**Support Vector Machines**

* SVM is a very powerful & versatile model, which can perform
  + Linear Classification
  + Non-linear Classification
  + Regression
  + Outlier Detection
* It is well suited for complex, small or medium sized datasets

# Linear SVM Classification

* Linear classification can separate two classes by a straight line.
* Good Model – Decision Boundary as far from training instances.
* Support vectors are located at the edge of the street.

**#Large Margin Classification Using SVC**

from sklearn.svm import SVC

from sklearn import datasets

iris = datasets.load\_data()

X= iris[‘data’][:2,3] **#petal length, petal width**

y = iris[‘target’]

setosa\_or\_versicolor = (y == 0) | (y == 1)

X = X[setosa\_or\_versicolor]

y = y[setosa\_or\_versicolor]

svm\_clf = SVC(kernel=’linear’,c=float(‘inf’))

svm\_clf.fit(X,y)

* It is important to do scaling for SVM Model.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* **Hard Margin Classification :** Strictly impose that all the instances should be off the street & on a particular side of the decision boundary. It only works if data is linearly separable. It quite sensitive to outliers.

***# "hard" margin classification:***

***# - all instances need to be "out of the street".***

***# - all instances need to be "on the right side of the street".***

***# problem: doable only if data is linearly separable***

***# problem: very sensitive to outliers***

X\_outliers = np.array([[3.4, 1.3], [3.2, 0.8]])

y\_outliers = np.array([0, 0])

Xo1 = np.concatenate([X, X\_outliers[:1]], axis=0)

yo1 = np.concatenate([y, y\_outliers[:1]], axis=0)

Xo2 = np.concatenate([X, X\_outliers[1:]], axis=0)

yo2 = np.concatenate([y, y\_outliers[1:]], axis=0)

svm\_clf2 = SVC(kernel="linear", C=10\*\*9)*#float("inf"))*

svm\_clf2.fit(Xo2, yo2)

plt.figure(figsize=(12,2.7))

plt.subplot(121)

plt.plot(Xo1[:, 0][yo1==1], Xo1[:, 1][yo1==1], "bs")

plt.plot(Xo1[:, 0][yo1==0], Xo1[:, 1][yo1==0], "yo")

plt.text(0.3, 1.0, "Impossible!", fontsize=20, color="red")

plt.xlabel("Petal length", fontsize=14)

plt.ylabel("Petal width", fontsize=14)

plt.annotate("Outlier",

xy=(X\_outliers[0][0], X\_outliers[0][1]),

xytext=(2.5, 1.7),

ha="center",

arrowprops=dict(facecolor='black', shrink=0.1),

fontsize=16,

)

plt.axis([0, 5.5, 0, 2])

plt.subplot(122)

plt.plot(Xo2[:, 0][yo2==1], Xo2[:, 1][yo2==1], "bs")

plt.plot(Xo2[:, 0][yo2==0], Xo2[:, 1][yo2==0], "yo")

plot\_svc\_decision\_boundary(svm\_clf2, 0, 5.5)

plt.xlabel("Petal length", fontsize=14)

plt.annotate("Outlier",

xy=(X\_outliers[1][0], X\_outliers[1][1]),

xytext=(3.2, 0.08),

ha="center",

arrowprops=dict(facecolor='black', shrink=0.1),

fontsize=16,

)

plt.axis([0, 5.5, 0, 2])

* **Soft Margin Classification :** is a balance between keeping the street as large as possible and limiting the margin violations. It is regulated by **‘C’ Parameter**

from sklearn import datasets

from sklearn.svm import LinearSVC

from sklearn.preprocessing import StandardScaler

from sklearn.numpy as np

from sklearn.matplotlib.pyplot as plt

from sklearn.pipeline import pipeline

iris = datasets.load\_iris()

X = iris[‘data’][:2,3]

