# CSE471: Statistical Methods in AI

## Assignment 4: SVM, Kernel Methods

Abhinav Moudgil [201331039]

## Contents



## Non linear Fisher's LDA derivation

In statistics, kernel Fisher discriminant analysis (KFD), also known as generalized discriminant analysis and kernel discriminant analysis, is a kernelized version of linear discriminant analysis. It is named after Ronald Fisher. Using the kernel trick, LDA is implicitly performed in a new feature space, which allows non-linear mappings to be learned.

**Linear Discriminant Analysis**

Intuitively, the idea of LDA is to find a projection where class separation is maximized. Given two sets of labeled data, $\mathbf{C}_1$ and $\mathbf{C}_2$, define the class means $\mathbf{m}_1$ and $\mathbf{m}_2$ to be

$$\mathbf{m}_i = \frac{1}{l_i}\sum_{n=1}^{l_i}\mathbf{x}_n^i,$$

where $l_i$ is the number of examples of class $\mathbf{C}_i$. The goal of linear discriminant analysis is to give a large separation of the class means while also keeping the in-class variance small. This is formulated as maximizing

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathsf{T}}\mathbf{S}_B\mathbf{w}}{\mathbf{w}^{\mathsf{T}}\mathbf{S}_W\mathbf{w}},$$

where $\mathbf{S}_B$ is the between-class covariance matrix and $\mathbf{S}_W$ is the total within-class covariance matrix:

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^{\mathsf{T}}$$

$$\mathbf{S}_W = \sum_{i=1,2}\sum_{n=1}^{l_i}(\mathbf{x}_n^i - \mathbf{m}_i)(\mathbf{x}_n^i - \mathbf{m}_i)^{\mathsf{T}}.$$

Differentiating $J(\mathbf{w})$ with respect to $\mathbf{w}$, setting equal to zero, and rearranging gives

$$(\mathbf{w}^{\mathsf{T}}\mathbf{S}_B\mathbf{w})\mathbf{S}_W\mathbf{w} = (\mathbf{w}^{\mathsf{T}}\mathbf{S}_W\mathbf{w})\mathbf{S}_B\mathbf{w}.$$

Since we only care about the direction of $\mathbf{w}$ and $\mathbf{S}_B\mathbf{w}$ has the same direction as $(\mathbf{m}_2-\mathbf{m}_1)$, $\mathbf{S}_B\mathbf{w}$ can be replaced by $(\mathbf{m}_2-\mathbf{m}_1)$ and we can drop the scalars $(\mathbf{w}^{\mathsf{T}}\mathbf{S}_B\mathbf{w})$ and $(\mathbf{w}^{\mathsf{T}}\mathbf{S}_W\mathbf{w})$ to give

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1).$$

**Kernel trick with LDA**

To extend LDA to non-linear mappings, the data can be mapped to a new feature space, F, via some function $\phi$. In this new feature space, the function that needs to be maximized is

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathsf{T}}\mathbf{S}_B^{\phi}\mathbf{w}}{\mathbf{w}^{\mathsf{T}}\mathbf{S}_W^{\phi}\mathbf{w}},$$

where

$$\begin{aligned}
\mathbf{S}_B^{\phi} &= (\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})^{\mathsf{T}} \\
\mathbf{S}_W^{\phi} &= \sum_{i=1,2}\sum_{n=1}^{l_i}(\phi(\mathbf{x}_n^i) - \mathbf{m}_i^{\phi})(\phi(\mathbf{x}_n^i) - \mathbf{m}_i^{\phi})^{\mathsf{T}},
\end{aligned}$$

and

$$\mathbf{m}_i^{\phi} = \frac{1}{l_i}\sum_{j=1}^{l_i}\phi(\mathbf{x}_j^i).$$

Further, note that $\mathbf{w} \in F$. Explicitly computing the mappings $\phi(\mathbf{x}_i)$ and then performing LDA can be computationally expensive, and in many cases intractable. For example, F may be infinitely dimensional. Thus, rather than explicitly mapping the data to F, the data can be implicitly embedded by rewriting the algorithm in terms of dot products and using the kernel trick in which the dot product in the new feature space is replaced by a kernel function, $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$.

