Machine Learning HW4 – Support Vector Machine

NCTU EE 0310115 鍾文玉, 0310128 游騰杰

Programming language: Python

Environment: Python 2.7.13 under Homebrew @ Mac OS X 10.11.6

Python module used: numpy, pandas, scikit-learn

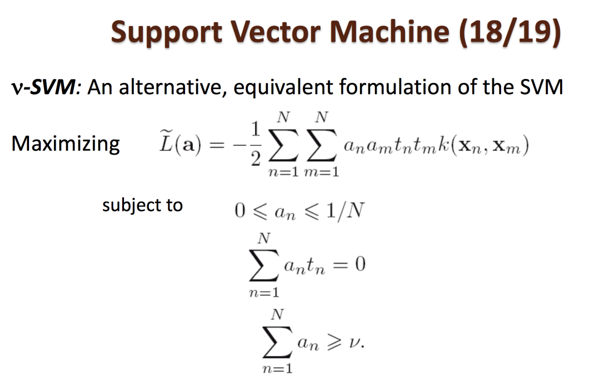
Plot: Matlab

1. ν-SVM

Implement the ν-SVM models with the different kernel types below

- **Linear function (linear)**
- **Polynomial function (poly, with degree = 2,3,4)**
- **Radial basis function (rbf)**

And compare the performance between them.

<table>
<tr><th>Reference:
(Lecture Notes) Kernel Methods P. 30</th><th>Implementation</th></tr>
<tr><td>


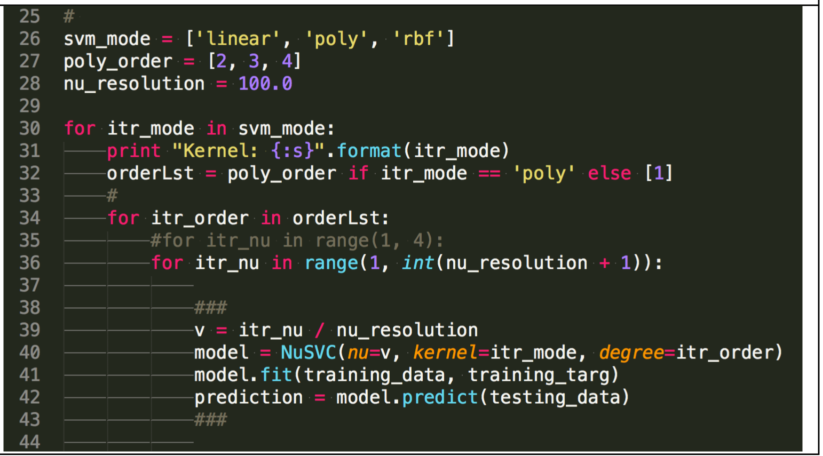
Support Vector Machine (18/19)

ν-SVM: An alternative, equivalent formulation of the SVM

Maximizing $\widetilde{L}(\mathbf{a}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m)$

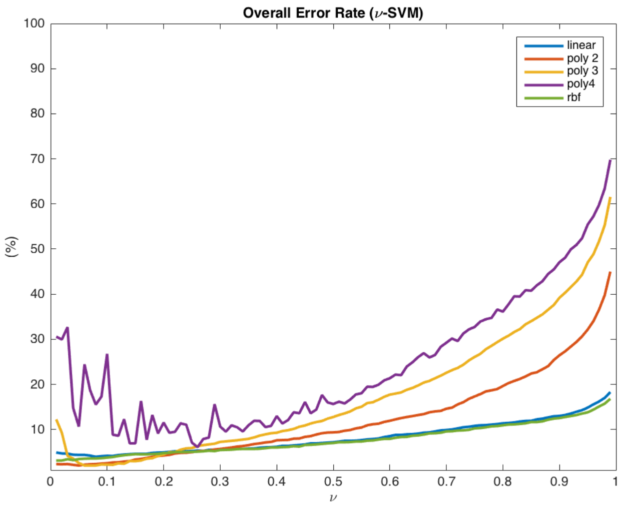
subject to $0 \leqslant a_n \leqslant 1/N$