Y = (iris[‘target’]==2).astype(np.float64)

scaler = StandardScaler()

svm\_clf1 = LinearSVC(C=1,loss=’hinge’)

scaled\_svm\_clf1 = Pipeline ((

(‘scaler’,scaler),

(‘linear\_svc’,svm\_clf1)

scaled\_svm\_clf1.fit(X,y)

scaled\_svm\_clf1.predict([[5.5,1.7]])

**# Convert to unscaled parameters**

b2 = svm\_clf1.decision\_function([-scaler.mean\_ / scaler.scale\_])

w2 = svm\_clf1.coef\_[0] / scaler.scale\_

svm\_clf1.intercept\_ = np.array([b2])

svm\_clf1.coef\_ = np.array([w2])

**#Finding Support Vectors**

t = y \* 2 - 1

support\_vectors\_idx2 = (t \* (X.dot(w2) + b2) < 1).ravel()

svm\_clf1.support\_vectors\_ = X[support\_vectors\_idx2]

***#Solution to ‘hard margins’ problem: Control hardness with C hyperparameter***

**from** **sklearn** **import** datasets

**from** **sklearn.pipeline** **import** Pipeline

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.svm** **import** LinearSVC

iris = datasets.load\_iris()

X = iris["data"][:, (2, 3)] *# petal length, petal width*

y = (iris["target"] == 2).astype(np.float64) *# Iris-Virginica*

scaler = StandardScaler()

svm\_clf1 = LinearSVC(C=100, loss="hinge")

svm\_clf2 = LinearSVC(C=1, loss="hinge")

scaled\_svm\_clf1 = Pipeline((

("scaler", scaler),

("linear\_svc", svm\_clf1),

))

scaled\_svm\_clf2 = Pipeline((

("scaler", scaler),

("linear\_svc", svm\_clf2),

))

scaled\_svm\_clf1.fit(X, y)

scaled\_svm\_clf2.fit(X, y)

scaled\_svm\_clf2.predict([[5.5, 1.7]])

***# Convert to unscaled parameters***

b1 = svm\_clf1.decision\_function([-scaler.mean\_ / scaler.scale\_])

b2 = svm\_clf2.decision\_function([-scaler.mean\_ / scaler.scale\_])

w1 = svm\_clf1.coef\_[0] / scaler.scale\_

w2 = svm\_clf2.coef\_[0] / scaler.scale\_

svm\_clf1.intercept\_ = np.array([b1])

svm\_clf2.intercept\_ = np.array([b2])

svm\_clf1.coef\_ = np.array([w1])

svm\_clf2.coef\_ = np.array([w2])

***# Find support vectors (LinearSVC does not do this automatically)***

t = y \* 2 - 1

support\_vectors\_idx1 = (t \* (X.dot(w1) + b1) < 1).ravel()

support\_vectors\_idx2 = (t \* (X.dot(w2) + b2) < 1).ravel()

svm\_clf1.support\_vectors\_ = X[support\_vectors\_idx1]

svm\_clf2.support\_vectors\_ = X[support\_vectors\_idx2]

# Non-Linear SVM Classification

* Polynomial features + Standard Scaler + LinearSVC
* If Datasets cannot be linearly separable, then :
* **Add more Polynomial Features to it**

from sklearn.datasets import make\_moons

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

X,y = make\_moons(n\_samples=100,noise=0.15,random\_state=42)

polynomial\_svm\_clf = Pipeline((

(‘poly\_feature’,PolynomialFeatures(degree=3)),

(‘scaler’,StandardScaler()),

(‘svm\_clf’,LinearSVC(C=10,loss=’hinge’))

))

Polynomial\_svm\_clf.fit(X,y)

**def** plot\_predictions(clf, axes):

x0s = np.linspace(axes[0], axes[1], 100)

x1s = np.linspace(axes[2], axes[3], 100)

x0, x1 = np.meshgrid(x0s, x1s)