LDA can be reformulated in terms of dot products by first noting that $\mathbf{w}$ will have an expansion of the form[5]

$$\mathbf{w} = \sum_{i=1}^{l}\alpha_i\phi(\mathbf{x}_i).$$

Then note that

$$\mathbf{w}^{\mathsf{T}}\mathbf{m}_i^{\phi} = \frac{1}{l_i}\sum_{j=1}^{l}\sum_{k=1}^{l_i}\alpha_j k(\mathbf{x}_j, \mathbf{x}_k^i) = \alpha^{\mathsf{T}}\mathbf{M}_i,$$

where

$$(\mathbf{M}_i)_j = \frac{1}{l_i}\sum_{k=1}^{l_i} k(\mathbf{x}_j, \mathbf{x}_k^i).$$

The numerator of $J(\mathbf{w})$ can then be written as:

$$\begin{aligned}\mathbf{w}^{\mathsf{T}}\mathbf{S}_B^{\phi}\mathbf{w} &= \mathbf{w}^{\mathsf{T}}(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})^{\mathsf{T}}\mathbf{w} \\ &= \alpha^{\mathsf{T}}\mathbf{M}\alpha,\end{aligned}$$

where $\mathbf{M} = (\mathbf{M}_2 - \mathbf{M}_1)(\mathbf{M}_2 - \mathbf{M}_1)^{\mathsf{T}}$. Similarly, the denominator can be written as

$$\mathbf{w}^{\mathsf{T}}\mathbf{S}_W^{\phi}\mathbf{w} = \alpha^{\mathsf{T}}\mathbf{N}\alpha,$$

where $\mathbf{N} = \sum_{j=1,2}\mathbf{K}_j(\mathbf{I} - \mathbf{1}_{l_j})\mathbf{K}_j^{\mathsf{T}}$, with the $n^{\text{th}}$, $m^{\text{th}}$ component of $\mathbf{K}_j$ defined as $k(\mathbf{x}_n, \mathbf{x}_m^j)$, $\mathbf{I}$ is the identity matrix, and $\mathbf{1}_{l_j}$ the matrix with all entries equal to $1/l_j$. This identity can be derived by starting out with the expression for $\mathbf{w}^{\mathsf{T}}\mathbf{S}_W^{\phi}\mathbf{w}$ and using the expansion of $\mathbf{w}$ and the definitions of $\mathbf{S}_W^{\phi}$ and $\mathbf{m}_i^{\phi}$

$$\begin{aligned}\mathbf{w}^{\mathsf{T}}\mathbf{S}_W^{\phi}\mathbf{w} &= \left(\sum_{i=1}^{l}\alpha_i\phi^{\mathsf{T}}(\mathbf{x}_i)\right)\left(\sum_{j=1,2}\sum_{n=1}^{l_j}(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})^{\mathsf{T}}\right)\left(\sum_{k=1}^{l}\alpha_k\phi(\mathbf{x}_k)\right) \\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\alpha_i\phi^{\mathsf{T}}(\mathbf{x}_i)(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})^{\mathsf{T}}\alpha_k\phi(\mathbf{x}_k) \\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i k(\mathbf{x}_i, \mathbf{x}_n^j) - \frac{1}{l_j}\sum_{p=1}^{l_j}\alpha_i k(\mathbf{x}_i, \mathbf{x}_p^j)\right)\left(\alpha_k k(\mathbf{x}_k, \mathbf{x}_n^j) - \frac{1}{l_j}\sum_{q=1}^{l_j}\alpha_k k(\mathbf{x}_k, \mathbf{x}_q^j)\right) \\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_n^j)\right.\right. \\
&\quad \left.\left. - \frac{2\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_p^j) + \frac{\alpha_i\alpha_k}{l_j^2}\sum_{p=1}^{l_j}\sum_{q=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_p^j)k(\mathbf{x}_k, \mathbf{x}_q^j)\right)\right) \\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_n^j) - \frac{\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_p^j)\right)\right) \\
&= \sum_{j=1,2}\alpha^{\mathsf{T}}\mathbf{K}_j\mathbf{K}_j^{\mathsf{T}}\alpha - \alpha^{\mathsf{T}}\mathbf{K}_j\mathbf{1}_{l_j}\mathbf{K}_j^{\mathsf{T}}\alpha \\
&= \alpha^{\mathsf{T}}\mathbf{N}\alpha.\end{aligned}$$

