29.95

Effects of Kernel Choice

Looking at the binarized dataset, it can be clearly seen that utilizing non-linear kernel significantly boosts performance for the particular problem. This is in line with theory. The argument for kernel in the first place is that higher-dimensional data provides better separation. When this effect is coupled with non-linear transformation, it can be seen how the benefit of increasing separability on non-linear data will be compounding.

Nevertheless, ss seen by the PCA/LDA transformed datasets, simply choosing a kernel is not enough. Attention needs to be given to the particular problem dataspace and further aspects of the learning (like the margin) might need to be tuned.

Effects of Polynomial Kernel Degree (Level of Non-Linearity)

The goal of polynomial kernel is to transform the data space into a feature space, where data samples are mapped over polynomials and hence non-linearly separable data can be mapped to a more-separable features, which relate to each other based on their similarity. My observations are in line with theory: with higher degrees, the model is more prone to instability and overfitting.

Effects of Margin Choice

As seen in all datasets, the choice of margin is extremely important. While hard margin can be useful for simpler problems, usually with good separability, it yields poor performance in complex feature spaces. Soft margin not only provides a better model fit but it also gives the

flexibility of controlling the trade-off between the perfect training fit and possible better generalization. Furthermore, as seen by the STD+PCA & LOG+LDA -transformed datasets utilizing soft margin might be the way to unlock the potential fit of the feature space and take advantage of its topology. In both cases testing accuracy went up over 30% after changing from hard to soft margin.

Effects of Soft Margin Cost Parameter Choice

In theory - smaller C value (softer margin) leads to larger margin and more misclassification of training data but could yield better test performance. Large C (moves towards hard margin) leads to smaller margin and less misclassification of training data but can cause overfitting. This could be looked at as the trade-off between generalization and complexity of the model. Usually a good balance needs to be found in order to get larger margin, which would allow generalization, while not over-extending it, which would lead to subdued learning and a bad fit.

The experimental results support the theory. It can be seen that with increasing C, the model overfits and testing accuracy drops. This can be clearly observed in the binarization dataset results. For soft margin SVM with polynomial kernel (p=2), it can be seen that for C=2.1 – training accuracy is 99.05%, while test accuracy is 91.99%; for C=0.1, the numbers are – 97.45% and 94.01%!

Effects of Preprocessing/Transformation Choice

As proved by the above presented results, the input data space, specifying the problem is determinant of the SVM performance. As seen in the table below, which compares the preprocessed datasets SVM fits with and without additional transformation the benefit of such transform depends on the architecture of the SVM classifier. Hard margin seems to be unable to fit the transformed feature spaces and hence transformation results in reduced accuracy. However, once soft margin is utilized, transformation greatly boosts performance.

Table 4: Effect of PCA/LDA on STD/LOG Datasets SVM Fit. Best Performance in Gold.

Dataset >>> Preprocessing: Standardization | Log | Transform Effect				
SVM Type	**Train Accuracy | False - | False + (%)**		**Test Accuracy | False - | False + (%)**	
Hard Margin With Polynomial Kernel	**STD | p = 3**	**LOG | p = 3**	**STD | p = 3**	**LOG | p = 3**
NO TRANSFORM	83.75 16.26 0.00	82.05 8.70 9.25	80.21 9.18 10.61	80.21 9.18 10.61
TRANSFORM	60.25 0.00 39.75	59.60 40.40 0.00	60.09 1.24 38.67	61.91 38.09 0.00
Soft Margin With Polynomial Kernel	**STD | C = 0.6**	**LOG | C = 0.6**	**STD | C = 0.6**	**LOG | C = 0.6**
NO TRANSFORM p = 3	94.30 3.25 2.45	44.45 33.05 22.50	84.57 6.77 8.66	44.66 32.94 22.40
TRANSFORM p = 3	98.09 1.00 0.10	94.40 2.90 2.70	90.56 3.26 6.18	93.62 2.86 3.52

