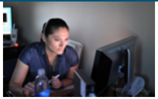
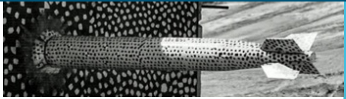
SAND2024-06425C

Classification Using Support Vector Machines with Uncertainty Quantification

Presented by:

Sofia Taylor, Kyle Neal, Erin Acquesta

Feb 27, 2024

Outline:

- Motivation
  - Structural Health Monitoring
- SVMs
  - Toy data
  - Variations and Ensembles
  - High dimensional data
  - In progress: PCA
- Ideas for improvement

Motivation: Structural Health Monitoring (SHM)

Source: Tufts Dept. of Civil and Envr. Eng.

- Assess the health of an engineered system through non-intrusive measurement of damage signatures
- SHM is used in many fields
- Measurements may be time series data
- We will assume pristine baseline measurements also exist
- Can ML identify damaged structures from SHM measurements and include UQ?

Support Vector Machine

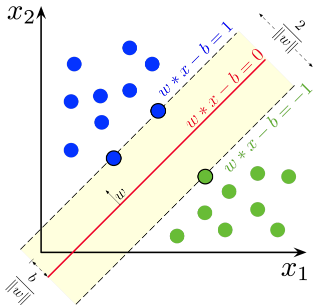


Figure: wikipedia

- Idea: hyperplane ($w^T x = b$) to separate 2 classes of data while maximizing the width of the margin.
- Datapoints are represented by location (x_i) and class (y_i):

  ● $y_i = 1$ ● $y_j = -1$

- w is the normal vector, and the width of the margin is given by $\frac{2}{\|w\|}$

Optimization: Hard Margin

- Optimization problem:

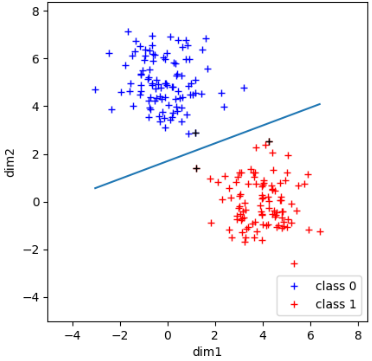
$$\max_{\text{hyperplanes}} \quad \text{margin length}$$

$$s.t. \quad \text{all samples on correct side of margin}$$

- Formally:

$$\min_{w,b} \frac{1}{2}\|w\|_2^2$$

$$s.t. \quad y_i(w^T x_i - b) \geq 1 \quad \text{for } i = 1 : \ell$$



When $y_i(w^T x_i - b) = 1$, x_i is a *support vector*. The collection of support vectors are the only vectors that define the hyperplane and the margin.

Optimization: Soft Margin

- Optimization problem:

$$\max_{\text{hyperplanes}} \quad \text{margin length}$$

$$s.t. \quad \textit{most} \text{ samples correct side of margin}$$

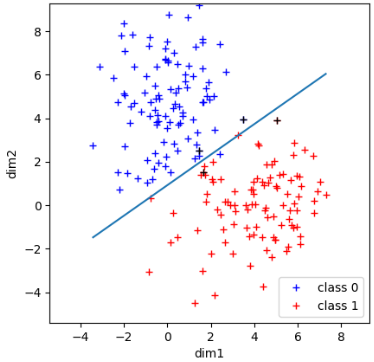
while penalizing misclassification of samples

- Formally:

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_i \xi_i$$

$$s.t. \quad y_i(w^T x_i - b) \geq 1 - \xi_i,$$

$$\xi_i \geq 0 \quad \text{for } i = 1:\ell$$



When $\xi_i > 0$, x_i is in the margin or passed it.

Our datasets:

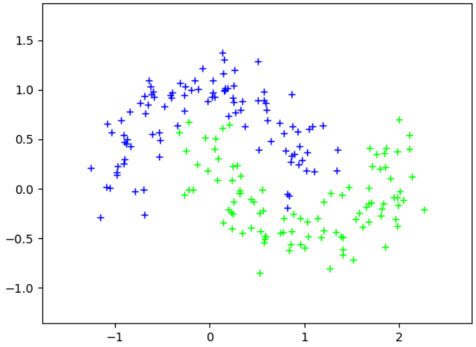


Figure: Moon dataset

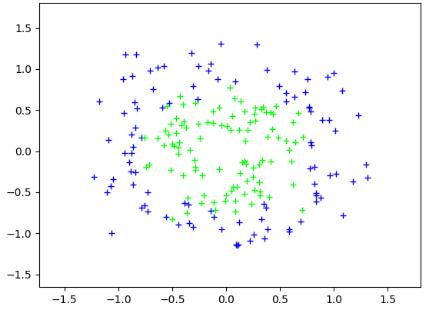
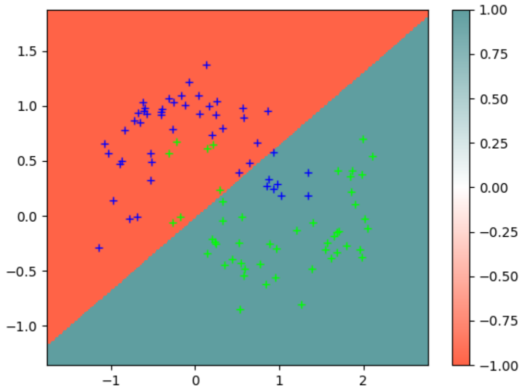


Figure: Donut dataset

Not linearly separable :(

We need nonlinear decision boundaries!

Even though soft-margin SVM can find a linear decision boundary, it does not perform well.



Kernel methods

- SVM performs best on higher-dimensional data because it is more likely to be linearly separable
- Project original data to observables in some higher dimension:

$$x_i \longrightarrow \phi(x_i)$$

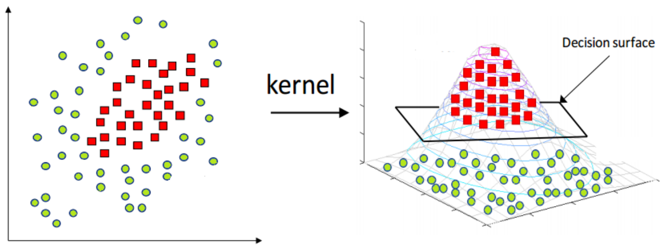


Figure: Zhang, Grace, medium.com

- Curse of dimensionality: expensive to work directly with observables

Kernel Trick

- The kernel trick allows us to use the observables without doing any computations in the high-dimensional space.
- The kernel represents inner products between observables:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

- Gaussian kernel:

$$K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}),$$

- Polynomial kernel:

$$K(x_i, x_j) = (x_i^T x_j + a)^p,$$

- σ, a, p are adjustable parameters

But first...

- We want to solve this (hard-margin) problem P:

$$\min_{w,b} \frac{1}{2} w^T w \qquad s.t. \quad y_i(w^T \phi(x_i) - b) \geq 1 \quad \text{for } i = 1 : \ell$$

  but since w is also a vector in the feature space, solving this directly requires working in the feature space.