With these equations for the numerator and denominator of $J(\mathbf{w})$, the equation for $J$ can be rewritten as

$$J(\alpha) = \frac{\alpha^\mathsf{T}\mathbf{M}\alpha}{\alpha^\mathsf{T}\mathbf{N}\alpha}.$$

Then, differentiating and setting equal to zero gives

$$(\alpha^\mathsf{T}\mathbf{M}\alpha)\mathbf{N}\alpha = (\alpha^\mathsf{T}\mathbf{N}\alpha)\mathbf{M}\alpha.$$

Since only the direction of $\mathbf{w}$, and hence the direction of $\alpha$, matters, the above can be solved for $\alpha$ as

$$\alpha = \mathbf{N}^{-1}(\mathbf{M}_2 - \mathbf{M}_1).$$

Note that in practice, $\mathbf{N}$ is usually singular and so a multiple of the identity is added to it

$$\mathbf{N}_\epsilon = \mathbf{N} + \epsilon\mathbf{I}.$$

Given the solution for $\alpha$, the projection of a new data point is given by

$$y(\mathbf{x}) = (\mathbf{w} \cdot \phi(\mathbf{x})) = \sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}).$$

## Kernel PCA and LDA

**Datasets**:
ARCENE[1]
It was obtained by merging three mass-spectrometry datasets to obtain enough training and test data for a benchmark. The original features indicate the abundance of proteins in human sera having a given mass value. Based on those features one must separate cancer patients from healthy patients. Distractor features called *probes* were added having no predictive power. The order of the features and patterns were randomized.

MADELON[2]
It is an artificial dataset containing data points grouped in 32 clusters placed on the vertices of a five dimensional hypercube and randomly labeled +1 or -1. The five dimensions constitute 5 informative features. 15 linear combinations of those features were added to form a set of 20 (redundant) informative features. Distractor feature called *probes* having no predictive power were added. The order of the features and patterns were randomized.

| Dataset | Instances | Real features | Probes | Datatype |
|---|---|---|---|---|
| Arcene | 900 | 7000 | 3000 | Real |
| Madelon | 4400 | 20 | 480 | Real |

**Kernel Principal Component Analysis**
Kernel PCA reduces the dimensions *without* taking into account the separation of classes. It picks up the top k dimensions with maximum variance, which facilitates

separation of classes.
The reduction procedure is explained below:
1. Kernel (similarity) matrix is computed. Following are the two kernels matrices computed in this assignment:
RBF Kernel:

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left( - \gamma \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 \right)$$

Linear Kernel:

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j^T \rangle$$

$\gamma$ is taken 15.
2. Since it is not guaranteed that the kernel matrix is centered, we can apply the following equation to do so:

$$K' = K - \mathbf{1}_\mathrm{N} K - K \mathbf{1}_\mathrm{N} + \mathbf{1}_\mathrm{N} K \mathbf{1}_\mathrm{N}$$

where $\mathbf{1}_\mathrm{N}$ is a $NxN$ matrix with all values equal to $1/N$.
3. Eigenvectors of the centered kernel matrix that correspond to the largest K (=10, 100) eigenvalues are the data points already projected onto the respective principal components.