X = np.c\_[x0.ravel(), x1.ravel()]

y\_pred = clf.predict(X).reshape(x0.shape)

y\_decision = clf.decision\_function(X).reshape(x0.shape)

plt.contourf(x0, x1, y\_pred, cmap=plt.cm.brg, alpha=0.2)

plt.contourf(x0, x1, y\_decision, cmap=plt.cm.brg, alpha=0.1)

plot\_predictions(polynomial\_svm\_clf, [-1.5, 2.5, -1, 1.5])

plot\_dataset(X, y, [-1.5, 2.5, -1, 1.5])

*#save\_fig("moons\_polynomial\_svc\_plot")*

plt.show()

* **Solving polynomial-feature problems via the kernel trick**
* SVC Polynomial kernel + Standard Scaler
* Adding Polynomial features works well. But when dataset is complex, Low Polynomial Degree won’t help & high Degree will be very slow.
* **Solution :** Polynomials Kernels OR kernel Trick

from sklearn.svm import SVC

from sklearn.datasets import make\_moons

X\_all,y\_all = make\_moons(n\_samples=150,noise=0.15,random\_state=42)

X, X\_test, y, y\_test = X\_all[:100], X\_all[100:], y\_all[:100], y\_all[100:]

***# train SVM classifier using 3rd-degree polynomial kernel***

poly\_kernel\_svm\_clf = Pipeline ((

(‘scaler’,StandardScaler()),

(‘svm\_clf’,SVC(kernel=’poly’,degree=3,coef0=1,C=5))

))

***# train SVM classifier using 10th-degree polynomial kernel(for comparison)***

Poly100\_kernel\_svm\_clf = Pipeline ((

(‘scaler’,StandardScaler()),

(‘svm\_clf’,SVC(kernel=’poly’,degree=10,coef0=100,C=5))

))

poly\_kernel\_svm\_clf.fit(X,y)

poly100\_kernel\_svm\_clf.fit(X,y)

plt.figure(figsize=(11, 4))

plt.subplot(121)

plot\_predictions(poly\_kernel\_svm\_clf, [-1.5, 2.5, -1, 1.5])

plot\_dataset(X, y, [-1.5, 2.5, -1, 1.5])

plt.title(r"$d=3, r=1, C=5$", fontsize=18)

plt.subplot(122)

plot\_predictions(poly100\_kernel\_svm\_clf, [-1.5, 2.5, -1, 1.5])

plot\_dataset(X, y, [-1.5, 2.5, -1, 1.5])

plt.title(r"$d=10, r=100, C=5$", fontsize=18)

*#save\_fig("moons\_kernelized\_polynomial\_svc\_plot")*

plt.show()

from sklearn.metrics import confusion\_matrix

y\_predict1 = poly\_kernel\_svm\_clf.predict(X\_test)

y\_predict2 = poly100\_kernel\_svm\_clf.predict(X\_test)

confusion\_matrix(y\_test,y\_predict1)

confusion\_matrix(y\_test,y\_predict2)

* **Solving polynomial-feature problems by adding Similarity features**
* Create a landmark at each and every instance of the dataset. If training set is huge, no. of new features added will be huge.
* **Gaussian RBF Kernel** can solve the problem without adding many features.

*# define similarity function to be Gaussian Radial Basis Function (RBF)*

*# equals 0 (far away) to 1 (at landmark)*

**def** gaussian\_rbf(x, landmark, gamma):

**return** np.exp(-gamma \* np.linalg.norm(x - landmark, axis=1)\*\*2)

from sklearn.svm import SVC

from sklearn.datasets import make\_moons

X,y = make\_moons(n\_samples=100,noise=0.15,random\_state=42)

rbf\_kernel\_svm\_clf = Pipeline((

(‘scaler’,StandardScaler()),

(‘svm\_clf’,SVC(kernel=’rbf’,gamma=5,C=0.001))