- Lagrangian:

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2} w^T w - \sum_{i=1}^{\ell} \lambda_i \left[y_i(w^T x_i - b) - 1 \right]$$

- Since the objective function is *convex* and the constraint functions are also *convex*, Karush-Khan-Tucker theorem states that:
- w_*, b_*, λ_* is a saddle point of $\mathcal{L}$ if and only if w_*, b_* is optimal for P.

Dual Problem and KKT Conditions

$$\max_{\lambda} \mathcal{L}(w_*, b_*, \lambda) = \max_{\lambda} \min_{w,b} \mathcal{L}(w, b, \lambda). \tag{1}$$

To find w_*, b_* in terms of λ, we set

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \implies w_* = \sum_i \lambda_i y_i \phi(x_i) \tag{2}$$

and

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \implies \sum_i \lambda_i y_i = 0. \tag{3}$$

And the following are the constraints on the KKT multipliers (nonnegativity and complimentary slackness):

$$\lambda_i \geq 0 \quad \text{and} \quad \lambda_i \left[y_i (w^T \phi(x_i) - b) - 1 \right] = 0 \quad \forall i. \tag{4}$$

QP Simplification

After plugging in KKT conditions, this reduces to

$$\max_{\lambda} \mathcal{L}(w_*, b_*, \lambda) = \max_{\lambda} \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \tag{5}$$

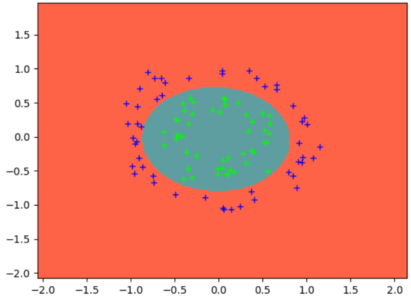
$$= \min_{\lambda} \frac{1}{2} \lambda^T X^T X \lambda - 1_\ell^T \lambda \qquad \text{s.t. } \lambda_i \geq 0, \quad \sum_i y_i \lambda_i = 0 \tag{6}$$

where

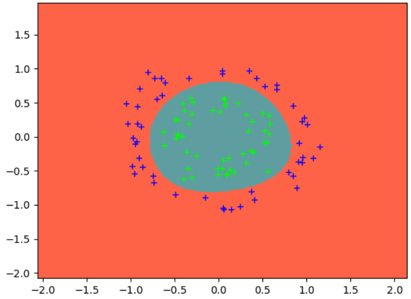
$$X = \begin{bmatrix} \dots & \begin{array}{c} | \\ y_i \phi(x_i) \\ | \end{array} & \dots \end{bmatrix}$$

When using the kernel trick, $X^T X_{i,j} = Ker(x_i, x_j)$ and $X^T X$ is symmetric positive (semi) definitess.

Hard Margin Polynomial Kernel SVM



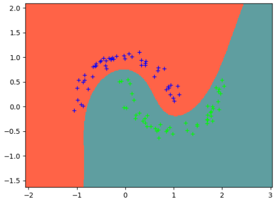
(a) Polynomial kernel with $p = 2$



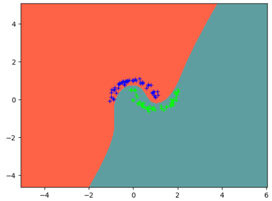
(b) Polynomial kernel with $p = 5$

Figure: Note that there is not a big difference between the two parameter settings on this data set.

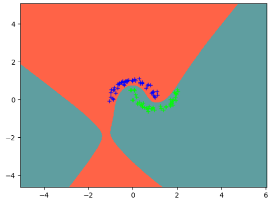
Hard Margin Polynomial Kernel SVM



(a) Polynomial kernel with $p = 3$



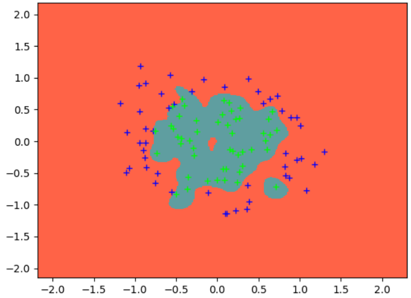
(b) Polynomial kernel with $p = 3$, zoomed out.



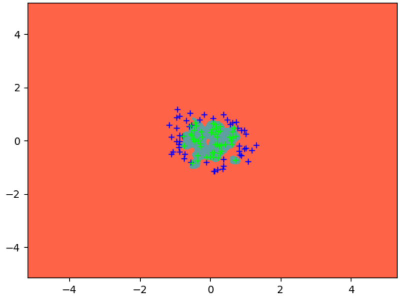
(c) Polynomial kernel with $p = 4$, zoomed out.

Figure: Similar decision boundary for $p = 3$ and $p = 4$ near the data, but far enough out $p = 4$ extrapolates wildly.

Hard Margin Gaussian

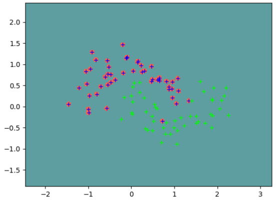


(a) Gaussian kernel with $\sigma = .1$

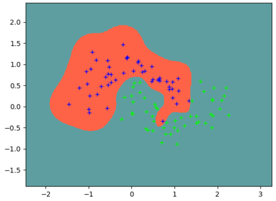


(b) Same as (a)

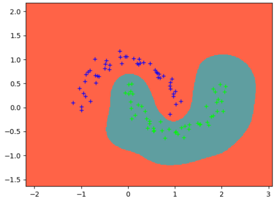
Hard Margin Gaussian



(a) Gaussian kernel with $\sigma = .02$

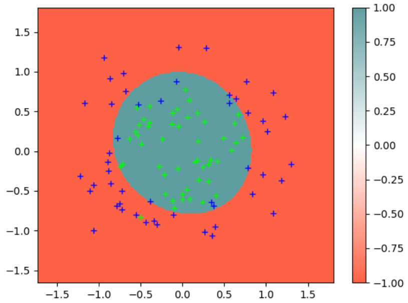


(b) Gaussian kernel with $\sigma = .25$

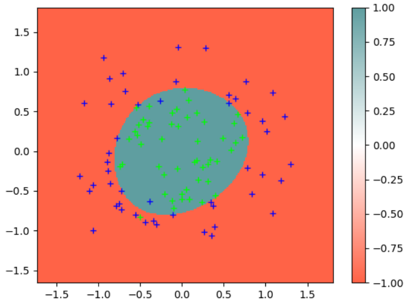


(c) Gaussian kernel with $\sigma = .8$ (less noisy data).

Soft Margin Polynomial SVM

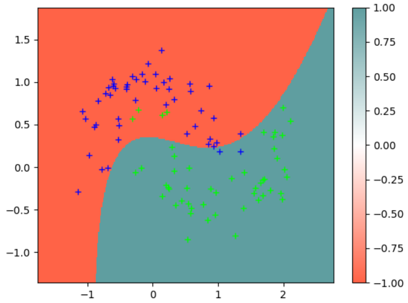


(a) $C = .1, p = 3$

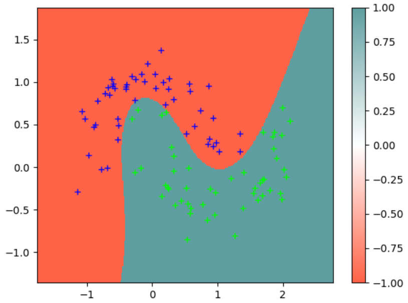


(b) $C = 10, p = 3$

Soft Margin Polynomial SVM



(a) $C = .1, p = 3$



(b) $C = 10, p = 3$

Figure: Polynomial softmargin with varying penaltization for misclassification.