$\sum_{n=1}^{N} a_n t_n = 0$

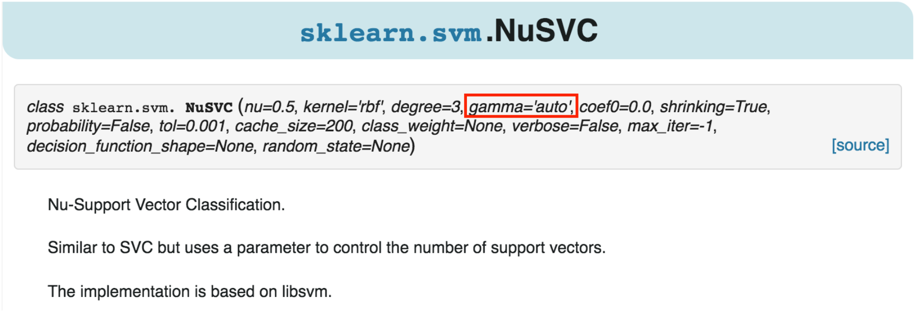
$\sum_{n=1}^{N} a_n \geqslant \nu.$
</td><td>


```
#
svm_mode = ['linear', 'poly', 'rbf']
poly_order = [2, 3, 4]
nu_resolution = 100.0

for itr_mode in svm_mode:
    print "Kernel: {:s}".format(itr_mode)
    orderLst = poly_order if itr_mode == 'poly' else [1]
    #
    for itr_order in orderLst:
        #for itr_nu in range(1, 4):
        for itr_nu in range(1, int(nu_resolution + 1)):

            ###
            v = itr_nu / nu_resolution
            model = NuSVC(nu=v, kernel=itr_mode, degree=itr_order)
            model.fit(training_data, training_targ)
            prediction = model.predict(testing_data)
            ###

```
</td></tr>
</table>

From the performance versus parameter **ν** (0.01 to 0.99) plot shown on the right, it is obvious the performance of Radial Basis Function (RBF) and Linear Function have stable, overall great performance, and also similar performance for this dataset over the entire **ν** scale. However, polynomials of order = 3 and 4 shown very low accuracy at low **ν**.



Additionally, for polynomial kernel function of order 4, the accuracy oscillates massively for **ν** < 0.3. It can be inferred that in the case we are currently dealing with, high order polynomial kernel function may not be a great choice. To be precise, the acceptable (error rate < 10%) order is 1 (a.k.a. Linear Function,) 2, and 3 (for **ν** > 0.01.)



sklearn.svm.NuSVC

class sklearn.svm. **NuSVC** (*nu=0.5, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None*) [source]

Nu-Support Vector Classification.

Similar to SVC but uses a parameter to control the number of support vectors.

The implementation is based on libsvm.

Reference:
sklearn.svm.NuSVC — scikit-learn 0.18.1 documentation
http://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html

Gamma in **ν**-SVM

We set gamma in **ν**-SVM to auto in the previous experiment, which is 1/N (N: number of features, which is 784 in this case,) according to the information we found, larger gamma leads to higher bias and lower variance models, and vice-versa. The trade-offs between bias and variance can be illustrate below, which shows how these two parameters affect the model.

Gamma is a case dependent parameter, and there’ s no easy formula or a certain pattern we can follow to find the best gamma, only by exhausting the possible range of gamma can we find a better one, so here we tried one gamma for discussion.

The Bias-Variance Decomposition (3/3)