**Linear Discriminant Analysis (LDA)**
LDA maximizes between class seperation i.e variance along resultant dimension is maximized.
It is done in 5 steps as below:
1. First, mean vector for each class are found.
2. Within class scatter matrix $SW$ and between class scatter matrix $SB$ is found.

$$SW = \sum_{i=1}^{c} \sum_{x=1}^{N} (x - m_i)(x - m_i)^T$$

$$SB = \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T$$

3. Calculate the eigen values and eigen vectors of matrix $SW^{-1}SB$
4. Eigen vector are sorted by decreasing eigen values and first one is picked.
5. New data is found by:

$$NewData = RowFeatureVector.RowDataAdjust$$

**RBF kernel PCA**

```
import numpy as np
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn import svm, preprocessing
```

```
from sklearn.metrics import classification_report as cr
from sklearn.decomposition import KernelPCA

def mergeData(trainData, testData):
    x = np.zeros((trainData.shape[0] + testData.shape[0], trainData.shape[1]))
    x[:trainData.shape[0], :] = trainData
    x[trainData.shape[0]:, :] = testData
    return x

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
    return b

def kPCA(X, gamma, k):
    distances = pdist(X, 'sqeuclidean')
    symmetricDistances = squareform(distances)
    K = exp(-gamma * symmetricDistances)
    N = K.shape[0]
    one_N = np.ones((N,N))/N
    normalizedK = K - one_N.dot(K) - K.dot(one_N)    + one_N.dot(K).dot(one_N)
    eigenValues, eigenVectors = eigh(normalizedK)
    alphas = np.column_stack((eigenVectors[:,-i] for i in range(1,k+1)))
    lambdas = [eigenValues[-i] for i in range(1,k+1)]
    return alphas, lambdas

def project(testData, X, k, gamma, alphas, lambdas):
    Data = np.zeros((testData.shape[0], k))
    for i in xrange(testData.shape[0]):
        Data[i, :] = project_x(testData[i], X, gamma, alphas, lambdas)
    return Data

def project_x(x_new, X, gamma, alphas, lambdas):
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
```

```
    return k.dot(alphas / lambdas)

file = open('arcene_train.data.txt')
X = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)

K = 100
gamma = 15
alphas, lambdas = kPCA(X, gamma, K)
testData = project(testData, X, K, gamma, alphas, lambdas)
```

### Linear kernel PCA

```
import numpy as np
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn import svm, preprocessing
from sklearn.metrics import classification_report as cr

def mergeData(trainData, testData):
    x = np.zeros((trainData.shape[0] + testData.shape[0], trainData.
    shape[1]))
    x[:trainData.shape[0], :] = trainData
    x[trainData.shape[0]:, :] = testData
    return x

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
    return b

def project(testData, X, k, gamma, alphas, lambdas):
    Data = np.zeros((testData.shape[0], k))
```

```
    for i in xrange(testData.shape[0]):
        Data[i, :] = project_x(testData[i], X, gamma, alphas, lambdas)
    return Data

def project_x(x_new, X, gamma, alphas, lambdas):
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
    return k.dot(alphas / lambdas)

def kPCA(X, gamma, k):
    K = X.dot(X.T)
    N = K.shape[0]
    one_N = np.ones((N,N))/N
    normalizedK = K - one_N.dot(K) - K.dot(one_N)   + one_N.dot(K).dot(one_N)
    eigenValues, eigenVectors = eigh(normalizedK)
    alphas = np.column_stack((eigenVectors[:,-i] for i in range(1,k+1)))
    lambdas = [eigenValues[-i] for i in range(1,k+1)]
    return alphas, lambdas

file = open('arcene_train.data.txt')
X = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)

K = 100
gamma = 15
alphas, lambdas = kPCA(X, gamma, K)
testData = project(testData, X, K, gamma, alphas, lambdas)
```