Ensembling

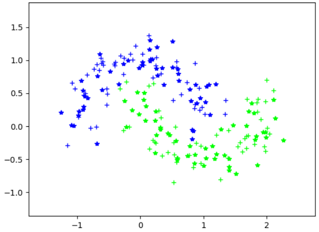


Figure: Moon dataset

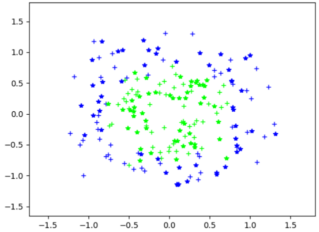


Figure: Donut dataset

- 100 training samples (pluses) and 100 testing data (stars) selected at random and evenly split between each class
- A model must pass with 80% accuracy on the test samples to be included in the ensemble

Ensembles

Parameters that vary between ensembles: C (penalization for misclassification), and kernel parameters p, σ.
Intuition behind colorbar: White: Overlap (uncertainty), Red: class 1, blue: class 2

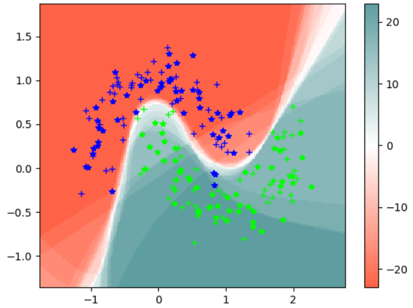


Figure: 22 Polynomial SVMs

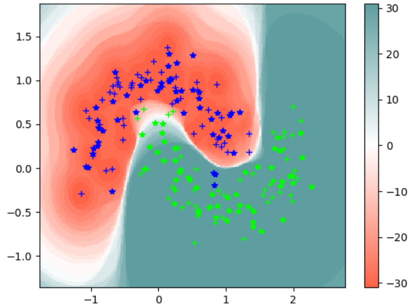


Figure: 30 Gaussian SVMs

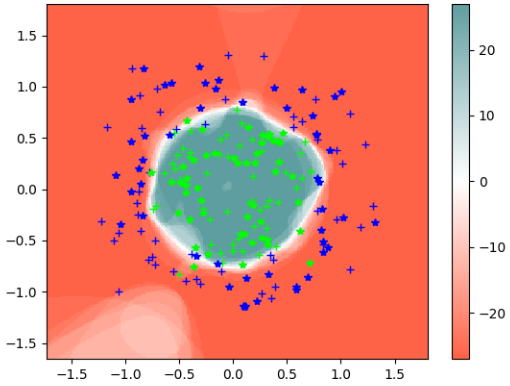


Figure: Polynomial and Gaussian kernels on the donut dataset, varying C, σ, p over 27 models. The confusion matrix: $\begin{bmatrix} 45 & 5 \\ 4 & 46 \end{bmatrix} = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix}$ gives the average number of true/false positives (green stars)/negatives (blue stars) classified by the SVMs.

Aleatoric Uncertainty

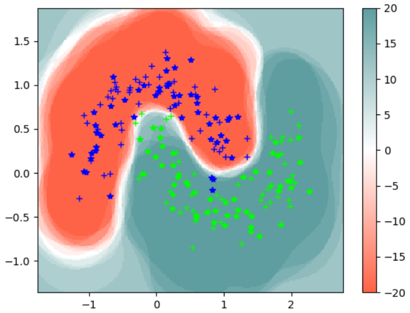


Figure: Using the Gaussian kernel with $\sigma = .25$ and $C = 1$, we tested 20 different sets of training data. To create a training data set, for each point x_i in the the original training dataset, a new point y_i was selected from a normal distribution around x_i. Then the SVM was ran on the collection of y_is.

In-class/out-of-class distinction

- Previous ensembles give high certainty far away from the data
- Solution: separating and *surrounding* each class
- Optimization problem given only one class (SVDD):

$$\min_{\text{hyperspheres}} \quad \text{radius}$$

$$s.t. \quad \text{hypersphere encloses most in-class samples}$$

$$\text{while penalizing samples outside of the sphere}$$

- Formally:

$$\min_{r,b} r^2 + C \sum \xi_i \tag{7}$$

$$s.t \|\phi(x_i) - b\| \leq r^2 + \xi_i \tag{8}$$

$$\xi_i \geq 0 \tag{9}$$

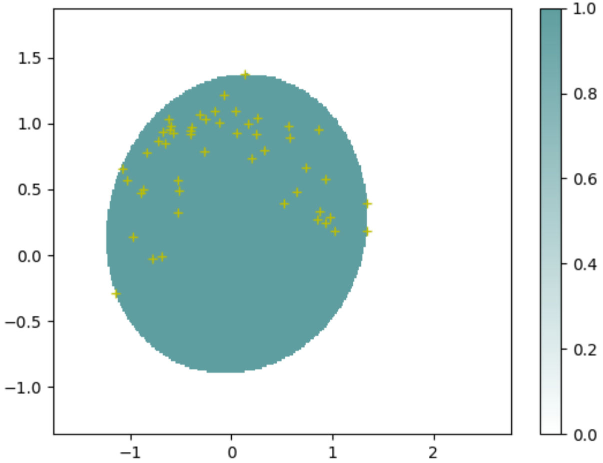


Figure: Gaussian kernel for Single class SVM with $\sigma = 2$

Single-Class SVM with negative examples

- What if we have samples that we know do *not* belong in the class (we do)?

$$\min_{\text{hyperspheres}} \quad \text{radius}$$

$$s.t. \quad \text{hypersphere encloses most in-class samples}$$

$$\text{while penalizing positive (negative) samples outside (inside) the sphere}$$

- Formally:

$$\min_{r,b} r^2 + C_1 \sum \xi_i + C_2 \sum \beta_i \tag{10}$$

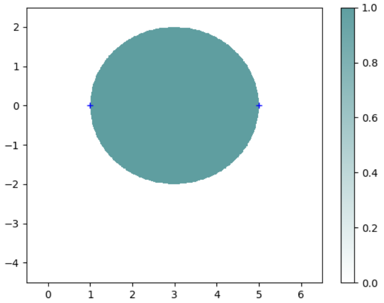
$$s.t \quad \|\phi(x_i) - b\| \leq r^2 + \xi_i \tag{11}$$

$$\|\phi(y_i) - b\| \geq r^2 - \beta_i \tag{12}$$

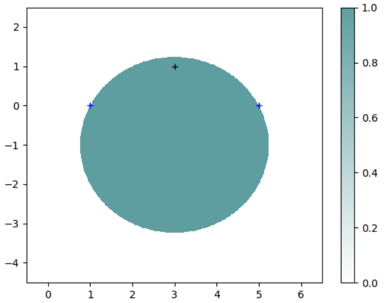
$$\xi_i \geq 0 \tag{13}$$

$$\beta_i \geq 0 \tag{14}$$

Comparisons

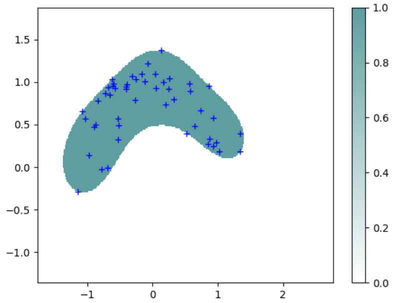


(a) Linear single class SVM with no negative samples ($C = 1$)