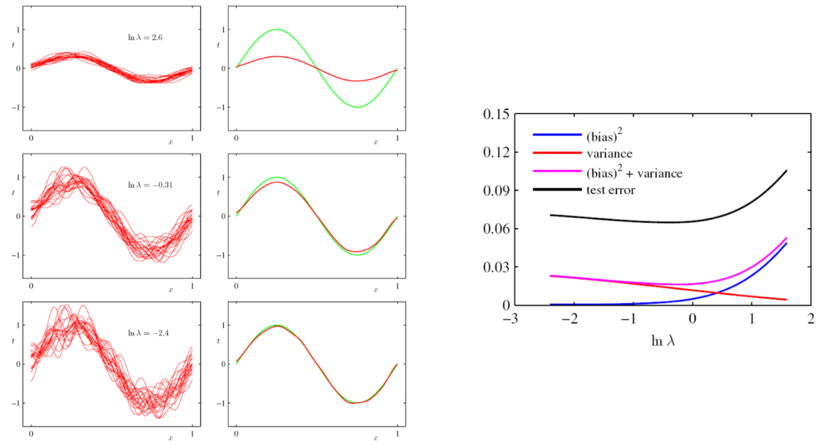
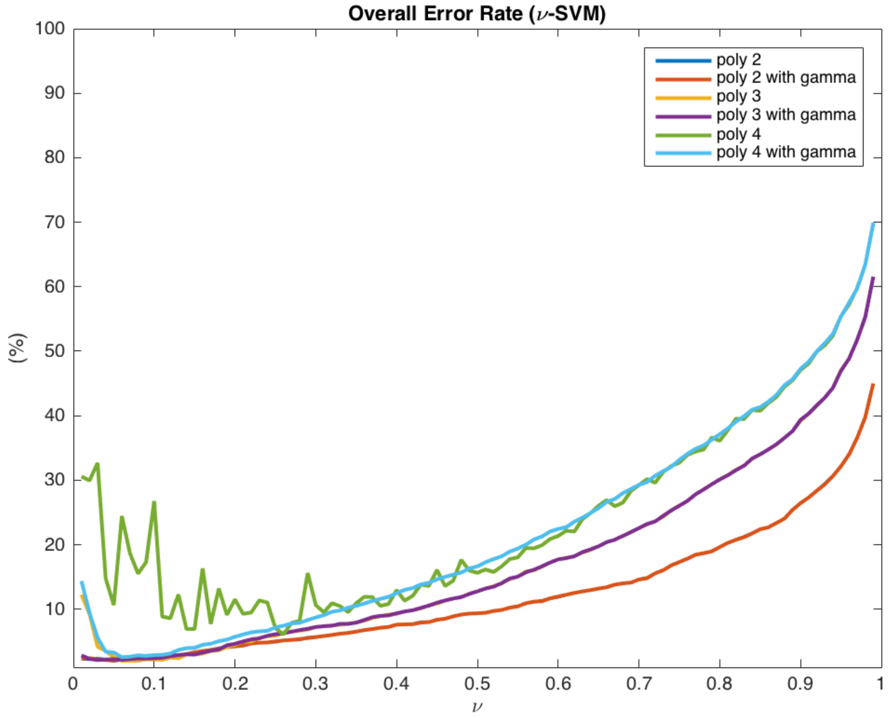


Figure 3.5 Illustration of the dependence of bias and variance on model complexity, governed by a regularization parameter λ, using the sinusoidal data set from Chapter 1. There are $L = 100$ data sets, each having $N = 25$ data points, and there are 24 Gaussian basis functions in the model so that the total number of parameters is $M = 25$ including the bias parameter. The left column shows the result of fitting the model to the data sets for various values of $\ln\lambda$ (for clarity, only 20 of the 100 fits are shown). The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green).

2017 NCTU EE
Linear Models for Regression

Reference: (Lecture Note) Linear Model for Regression P. 22



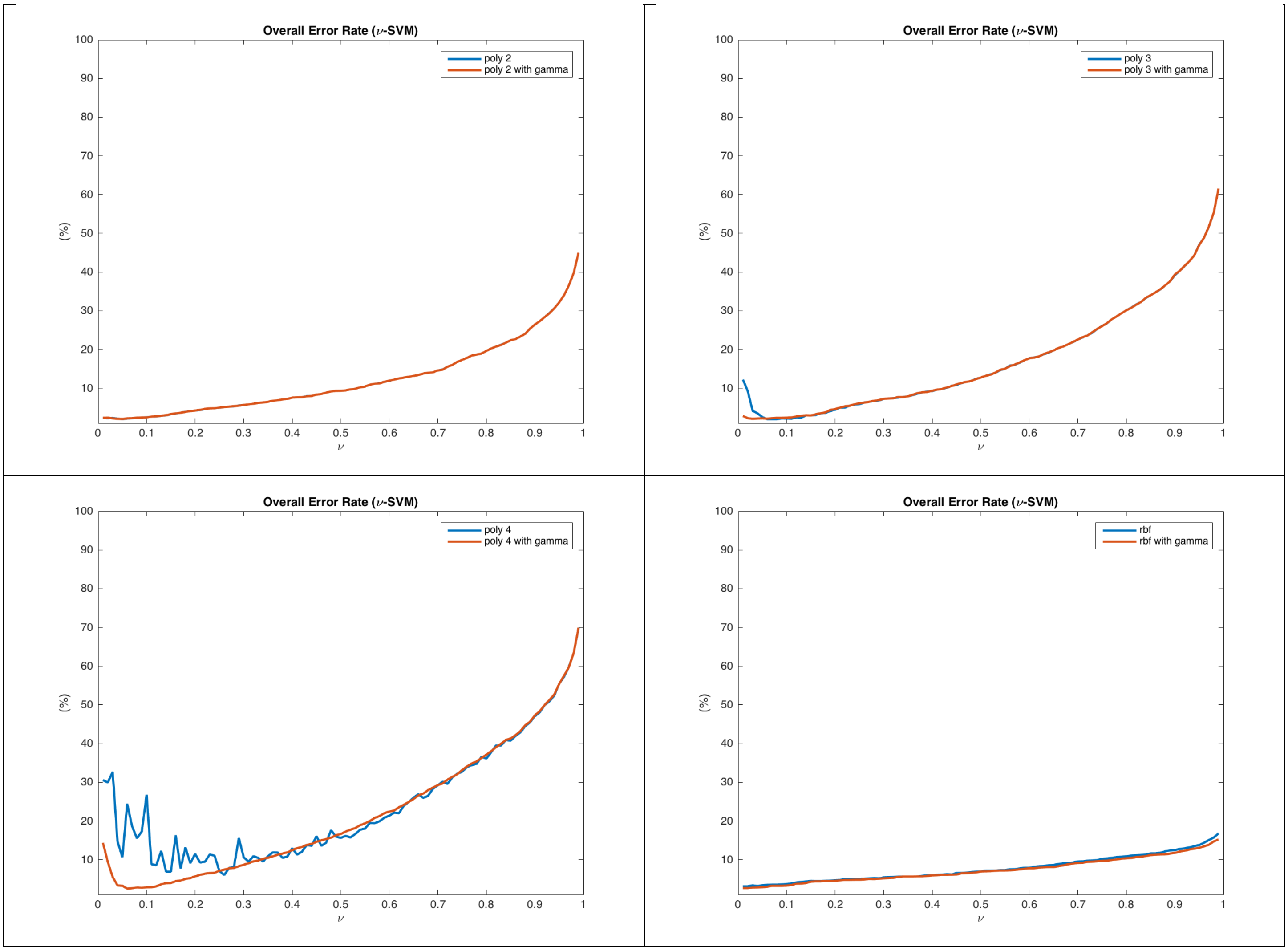
The chart above shows the error rate versus **ν** for polynomial kernel functions of order 2, 3, and 4. The improvement between the one with and without tuning gamma is obvious, especially for the case of polynomial kernel functions of order 4.