### LDA

```
import numpy as np
from numpy import linalg as LA
import math
from sklearn import preprocessing
from sklearn import svm, preprocessing
from sklearn.metrics import classification_report as cr

def ldaTransform(data):
    C0 = data[data[:, -1] == -1]
    C1 = data[data[:, -1] == 1]
    C0 = C0[:, :-1]
    C1 = C1[:, :-1]
    S0 = np.cov(np.transpose(C0))
    S1 = np.cov(np.transpose(C1))
    SW = S0 + S1
    Mu0 = np.mean(C0, axis = 0)
    Mu1 = np.mean(C1, axis = 0)
    Mu = np.mean(data, axis = 0)
    Mu = Mu[:-1]
    Mu = np.matrix(Mu)
    Mu0 = np.matrix(Mu0)
```

```
    Mu1 = np.matrix(Mu1)
    SB = C0.shape[0] * np.transpose(Mu0 - Mu) * (Mu0 - Mu) + C1.shape
    [0] * np.transpose(Mu1 - Mu) * (Mu1 - Mu)
    Swin = LA.pinv(SW) #costly
    Swin = np.matrix(Swin)
    SwinSB = Swin * SB #costly
    e, v = LA.eig(SwinSB) #costly
    s = np.argsort(e)[::-1]
    v = np.array(v)
    ev = np.zeros(v.shape)
    for i in xrange(e.shape[0]):
        ev[:, i] = v[:, s[i]]
    w = ev[:, 0]
    w = np.matrix(w)
    return w

def project(data, w):
    data = np.matrix(data)
    data = np.transpose(data)
    newData = w * data
    newData = np.transpose(newData)
    newData = np.array(newData)
    return newData

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
    return b

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data

file = open('arcene_train.data.txt')
data = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)
```

```
trainData = addLabels(data, trainLabels)
ev = ldaTransform(trainData)
trainData = trainData[:, :-1]
trainData = project(trainData, ev)
testData = project(testData, ev)
```

## SVM classifier with Kernel PCA and LDA

**Datasets**:
1. Arcene
2. Madelon

**Pre processing:**
Dimentionality reduction is done by following techniques:
1. RBF kernel
2. Linear kernel
3. Kernel LDA

**Classifier:** SVM classifier from the standard scikit-learn python library is used with soft margin $C = 1.0$ and kernel being linear.

**Results:**
Kernel PCA

| Dataset | Kernel | K | Mean Accuracy | Time(s) |
|---|---|---|---|---|
| Arcene | RBF | 10 | 56.0 | 1.239 |
| | RBF | 100 | 56.0 | 1.260 |
| | linear | 10 | 56.0 | 1.059 |
| | linear | 100 | 56.0 | 1.260 |
| Madelon | RBF | 10 | 50.0 | 12.158 |
| | RBF | 100 | 50.0 | 12.757 |
| | linear | 10 | 50.0 | 10.812 |
| | linear | 100 | 50.0 | 11.640 |

LDA

| Dataset | Mean Accuracy | Time(m) |
|---|---|---|
| Arcene | 56.0 | 16.27 |
| Madelon | 50.0 | 28.39 |

**Observations:**
1. Linear kernel PCA computes exactly the same result as standard PCA method but it is much faster as it does eigen decomposition and doesn't explicitly computes the covariance matrix.
2. SVM classifier with PCA gives poor performance than Bayesian classifier used in previous assignment.
3. Variation of $\gamma$ in RBF kernel doesn't affect classification performance of SVM.
4. LDA calculates the inverse of within class scatter matrix $SW$. For this dataset,

withing class scatter matrix $SW$ is singular. Hence pseudo inverse for scatter matrix is calculated, which results in high execution time than PCA.