))

rbf\_kernel\_svm\_clf.fit(X,y)

gamma1,gamma2 = 0.1,5

C1,C2 = 0.001,1000

hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

svm\_clfs = []

for gamma,C in hyperparams :

rbf\_kernel\_svm\_clf = Pipeline((

(‘scaler’,StandardScaler()),

(‘svm\_clf’,SVC(kernel=’rbf’,gamma=gamma,C=C))

))

rbf\_kernel\_svm\_clf.fit(X, y)

svm\_clfs.append(rbf\_kernel\_svm\_clf)

plt.figure(figsize=(11, 7))

**for** i, svm\_clf **in** enumerate(svm\_clfs):

plt.subplot(221 + i)

plot\_predictions(svm\_clf, [-1.5, 2.5, -1, 1.5])

plot\_dataset(X, y, [-1.5, 2.5, -1, 1.5])

gamma, C = hyperparams[i]

plt.title(r"$\gamma = **{}**, C = **{}**$".format(gamma, C), fontsize=16)

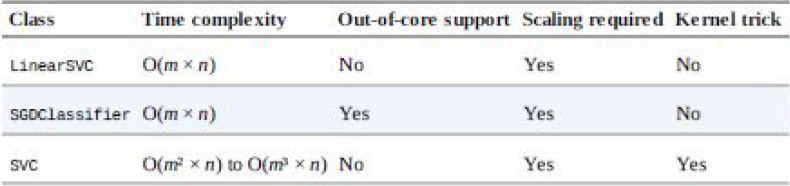
***#save\_fig("moons\_rbf\_svc\_plot")***

plt.show()

***# bigger gamma = narrower bell curve, so each instance's area of influence = smaller.***

***# smaller gamma: bigger bell curve = smoother decision boundary.***

* **With increasing Gamma value,** bell curve becomes narrow, reduces influence of each instance. Decision Boundary becomes irregular.
* **Which kernel to use when ?** 
  + LinearSVC faster than SVC(kernel=’linear’) for large datasets with a lot of features.
  + Gaussian RBF Kernel
  + Cross Validation & grid Search.



# SVM Regression

* **Linear SVM : LinearSVR + Epsilon**
  + Fits as many instances as possible on the street while limiting margin violations.
  + Width of the SVM Regression Model is controlled by a hyperparameter epsilon. Width is not affected by adding Instances.

***### Linear SVM Regression: Comparison of eps = 1.5 and exp-0.5***

**from** **sklearn.svm** **import** LinearSVR

**import** **numpy.random** **as** **rnd**

rnd.seed(42)

m = 50

X = 2 \* rnd.rand(m, 1)

y = (4 + 3 \* X + rnd.randn(m, 1)).ravel()

svm\_reg1 = LinearSVR(epsilon=1.5)

svm\_reg2 = LinearSVR(epsilon=0.5)

svm\_reg1.fit(X, y)

svm\_reg2.fit(X, y)

**def** find\_support\_vectors(svm\_reg, X, y):

y\_pred = svm\_reg.predict(X)

off\_margin = (np.abs(y - y\_pred) >= svm\_reg.epsilon)

**return** np.argwhere(off\_margin)

svm\_reg1.support\_ = find\_support\_vectors(svm\_reg1, X, y)

svm\_reg2.support\_ = find\_support\_vectors(svm\_reg2, X, y)

eps\_x1 = 1

eps\_y\_pred = svm\_reg1.predict([[eps\_x1]])

plt.scatter(X,y)

plt.xlabel(r"$x\_1$", fontsize=18)

plt.ylabel(r"$y$", fontsize=18)