The suppression on the curve is surprising, with gamma manually adjusted on the model, the error rate curve of polynomial kernel functions of order 4 resembles to the one of order 3 without tuning gamma. Additionally, the polynomial kernel functions of order 3 with adjusted gamma shown very close curve to the one of order 2 without tuning gamma.

However, the difference between model of polynomial kernel functions of order 2 with and without tuning gamma is not so obvious, but that doesn’t mean the gamma has no effect on the model in this case for sure, perhaps the gamma in this case is just not the optimal one.

Lastly, for the radial basis kernel function (rbf,) the effect of adjusting gamma makes almost no change on the model. The side-by-side comparison between the tuned model and un-tuned model of the error rate versus **ν** curve for each kernel function is shown in the preceding page, with tuning gamma (392^{-1}) to be twice of the auto ones (784^{-1}.)

Model: $\boldsymbol{\nu}$ –SVM with kernel = rbf, $\boldsymbol{\nu} = 0.3$, $\boldsymbol{\gamma} = 392.0^{-1}$



Reference:

Support Vector Machine 簡介

http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/SVM2.pdf

What are C and gamma with regards to a support vector machine? - Quora

https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine

2. C-SVM

Implement the C-SVM models with the different kernel types below

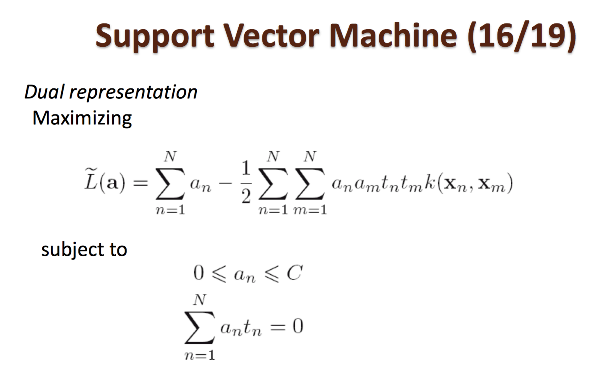
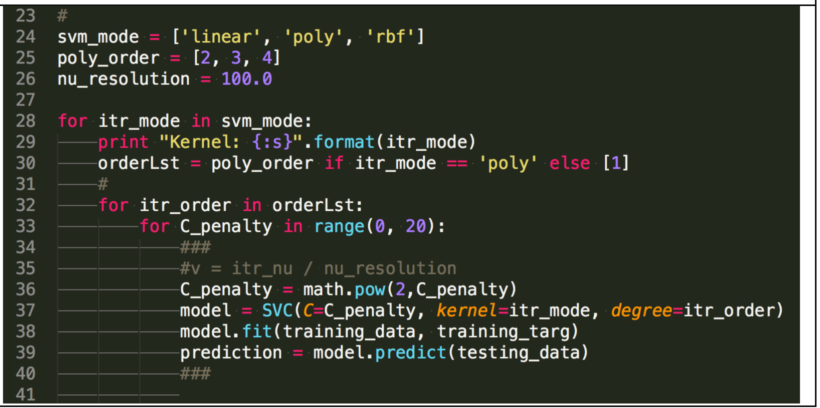
- Linear function (linear)

- Polynomial function (poly, with degree = 2,3,4)

- Radial basis function (rbf)

And compare the performance between them.