### SVM with RBF kernel PCA

```python
import numpy as np
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn import svm, preprocessing
from sklearn.metrics import classification_report as cr
from sklearn.decomposition import KernelPCA

def train(X, y):
    clf = svm.SVC(kernel='linear', C = 1.0, max_iter = -1)
    clf.fit(X, y)
    return clf

def predict(model, vector):
    return model.predict(vector)

def classify(model, featureVectors):
    true = 0
    total = 0
    z = []
    for feature in featureVectors:
        if feature[-1] == predict(model, feature[:-1]):
            true += 1
        z = z + predict(model, feature[:-1]).astype(np.int).tolist()
        total += 1
    data = featureVectors[:,-1].flatten()
    data = data.astype(np.int).tolist()
    print cr(data, z)
    print "Accuracy : ",
    print (true * 100) / total

def mergeData(trainData, testData):
    x = np.zeros((trainData.shape[0] + testData.shape[0], trainData.shape[1]))
    x[:trainData.shape[0], :] = trainData
    x[trainData.shape[0]:, :] = testData
    return x

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
```

```
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
    return b

def kPCA(X, gamma, k):
    distances = pdist(X, 'sqeuclidean')
    symmetricDistances = squareform(distances)
    K = exp(-gamma * symmetricDistances)
    N = K.shape[0]
    one_N = np.ones((N,N))/N
    normalizedK = K - one_N.dot(K) - K.dot(one_N)   + one_N.dot(K).dot(one_N)
    eigenValues, eigenVectors = eigh(normalizedK)
    alphas = np.column_stack((eigenVectors[:,-i] for i in range(1,k+1)))
    lambdas = [eigenValues[-i] for i in range(1,k+1)]
    return alphas, lambdas

def project(testData, X, k, gamma, alphas, lambdas):
    Data = np.zeros((testData.shape[0], k))
    for i in xrange(testData.shape[0]):
        Data[i, :] = project_x(testData[i], X, gamma, alphas, lambdas)
    return Data

def project_x(x_new, X, gamma, alphas, lambdas):
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
    return k.dot(alphas / lambdas)

file = open('arcene_train.data.txt')
X = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)

K = 100
gamma = 15
alphas, lambdas = kPCA(X, gamma, K)
testData = project(testData, X, K, gamma, alphas, lambdas)
testData = addLabels(testData, testLabels)
model = train(alphas, trainLabels)
classify(model, testData)
```

### SVM with linear kernel PCA

```
import numpy as np
from scipy.spatial.distance import pdist, squareform
from scipy import exp
```

```python
from scipy.linalg import eigh
from sklearn import svm, preprocessing
from sklearn.metrics import classification_report as cr

def train(X, y):
    clf = svm.SVC(kernel='poly', C = 1.0, max_iter = -1)
    clf.fit(X, y)
    return clf

def predict(model, vector):
    return model.predict(vector)

def classify(model, featureVectors):
    true = 0
    total = 0
    z = []
    for feature in featureVectors:
        if feature[-1] == predict(model, feature[:-1]):
            true += 1
        z = z + predict(model, feature[:-1]).astype(np.int).tolist()
        total += 1
    data = featureVectors[:,-1].flatten()
    data = data.astype(np.int).tolist()
    print z
    print cr(data, z)
    print "Accuracy : ",
    print (true * 100) / total

def mergeData(trainData, testData):
    x = np.zeros((trainData.shape[0] + testData.shape[0], trainData.shape[1]))
    x[:trainData.shape[0], :] = trainData
    x[trainData.shape[0]:, :] = testData
    return x

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
```

```
    return b

def project(testData, X, k, gamma, alphas, lambdas):
    Data = np.zeros((testData.shape[0], k))
    for i in xrange(testData.shape[0]):
        Data[i, :] = project_x(testData[i], X, gamma, alphas, lambdas)
    return Data

def project_x(x_new, X, gamma, alphas, lambdas):
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
    return k.dot(alphas / lambdas)

def kPCA(X, gamma, k):
    K = X.dot(X.T)
    N = K.shape[0]
    one_N = np.ones((N,N))/N
    normalizedK = K - one_N.dot(K) - K.dot(one_N)   + one_N.dot(K).dot(
    one_N)
    eigenValues, eigenVectors = eigh(normalizedK)
    alphas = np.column_stack((eigenVectors[:,-i] for i in range(1,k+1))
    )
    lambdas = [eigenValues[-i] for i in range(1,k+1)]
    return alphas, lambdas

file = open('arcene_train.data.txt')
X = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)

K = 100
gamma = 15
alphas, lambdas = kPCA(X, gamma, K)
testData = project(testData, X, K, gamma, alphas, lambdas)
testData = addLabels(testData, testLabels)
model = train(alphas, trainLabels)
classify(model, testData)
```