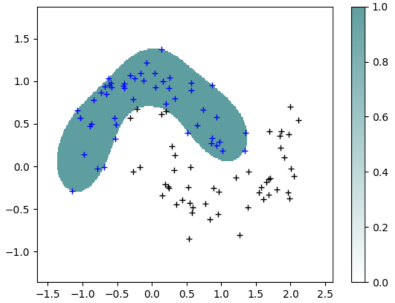


(b) Linear with one negative sample ($C_1 = 1, C_2 = 1$)

Comparisons



(a) Gaussian kernel with no negative examples

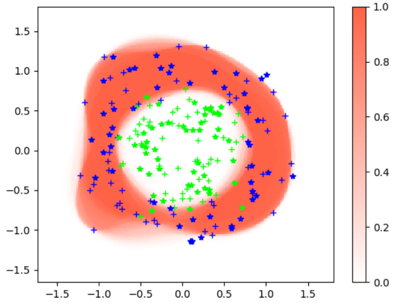


(b) Gaussian kernel with negative examples

Ensemble of (Gaussian) Single-Class Classifiers

In addition to kernel parameters, there are 2 penalization parameters to vary:

- C_1 penalization for in-class outliers
- C_2 penalization for out-class inliers



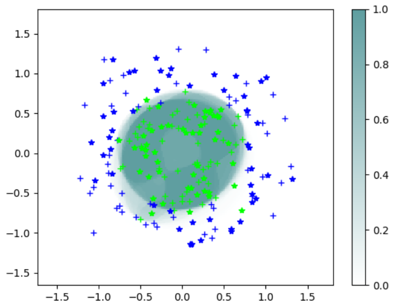
163 SVMs passed for class 1 with average confusion matrix:

$$\begin{bmatrix} \# \text{TP} & \# \text{FN} \\ \# \text{FP} & \# \text{TN} \end{bmatrix} = \begin{bmatrix} 40.5 & 9.5 \\ 5.2 & 44.8 \end{bmatrix}.$$

Ensemble of (Gaussian) Single-Class Classifiers

Note:

- More false negatives than false positives
- There are less SVMs passing with 80% accuracy for class 2



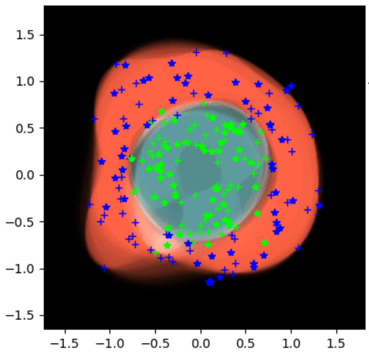
114 SVMs passed for class 2 with average confusion matrix:

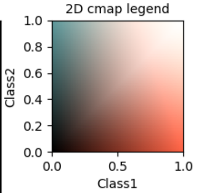
$$\begin{bmatrix} \# \text{TP} & \# \text{FN} \\ \# \text{FP} & \# \text{TN} \end{bmatrix} = \begin{bmatrix} 44.4 & 5.6 \\ 1.6 & 48.4 \end{bmatrix}$$

Combining the ensembles

Each point is associated with a vector (p_1, p_2):

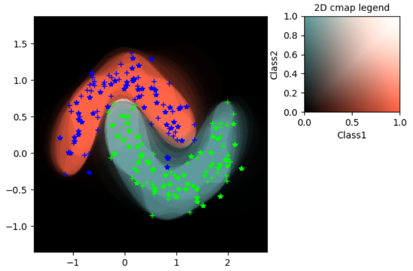
- p_1 is the value from the class 1 ensemble, p_2 from the class 2 ensemble
- $p_1 + p_2 \neq 1$





- White $(1, 1)$: indicates uncertainty from class overlap
- Black $(0, 0)$: indicates uncertainty from lack of data

Moon Dataset Gaussian Ensemble



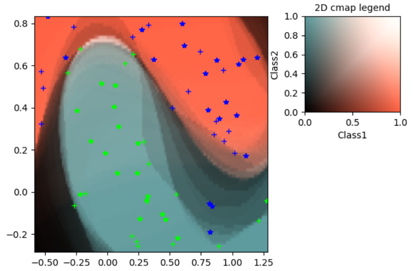


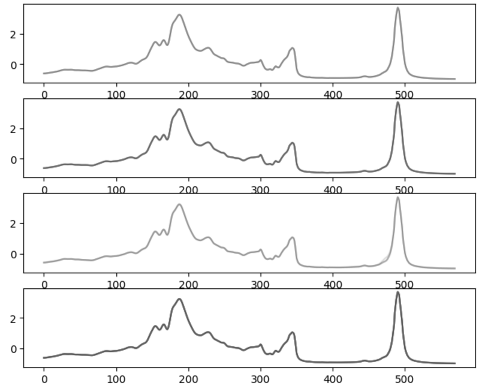
Figure: 139 SVMs passed for class 1 with confusion matrix $\begin{bmatrix} 42.3 & 7.7 \\ 1.3 & 48.7 \end{bmatrix}$. 129 SVMs passed for class 2 with confusion matrix: $\begin{bmatrix} 45.4 & 4.6 \\ 2.6 & 47.4 \end{bmatrix}$

Higher Dimensional Data: Time Series

- Class overlap is less likely: easier to surround and separate a class without using kernels
- Overfitting more likely with kernels
- New datasets!

Olive Oil: Food Spectrographs

Task: distinguish 4 different types of olive oil given food spectroscopy data (vibrational data from exposure to infrared radiation)



- Number of classes: 4
- Train size: 30
- Test size: 30
- Series length: 570
- Ensemble: Each model must pass with 75% accuracy: 40 SVMs for class 1, 44 for class 2, 41 for class 3, only 3 for class 4

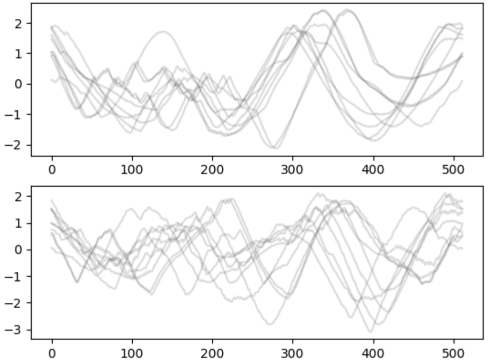
Olive Oil: Average confusion matrices

- 1: $\begin{bmatrix} 2.15 & 2.85 \\ 0 & 25 \end{bmatrix}$
- 2: $\begin{bmatrix} 7.2 & 1.8 \\ .5 & 20.5 \end{bmatrix}$
- 3: $\begin{bmatrix} 2.6 & 1.5 \\ 1.5 & 24.5 \end{bmatrix}$
- 4: $\begin{bmatrix} 11 & 2.7 \\ 1 & 14.3 \end{bmatrix}$

To note: more than 2 classes $\implies$ many more positive samples than negative for a given class. This means a model may classify all the samples as negative, and still pass with 75% accuracy.