If you design the ν properly, you might get the same outcomes with Task 1.

Reference: (Lecture Notes) Kernel Methods P. 28	Implementation
	

Support Vector Machine (16/19)

Dual representation

Maximizing

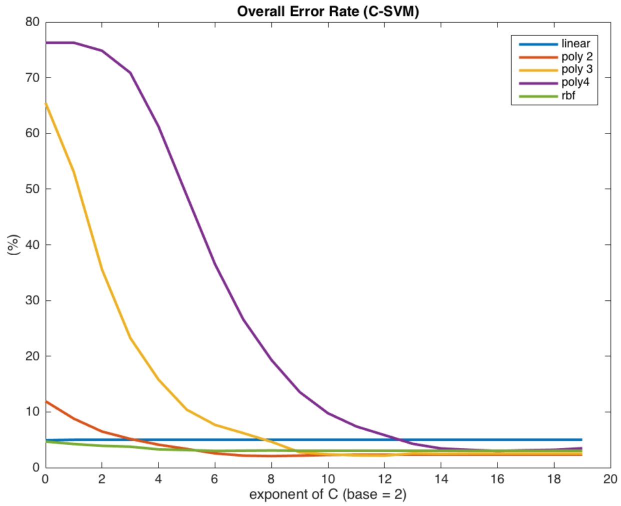
$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to

$$0 \leqslant a_n \leqslant C$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

```
#
svm_mode = ['linear', 'poly', 'rbf']
poly_order = [2, 3, 4]
nu_resolution = 100.0

for itr_mode in svm_mode:
    print "Kernel: {:s}".format(itr_mode)
    orderLst = poly_order if itr_mode == 'poly' else [1]
    #
    for itr_order in orderLst:
        for C_penalty in range(0, 20):
            ###
            #v = itr_nu / nu_resolution
            C_penalty = math.pow(2,C_penalty)
            model = SVC(C=C_penalty, kernel=itr_mode, degree=itr_order)
            model.fit(training_data, training_targ)
            prediction = model.predict(testing_data)
            ###

```



From the performance versus parameter c (penalty of misclassification) shown on the right, as c gets larger, the error rate becomes lower for polynomials kernel functions and rbf kernel function, as for the linear kernel function, the error rate is fixed at around 5%. While c gets larger, the decision hyperplane would try to fit more to the data.

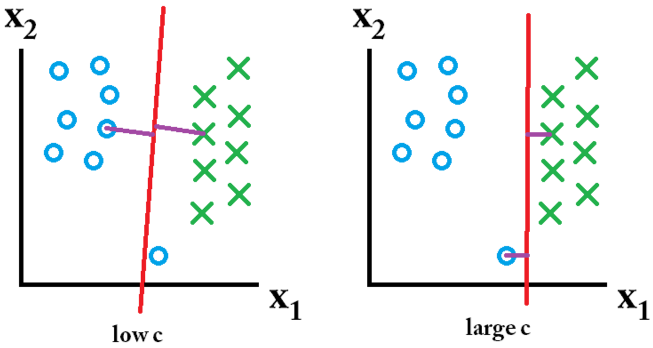
We adjust the parameter c with exponential of base 2, namely, 2^n. When n is large enough (2^{19},) the error rate reaches 2% but never lower than 2%. After doing some research, we infer that the model has become over-fitting to some extent due to the large penalty.

For large values of C, C-SVM will choose a smaller-margin hyperplane if that hyperplane classifies all the training points correctly. Conversely, for very small value of C, the model would try to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we would get misclassified even if our training data is linearly separable.