### SVM with LDA

```
import numpy as np
from numpy import linalg as LA
import math
from sklearn import preprocessing
from sklearn import svm, preprocessing
from sklearn.metrics import classification_report as cr

def mergeData(trainData, testData):
    x = np.zeros((trainData.shape[0] + testData.shape[0], trainData.
    shape[1]))
    x[:trainData.shape[0], :] = trainData
    x[trainData.shape[0]:, :] = testData
    return x

```

```
def ldaTransform(data):
    C0 = data[data[:, -1] == -1]
    C1 = data[data[:, -1] == 1]
    C0 = C0[:, :-1]
    C1 = C1[:, :-1]
    S0 = np.cov(np.transpose(C0))
    S1 = np.cov(np.transpose(C1))
    SW = S0 + S1
    Mu0 = np.mean(C0, axis = 0)
    Mu1 = np.mean(C1, axis = 0)
    Mu = np.mean(data, axis = 0)
    Mu = Mu[:-1]
    Mu = np.matrix(Mu)
    Mu0 = np.matrix(Mu0)
    Mu1 = np.matrix(Mu1)
    SB = C0.shape[0] * np.transpose(Mu0 - Mu) * (Mu0 - Mu) + C1.shape[0] * np.transpose(Mu1 - Mu) * (Mu1 - Mu)
    Swin = LA.pinv(SW) #costly
    Swin = np.matrix(Swin)
    SwinSB = Swin * SB #costly
    e, v = LA.eig(SwinSB) #costly
    s = np.argsort(e)[::-1]
    v = np.array(v)
    ev = np.zeros(v.shape)
    for i in xrange(e.shape[0]):
        ev[:, i] = v[:, s[i]]
    w = ev[:, 0]
    w = np.matrix(w)
    return w

def project(data, w):
    data = np.matrix(data)
    data = np.transpose(data)
    newData = w * data
    newData = np.transpose(newData)
    newData = np.array(newData)
    return newData

def addLabels(data, trainLabels):
    b = np.zeros((data.shape[0], data.shape[1] + 1))
    b[:, :-1] = data
    b[:, -1] = trainLabels
    return b

def getDataMatrix(file, intOrFloat):
    #intOrFloat decides whether data should be int or float
    if (intOrFloat == 1):
        featureVectors = []
        for line in file :
            vector = line.strip().lower().split(' ')
            featureVectors.append(vector)
        data = np.array(featureVectors)
        data = data.astype(float)
    else:
        trainLabels = []
        for line in file :
            vector = line
```

```
            trainLabels.append(vector)
        data = np.array(trainLabels)
        data = data.astype(int)
    return data


def train(X, y):
    clf = svm.SVC(kernel='linear', C = 1.0, max_iter = -1)
    clf.fit(X, y)
    return clf

def predict(model, vector):
    return model.predict(vector)

def classify(model, featureVectors):
    true = 0
    total = 0
    z = []
    for feature in featureVectors:
        if feature[-1] == predict(model, feature[:-1]):
            true += 1
        z = z + predict(model, feature[:-1]).astype(np.int).tolist()
        total += 1
    data = featureVectors[:,-1].flatten()
    data = data.astype(np.int).tolist()
    print z
    print cr(data, z)
    print "Accuracy : ",
    print (true * 100) / total

file = open('arcene_train.data.txt')
data = getDataMatrix(file, 1)
file = open('arcene_train.labels.txt')
trainLabels = getDataMatrix(file, 0)
file = open('arcene_valid.data.txt')
testData = getDataMatrix(file, 1)
file = open('arcene_valid.labels.txt')
testLabels = getDataMatrix(file, 0)

trainData = addLabels(data, trainLabels)
ev = ldaTransform(trainData)
trainData = trainData[:, :-1]
trainData = project(trainData, ev)
testData = project(testData, ev)
model = train(trainData, trainLabels)
testData = addLabels(testData, testLabels)
classify(model, testData)
```

## References


1. Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, Gideon Dror, 2004. Result analysis of the NIPS 2003 feature selection challenge.

2. Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, Gideon Dror, 2004. Result analysis of the NIPS 2003 feature selection challenge.