Bird vs Chicken

Task: distinguish birds from chickens

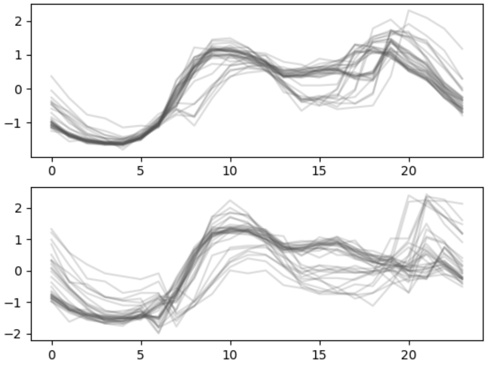


- Data: distance from the center while tracing the silhouette
- Train size: 20
- Test size: 20
- Series length: 512

Testing data: class 1 (top), class 2 (bottom). Not one SVM could pass with 75% accuracy.

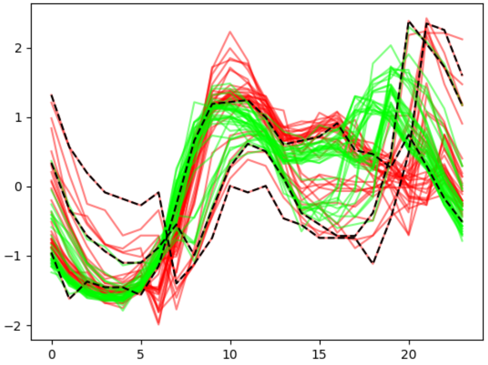
Daily Power Demand in Italy

Each time series is (centered) power usage in a day. Task: distinguish days in Oct-March from days in Apr-Sept



- Train size: 67
- Test size: 1029
- Series length: 24
- Testing data: class 1 (top), class 2 (bottom)

Daily Power Demand in Italy: No Kernel



- In class: red
- 94% accuracy on testing data!
- $\begin{bmatrix} 499 & 17 \\ 44 & 469 \end{bmatrix}$
- Figure: testing data. Dashed lines are the data points defining the border of the hypersphere
- Hard to visualize :(

Daily Power Demand in Italy: Ensemble

- 42 SVMs passed for class 1, 43 for class 2.
- The averaged confusion matrices are $\begin{bmatrix} 466.9 & 49.1 \\ 42.5 & 470.5 \end{bmatrix}$, $\begin{bmatrix} 415.0 & 49.0 \\ 36.5 & 430.5 \end{bmatrix}$
- Again, more false negatives than false positives

Daily Power Demand in Italy: Ensemble

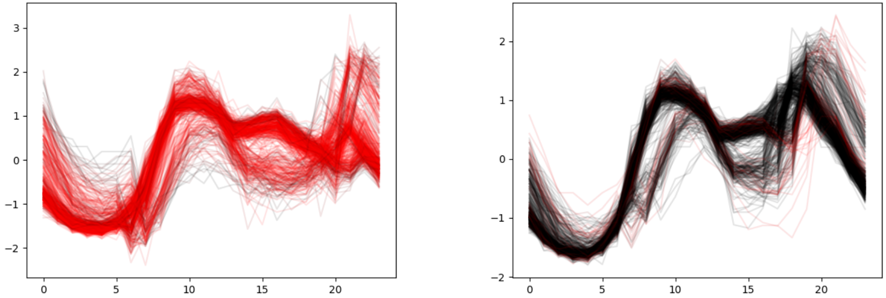
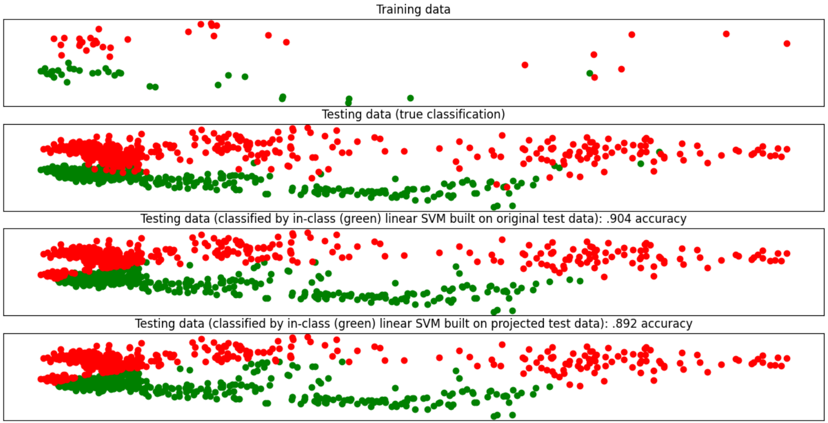


Figure: Ensemble for class 1. Class 1 testing data (left), Class 1 testing data (right).

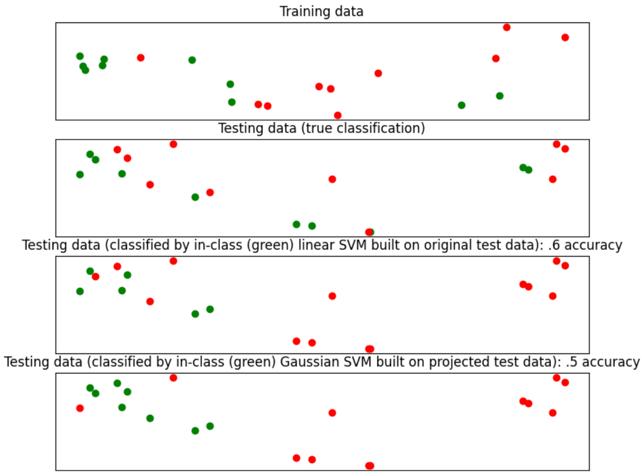
PCA: project onto hyperplane of best fit (ongoing work)

- PCA (Principal Component Analysis): Use singular value decomposition to find the hyperplane of best fit (described by first and second principal components)
- Project data onto hyperplane
- Classify before or after the projection?
- Adds uncertainty

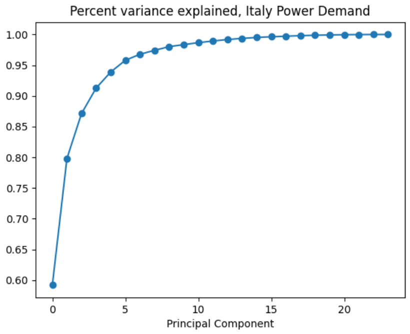
PCA: Power Demand in Italy

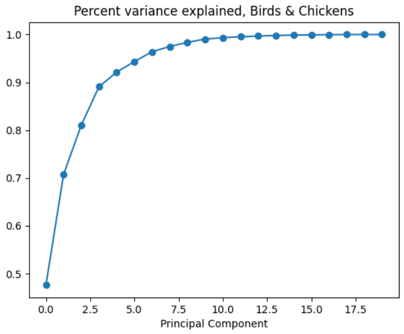


PCA: Birds vs Chickens



Percent Variance Explained





PCA: future work

- Experiment with number of principal components
- Variance explained vs separability: maybe the components that explain the most variance are not the components where the data is the most separable
- Functional PCA:
  - Allows for data sets with time series of different lengths
  - Considers the order of the data (unlike PCA)

Conclusions:

What we have addressed so far:

- Classification (SVMs)
- Uncertainty in classification due to
  - Class overlap
  - Extrapolation

Future plans:

- Apply fPCA
- Incorporate UQ analysis of the PCA
- Incorporate QUQ concepts to assess the accuracy of our UQ estimates

Thank you!