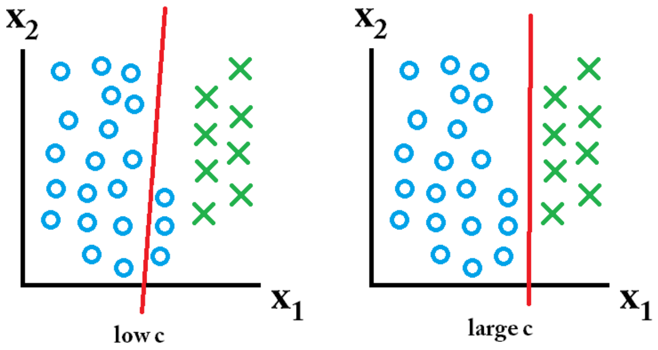
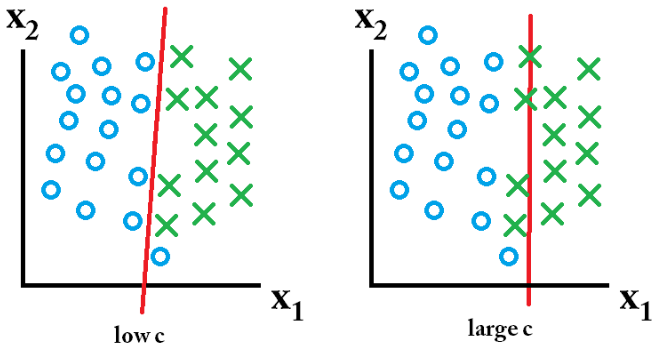
Case discussion / illustration:

In a SVM we are searching for two things: A hyperplane with the largest minimum margin; a hyperplane that correctly separates as many instances as possible. The c parameter in C-SVM determines how great our desire is for the latter one.

As illustrated on right, a lower c (left) which gives us a large minimum margin (purple,) however, this requires that we neglect the outlier (blue circle) that we have failed to classify correctly. For a higher c (right,) we do not neglect the outlier and thus ended up with a much smaller margin.



If the future data comes in the preceding scenarios:

If the distribution of the data is as shown on the right, then the classifier learned using a larger c value is better.	
If the distribution of the data is as shown on the right, then the classifier learned using a lower c value is better.	

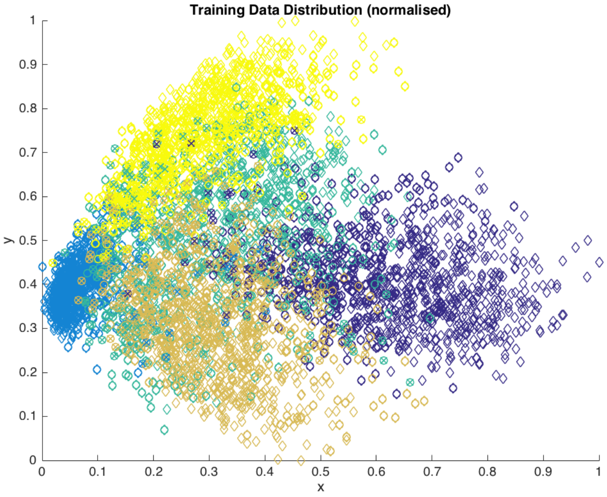
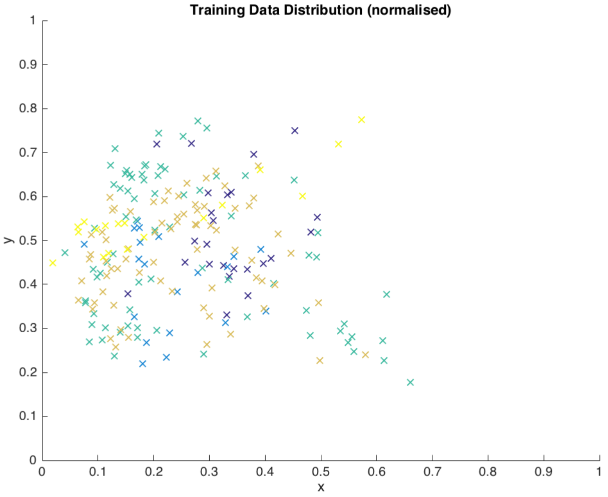
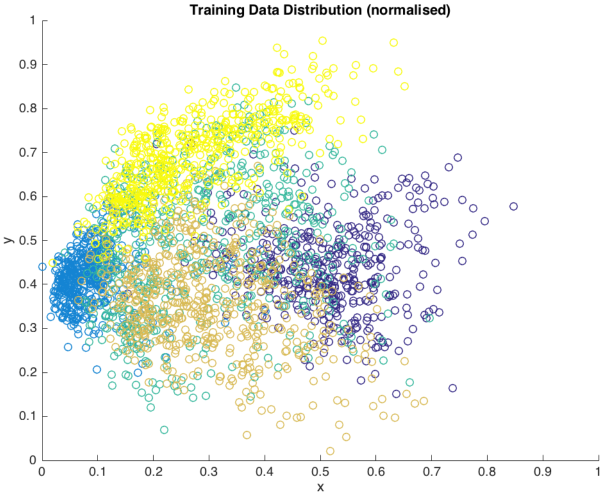
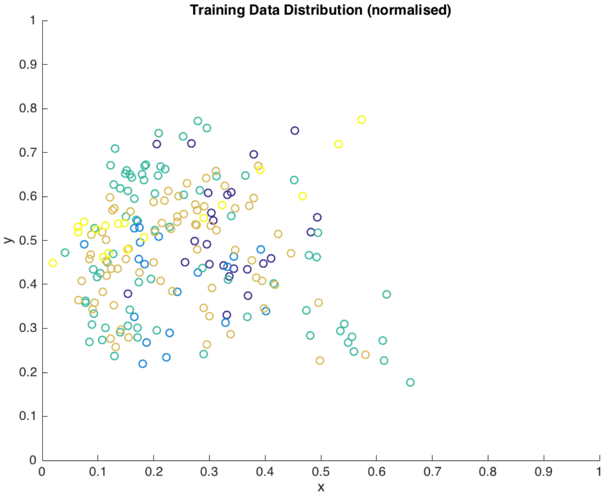
Reference:

machine learning - What is the influence of C in SVMs with linear kernel? - Cross Validated
https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

3. Supporting Vectors

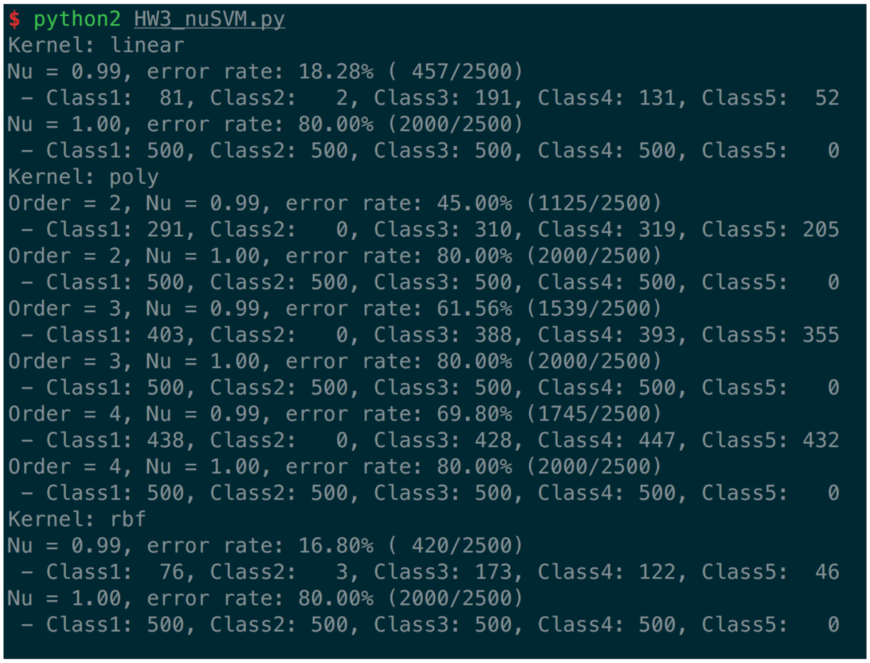
Choose one of the models you’ ve trained to do analysis and find those “supporting vectors” and “outliers” for this model. You should use PCA to map data down to two dimensions.
Please plot at least 30 samples for each and discuss your observations.

Model: $\boldsymbol{\nu}$ –SVM with kernel $= \text{rbf}$, $\boldsymbol{\nu} = 0.3$

All (5000 samples)	Outliers (208 samples)
	
Support Vectors (2612 samples)	**Outliers && Support Vectors (208 samples)**
	

Observation:

1. When **ν** increases from 0.99 to 1.00, the sudden jump in error rate.



```
$ python2 HW3_nuSVM.py
Kernel: linear
Nu = 0.99, error rate: 18.28% ( 457/2500)
 - Class1:  81, Class2:   2, Class3: 191, Class4: 131, Class5:  52
Nu = 1.00, error rate: 80.00% (2000/2500)
 - Class1: 500, Class2: 500, Class3: 500, Class4: 500, Class5:   0
Kernel: poly
Order = 2, Nu = 0.99, error rate: 45.00% (1125/2500)
 - Class1: 291, Class2:   0, Class3: 310, Class4: 319, Class5: 205
Order = 2, Nu = 1.00, error rate: 80.00% (2000/2500)
 - Class1: 500, Class2: 500, Class3: 500, Class4: 500, Class5:   0
Order = 3, Nu = 0.99, error rate: 61.56% (1539/2500)
 - Class1: 403, Class2:   0, Class3: 388, Class4: 393, Class5: 355
Order = 3, Nu = 1.00, error rate: 80.00% (2000/2500)
 - Class1: 500, Class2: 500, Class3: 500, Class4: 500, Class5:   0
Order = 4, Nu = 0.99, error rate: 69.80% (1745/2500)
 - Class1: 438, Class2:   0, Class3: 428, Class4: 447, Class5: 432
Order = 4, Nu = 1.00, error rate: 80.00% (2000/2500)
 - Class1: 500, Class2: 500, Class3: 500, Class4: 500, Class5:   0
Kernel: rbf
Nu = 0.99, error rate: 16.80% ( 420/2500)
 - Class1:  76, Class2:   3, Class3: 173, Class4: 122, Class5:  46
Nu = 1.00, error rate: 80.00% (2000/2500)
 - Class1: 500, Class2: 500, Class3: 500, Class4: 500, Class5:   0
```

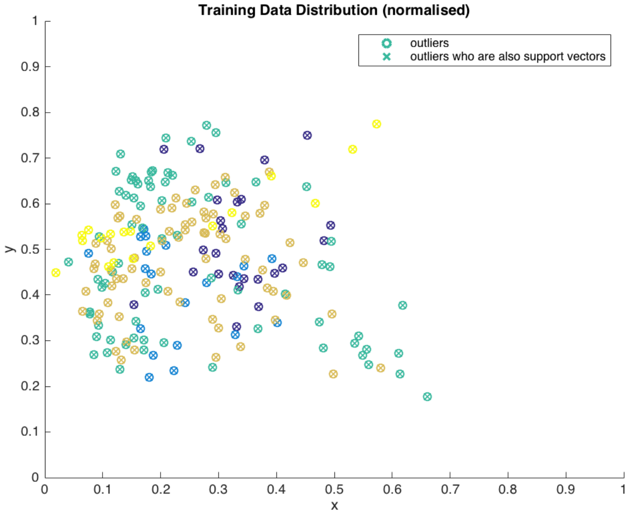
Starting with the interpretation of **ν**, the parameter **ν** is an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. In our case above, we set **ν** to 0.30 we are guaranteed to find at most 30% (≤ 1500 samples) of your training examples being misclassified (208 samples,) at the cost of a small margin, though, and at least 30% (≥ 1500 samples) of your training examples being support vectors (2612 samples.)

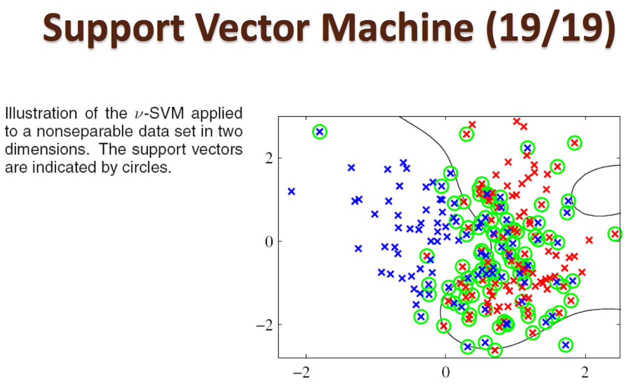
If **ν** is set to 1.00, which in turn means we are guaranteed to find at most 100% of training examples being misclassified and at least 100% of the training examples being the support vectors. The above statement shows that every training data become support vectors, and the margin between the class in the model becomes the entire feature space, i.e. the entire feature space is occupied by one class and the rest classes has infinitesimal region in the feature space, which meet our result in the above, when **ν** equals 1.00, the model will classify every sample to be one of the class.

Reference:

What is the meaning of the nu parameter in Scikit-Learn's SVM class? - Stack Overflow
http://stackoverflow.com/questions/11230955/what-is-the-meaning-of-the-nu-parameter-in-scikit-learns-svm-class

2. Every outlier is also a support vector.





Support Vector Machine (19/19)

Illustration of the ν-SVM applied to a nonseparable data set in two dimensions. The support vectors are indicated by circles.

Support vectors are samples that are within the margin, some of them are correctly classified and some of them are misclassified, namely, outliers.

Reference: (Lecture Notes) Kernel Method P. 31