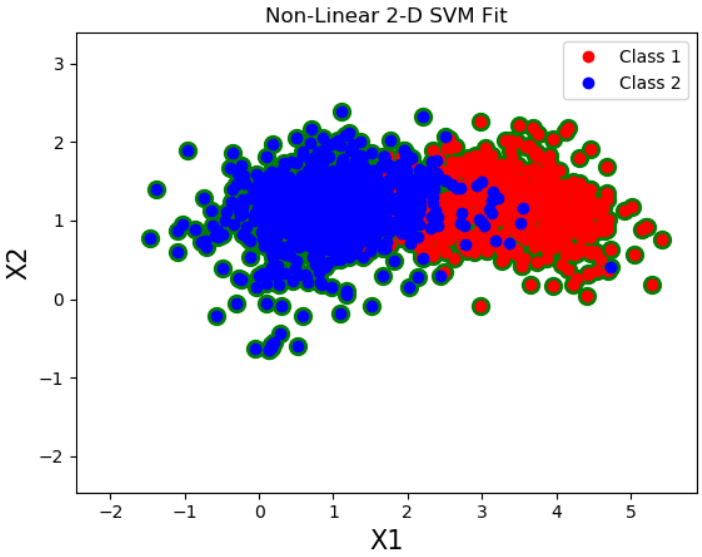

Fig.1: Hard Margin, Polynomial Kernel SVM LOG+LDA Dataset Fit.

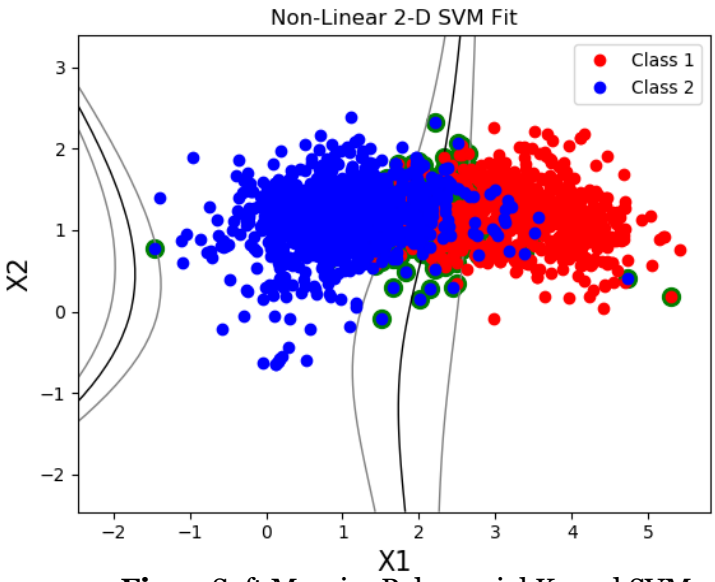

Fig.2: Soft Margin, Polynomial Kernel SVM LOG+LDA Dataset Fit.

In the above figures, visualization of the SVM LOG+LDA SVM fits are shown. It can be seen that the hard margin SVM takes all points as support vectors and is unable to form a good separation solution (has a zero margin), while the soft margin SVM builds a good margin separation.

Final Discussion

In the experiment, all proposed kernels satisfied the Mercer's condition and hence were used for extracting a model fit.

Overall, the highest performance was achieved by the binarized dataset without further transformations with 94.01% accuracy and 2.93% false positives. Binarization seems to condition the feature space very well for SVM learning, since it significantly outperformers the standardized and logarithmic (as well as, normalized, which is not presented) preprocessed datasets without transformation.

Once PCA & LDA transformation is applied to the standardized and logarithmic datasets, their performance comes very close to that of the binarized dataset, with LOG+LDA outperforming STD+PCA. The value of the transformation comes from the fact that the LDA dataset is only 2-D, yet it is able to perform on par with the 57-D binarized samples.

Polynomial kernel with low-to-average degree (p=2,3), tends to perform best on this particular problem, balancing out overfitting & instability with better separability of the non-linear data. Utilizing LDA gives better stability across degrees and reduces overfitting. Additionally, gaussian kernel was fitted as part of the evaluation section. It performs very well, on the same level as the top polynomial fits, with skew towards the transformed sets.

For the specific problem of classifying spam emails, using softer margin (C~0.1) outperforms as the data is non-linear and overlapping. Harder margins lead to overfitting.

[1] *Web.stanford.edu*, 2019. [Online]. Available: https://web.stanford.edu/~hastie/ElemStatLearn/datasets/spam.info.txt [Accessed: 7- Apr- 2019].