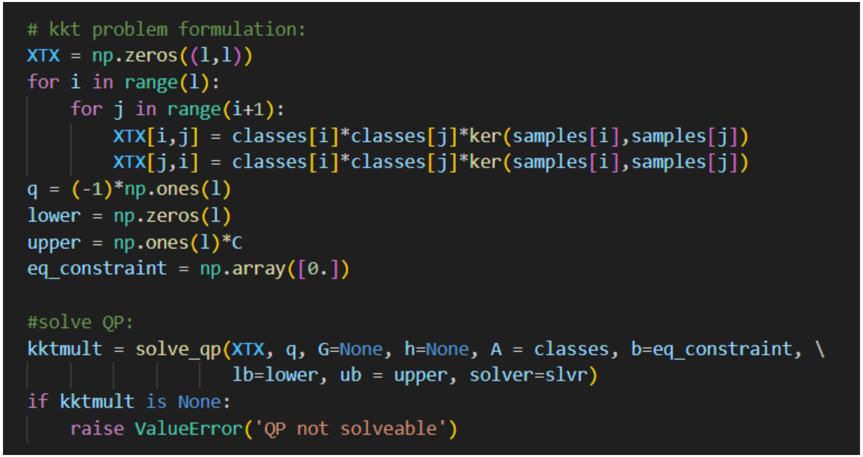
Notes

- We don't know the observables / feature space
- We cannot solve for w ($w = \sum_i \lambda_i y_i \phi(x_i)$)
- We cannot find the decision boundary analytically
- We can test a point by checking whether

$$w^T x - b = \sum_i \lambda_i y_i Ker(x_i, x) - b$$

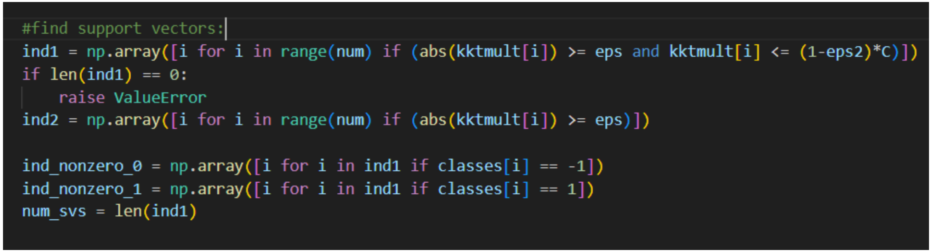
is positive or negative.

Code: 1. Formulate and Solve QP



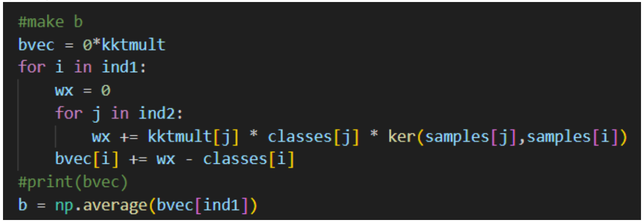
```
# kkt problem formulation:
XTX = np.zeros((l,l))
for i in range(l):
    for j in range(i+1):
        XTX[i,j] = classes[i]*classes[j]*ker(samples[i],samples[j])
        XTX[j,i] = classes[i]*classes[j]*ker(samples[i],samples[j])
q = (-1)*np.ones(l)
lower = np.zeros(l)
upper = np.ones(l)*C
eq_constraint = np.array([0.])

#solve QP:
kktmult = solve_qp(XTX, q, G=None, h=None, A = classes, b=eq_constraint, \
                        lb=lower, ub = upper, solver=slvr)
if kktmult is None:
    raise ValueError('QP not solveable')
```

2. Find Support Vectors



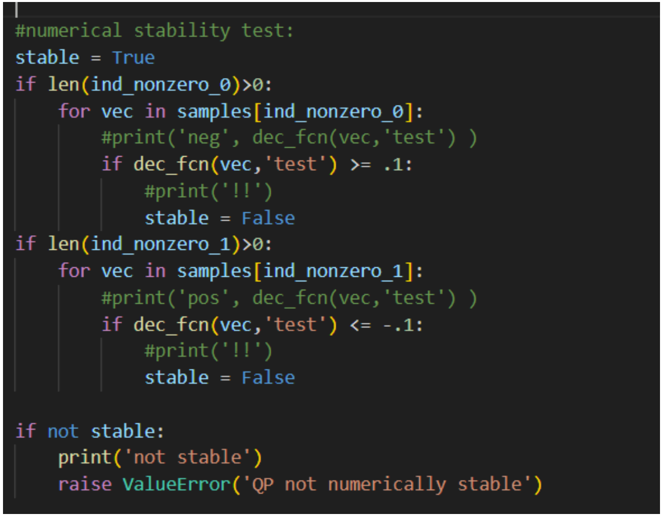
```python
#find support vectors:
ind1 = np.array([i for i in range(num) if (abs(kktmult[i]) >= eps and kktmult[i] <= (1-eps2)*C)])
if len(ind1) == 0:
    raise ValueError
ind2 = np.array([i for i in range(num) if (abs(kktmult[i]) >= eps)])

ind_nonzero_0 = np.array([i for i in ind1 if classes[i] == -1])
ind_nonzero_1 = np.array([i for i in ind1 if classes[i] == 1])
num_svs = len(ind1)
```

3. Solve for b



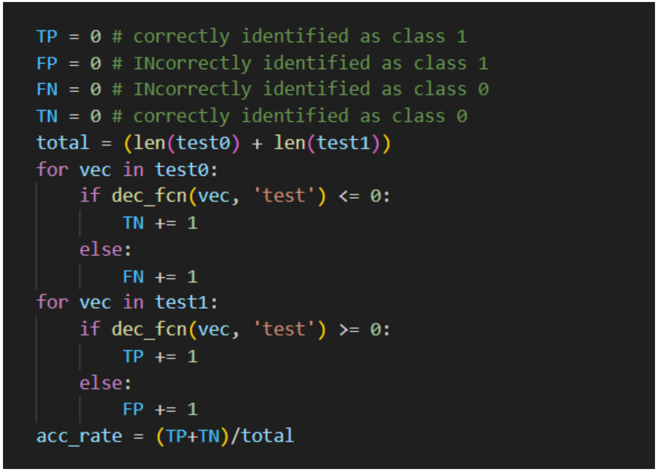
```python
#make b
bvec = 0*kktmult
for i in ind1:
    wx = 0
    for j in ind2:
        wx += kktmult[j] * classes[j] * ker(samples[j],samples[i])
    bvec[i] += wx - classes[i]
#print(bvec)
b = np.average(bvec[ind1])
```