**def** plot\_svm\_regression(svm\_reg, X, y, axes):

x1s = np.linspace(axes[0], axes[1], 100).reshape(100, 1)

y\_pred = svm\_reg.predict(x1s)

plt.plot(x1s, y\_pred, "k-", linewidth=2, label=r"$\hat**{y}**$")

plt.plot(x1s, y\_pred + svm\_reg.epsilon, "k--")

plt.plot(x1s, y\_pred - svm\_reg.epsilon, "k--")

plt.scatter(X[svm\_reg.support\_], y[svm\_reg.support\_], s=180, facecolors='#FFAAAA')

plt.plot(X, y, "bo")

plt.xlabel(r"$x\_1$", fontsize=18)

plt.legend(loc="upper left", fontsize=18)

plt.axis(axes)

plt.figure(figsize=(9, 4))

plt.subplot(121)

plot\_svm\_regression(svm\_reg1, X, y, [0, 2, 3, 11])

plt.title(r"$\epsilon = **{}**$".format(svm\_reg1.epsilon), fontsize=18)

plt.ylabel(r"$y$", fontsize=18, rotation=0)

*#plt.plot([eps\_x1, eps\_x1], [eps\_y\_pred, eps\_y\_pred - svm\_reg1.epsilon], "k-", linewidth=2)*

plt.annotate(

'', xy=(eps\_x1, eps\_y\_pred), xycoords='data',

xytext=(eps\_x1, eps\_y\_pred - svm\_reg1.epsilon),

textcoords='data', arrowprops={'arrowstyle': '<->', 'linewidth': 1.5}

)

plt.text(0.91, 5.6, r"$\epsilon$", fontsize=20)

plt.subplot(122)

plot\_svm\_regression(svm\_reg2, X, y, [0, 2, 3, 11])

plt.title(r"$\epsilon = **{}**$".format(svm\_reg2.epsilon), fontsize=18)

*#save\_fig("svm\_regression\_plot")*

plt.show()

* **Main goal w.r.t. Epsilon Value to maximize the no of training sets within the epsilon line.**
* **Nonlinear SVM : SVR Polynomial Kernel + degree + C + Epsilon**
* A ‘kernelized SVM’ Regression model can be used to
  + **C –** Penalty for being outside the margin or error in classification
  + Higher C 🡪 Lessor Violations but lesser regularization

*### Example: Comparison of non-linear SVM Regression with different hyper parameters*

**from** **sklearn.svm** **import** SVR

*# random quadratic training set.*

rnd.seed(42)

m = 100

X = 2 \* rnd.rand(m, 1) - 1

y = (0.2 + 0.1 \* X + 0.5 \* X\*\*2 + rnd.randn(m, 1)/10).ravel()

svm\_poly\_reg1 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)

svm\_poly\_reg2 = SVR(kernel="poly", degree=2, C=0.01, epsilon=0.1)

svm\_poly\_reg3 = SVR(kernel="poly", degree=2, C=100, epsilon=0.25)

svm\_poly\_reg4 = SVR(kernel="poly", degree=2, C=0.01, epsilon=0.25)

svm\_poly\_reg1.fit(X, y)

svm\_poly\_reg2.fit(X, y)

svm\_poly\_reg3.fit(X, y)

svm\_poly\_reg4.fit(X, y)

plt.figure(figsize=(9, 4))

plt.subplot(121)

plot\_svm\_regression(svm\_poly\_reg1, X, y, [-1, 1, 0, 1])

plt.title(r"$degree=**{}**, C=**{}**, \epsilon = **{}**$".format(svm\_poly\_reg1.degree, svm\_poly\_reg1.C, svm\_poly\_reg1.epsilon), fontsize=18)

plt.ylabel(r"$y$", fontsize=18, rotation=0)

plt.subplot(122)

plot\_svm\_regression(svm\_poly\_reg2, X, y, [-1, 1, 0, 1])

plt.title(r"$degree=**{}**, C=**{}**, \epsilon = **{}**$".format(svm\_poly\_reg2.degree, svm\_poly\_reg2.C, svm\_poly\_reg2.epsilon), fontsize=18)

*#save\_fig("svm\_with\_polynomial\_kernel\_plot")*

plt.show()