4. Check Numerical Stability of QP solution



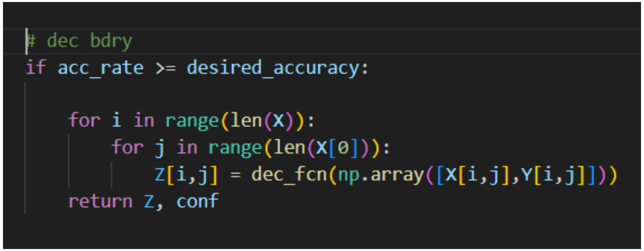
```python
#numerical stability test:
stable = True
if len(ind_nonzero_0)>0:
    for vec in samples[ind_nonzero_0]:
        #print('neg', dec_fcn(vec,'test') )
        if dec_fcn(vec,'test') >= .1:
            #print('!!')
            stable = False
if len(ind_nonzero_1)>0:
    for vec in samples[ind_nonzero_1]:
        #print('pos', dec_fcn(vec,'test') )
        if dec_fcn(vec,'test') <= -.1:
            #print('!!')
            stable = False

if not stable:
    print('not stable')
    raise ValueError('QP not numerically stable')
```

5. Compute Accuracy Rate and Confusion matrix



```
TP = 0 # correctly identified as class 1
FP = 0 # INcorrectly identified as class 1
FN = 0 # INcorrectly identified as class 0
TN = 0 # correctly identified as class 0
total = (len(test0) + len(test1))
for vec in test0:
    if dec_fcn(vec, 'test') <= 0:
        TN += 1
    else:
        FN += 1
for vec in test1:
    if dec_fcn(vec, 'test') >= 0:
        TP += 1
    else:
        FP += 1
acc_rate = (TP+TN)/total
```

6. Draw Decision Boundary



```
# dec bdry
if acc_rate >= desired_accuracy:

    for i in range(len(X)):
        for j in range(len(X[0])):
            Z[i,j] = dec_fcn(np.array([X[i,j],Y[i,j]]))
    return Z, conf
```

Lagrangian and QP derivation for Soft Margin

The only difference in the Lagrangian dual problem between soft-margin and hard-margin is a box constraint on the KKT multipliers in soft-margin. The following is the lagrangian:

$$\mathcal{L}(w,b,\xi,\lambda,\alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}\xi_i - \sum_{i=1}^{\ell}\lambda_i\left[y_i(w^T\phi(x_i)-b)+\xi_i-1\right] - \sum_{i=1}^{\ell}\alpha_i\xi_i.$$

And the only change in the KKT conditions from hard-margin comes from

$$\frac{\partial\mathcal{L}}{\partial\xi} = 0 \implies C - \lambda_i - \alpha_i = 0 \tag{15}$$

Now, to maximize over the KKT multipliers λ, α, we can remove remove the dependency on α_i by rewriting the above as

$$0 \leq \lambda_i \leq C.$$

And plugging in $\alpha_i = C - \lambda_i$, we return to the hard margin lagrangian:

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{\ell} \lambda_i \left[y_i(w^T\phi(x_i) - b) - 1\right] \tag{16}$$

$$= \min_{\lambda} \frac{1}{2}\lambda^T K \lambda - 1_{\ell}^T \qquad s.t. \quad 0 \leq \lambda_i \leq C, \quad \sum_i \lambda_i y_i = 0 \tag{17}$$

And the KKT conditions give us that

$$y_i(w^T\phi(x_i) - b) + \xi_i - 1 = 0 \quad \forall \lambda_i \neq 0$$
$$\xi_i = 0 \quad \forall \alpha_i \neq 0$$

Note that from the above and (15):

$$\lambda_i < C \implies \alpha_i \neq 0 \implies \xi_i = 0$$

So we can compute b by averaging all of the solutions to the following:

$$y_i(w^T\phi(x_i) - b) - 1 = 0 \quad \text{if } \epsilon < \lambda_i < C - \epsilon$$

As $C \to \infty$ we approach the hard margin SVM.

Single Class SVM

Lagrangian dual problem:

$$\max_{\lambda,\alpha} \min_{r,b,\xi} L = \max_{\lambda,\alpha} \min_{r,b,\xi} r^2 + C \sum \xi_i - \sum_i \lambda_i \left[r^2 + \xi_i - \|\phi(x_i) - b\|^2 \right] - \sum_{i=1}^{l} \alpha_i \xi_i \tag{18}$$

For r, b to be a minimizer, it must satisfy the KKT conditions:

$$\frac{\partial L}{\partial r} = 0 \implies \sum_i \lambda_i = 1 \tag{19}$$

$$\frac{\partial L}{\partial b} = 0 \implies b = \sum_i \lambda_i \phi(x_i) \tag{20}$$

$$\frac{\partial L}{\partial \xi} = 0 \implies \alpha_i + \lambda_i = C \implies 0 \leq \lambda_i \leq C \tag{21}$$

And complimentary slackness:

$$\alpha_i \xi_i = 0, \qquad \lambda_i \left[r^2 + \xi_i - \|\phi(x_i) - b\|^2 \right] = 0 \tag{22}$$

After plugging in the first three KKT conditions, we arrive at the following quadratic programming problem and can remove α from the problem entirely:

$$\min_{\lambda} \lambda^T K \lambda - \sum_i \lambda_i ker(x_i, x_i) \tag{23}$$

$$s.t. \quad 0 \leq \lambda_i \leq C, \tag{24}$$

$$\sum_i \lambda_i = 1 \tag{25}$$

Where

$$K_{i,j} = ker(x_i, x_j) = ker(x_j, x_i) = K_{j,i}$$

KKT derivation for Single Class w Negative Examples

$$\min_{r,b} r^2 + C_1 \sum \xi_i + C_2 \sum y_i \tag{26}$$

$$s.t \quad \|\phi(x_i) - b\| \leq r^2 + \xi_i \tag{27}$$

$$\|\phi(y_i) - b\| \geq r^2 - \beta_i \tag{28}$$

$$\xi_i \geq 0 \tag{29}$$

$$\beta_i \geq 0 \tag{30}$$

Lagrangian dual problem:

$$\begin{aligned}
&\max_{\lambda,\alpha,\gamma}\min_{r,b,\xi} L \\
&= \max_{\lambda,\alpha,\gamma}\min_{r,b,\xi} r^2 + C_1\sum_{i=1}^{l}\xi_i + C_2\sum_{i=1}^{m}\beta_i \\
&- \sum_i \lambda_i\left[r^2 + \xi_i - \|\phi(x_i) - b\|^2\right] - \sum p_i\left[-r^2 + \beta_i + \|\phi(y_i) - b\|^2\right] \\
&- \sum_{i=1}^{l}\alpha_i\xi_i - \sum_{i=1}^{m}\gamma_i\beta_i
\end{aligned} \tag{31}$$

For r, b to be a minimizer, it must satisfy the KKT conditions:

$$\frac{\partial L}{\partial r} = 0 \implies \sum_{i=1}^{l} \lambda_i - \sum_{i=1}^{m} p_i = 1 \tag{32}$$

$$\frac{\partial L}{\partial b} = 0 \implies b = \sum_{i=1}^{l} \lambda_i \phi(x_i) - \sum_{i=1}^{m} p_i \phi(y_i) \tag{33}$$

$$\frac{\partial L}{\partial \xi} = 0 \implies \alpha_i + \lambda_i = C_1 \implies 0 \leq \lambda_i \leq C_1 \tag{34}$$

$$\frac{\partial L}{\partial \beta} = 0 \implies \gamma_i + p_i = C_2 \implies 0 \leq p_i \leq C_2 \tag{35}$$

And complimentary slackness:

$$\alpha_i \xi_i = 0 \tag{36}$$

$$\gamma_i \beta_i = 0 \tag{37}$$

$$p_i \left[-r^2 + \beta_i + \|\phi(y_i) - b\|^2 \right] = 0 \tag{38}$$

$$\lambda_i \left[r^2 + \xi_i - \|\phi(x_i) - b\|^2 \right] = 0 \tag{39}$$

For the sake of notation, we convert to using only λ and x: let

$$\lambda_{l+j} = p_j, \quad x_{l+j} = y_j \qquad \text{for } j = 1 : m.$$

After plugging in the first three KKT conditions, we arrive at the following quadratic programming problem and can remove α, γ from the problem entirely:

$$\min_{\lambda} \lambda^T Q\lambda - \sum_{i=1}^{l} \lambda_i ker(x_i, x_i) + \sum_{i=l+1}^{l+m} \lambda_i ker(x_i, x_i) \tag{40}$$

$$s.t. \quad \sum_{i=1}^{l} \lambda_i - \sum_{i=l+1}^{l+m} \lambda_i = 1 \tag{41}$$

$$0 \leq \lambda_i \leq C_1 \text{ for } i = 1 : l \tag{42}$$

$$0 \leq \lambda_i \leq C_2 \text{ for } i = l+1 : l+m \tag{43}$$

Where

$$Q_{i,j} = ker(x_i, x_j) = ker(x_j, x_i) = Q_{j,i} \text{ for } i = 1 : l, j = 1 : l$$

$$Q_{i,j} = ker(x_i, x_j) = ker(x_j, x_i) = Q_{j,i} \text{ for } i = l+1 : l+m, j = l+1 : l+m$$

$$Q_{i,j} = -ker(x_i, x_j) = -ker(x_j, x_i) = Q_{j,i} \text{ for } i = 1 : l, j = l+1 : l+m$$

QP Solvers

1. DAQP: Dual active set solver for embedded quadratic programming [Arnstrom2022: daqp]
2. ECOS: Interior-point solver for second-order cone programming (SOCP), ie convex quadratically constrained problems. [Domahidi2013: ecos]
3. OSQP: Operator splitting solver for quadratic programs [Stellato2020: osqp]

DAQP and ECOS are both faster than OSQP. DAQP is more catered to our QP, and both are more accurate than OSQP. However, they require $X^T X$ to be strictly positive definite. In the linear case, X is often not invertible, especially when the number of data points is larger than the dimension. When using the kernel, X is almost always invertible and DAQP or ECOS are most ideal.

Active Set Method

- Simplex is an active set method
- Idea:
  - Guess the set of active constraints
  - Reduce the number of unknowns
  - Solve an unconstrained subproblem

How to guess at the active set?

Dual Active Set Solver for QPs

General formulation of a QP:

$$\min_x \frac{1}{2} x^T H x + f^T x$$

$$s.t. \quad Ax \leq b$$

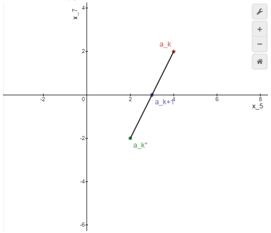
Dual problem:

$$\min_{\alpha \geq 0} \frac{1}{2} \alpha^T M M^T \alpha + d^T \alpha = \min_\alpha D$$

Maintain complimentary slackness ($\alpha_i \neq 0 \iff [Ax]_i = b_i$), aim for dual feasibility and primal feasibility.

1: Dual Feasibility

- At iteration k, working set of indices w_k, solve the new unconstrained problem on those indices. Since this is quadratic, we can take the gradient and solve a linear system:
$$\nabla D_{w_k}(\alpha^*_{w_k}) = 0$$
- If $\alpha^*_{w_k} < 0$, for some index, we do a line search (towards dual feasibility):



- Until 1 component of α^*_k becomes nonnegative, and we remove that index from the active set.

2: Primal Feasibility

- Check primal feasibility by checking if the gradient of the *inactive* constraints is positive:

$$\nabla D_{\overline{w}_k} \geq 0$$

- Add the most negative component of this term to the active set.

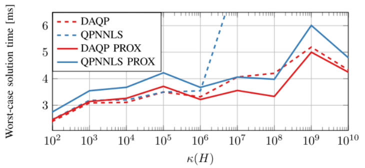
LDL^T Decompositions

- Solving $\nabla D_{w_k} = 0$ means solving a linear system involving

$$M_k M_k^T = LDL^T$$

- Between steps and iterations, only 1 index is added or removed to the active set i.e. only 1 row is added or removed from M_k
- Easy to update LDL^T recursively.

Results



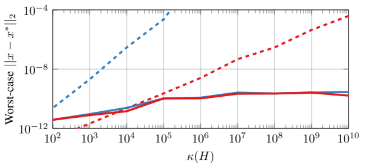


Figure: Comparison with QPNNLS on QPs with varying condition number, $n = 25$ $m = 100$.

Results II

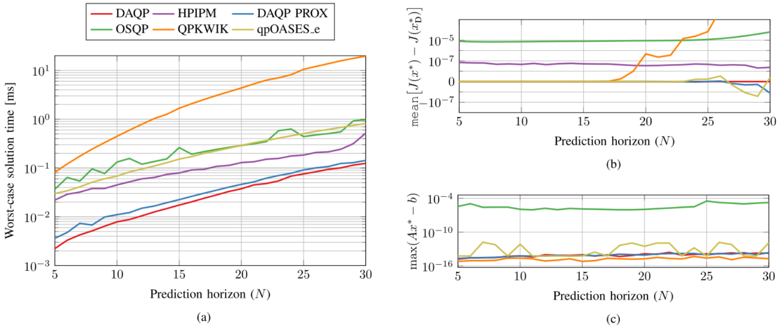


Figure: Comparison with other solvers as QPs grow over time. $n = 2N + 1, m = N + 2(N - 1)$. One of the limitiations with DAQP is that it doesn't scale well with large QPs (order $n = 1000+$).

Results III

Thanks!