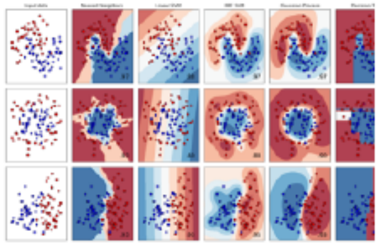
Scikit-learn

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable

Classification

Identifying which category an object belongs to.

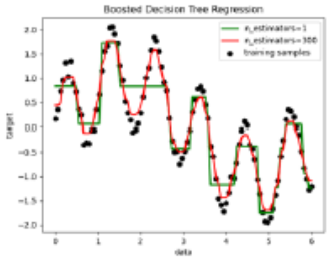
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

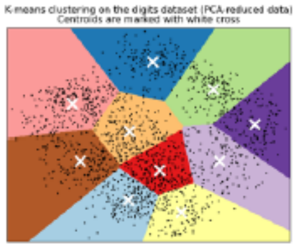
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Clustering

Automatic grouping of similar objects into sets.

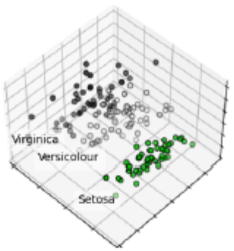
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



Dimensionality reduction

Reducing the number of random variables to consider.

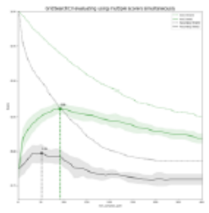
Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...



Model selection

Comparing, validating and choosing parameters and models.

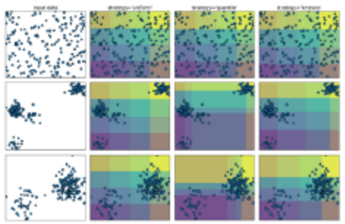
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...



Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Data in scikit-learn stored as a 2D array [m_samples, n_features]

Scikit-learn’s interface

Linear Models

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train)
model.predict(X_new)
```

Ordinary least squares
Linear Regression

Stochatic gradient descent

$$J(\theta) = MSE(\theta) + \frac{\lambda}{2}\sum_{i=1}^{n}\theta_i^2 \quad \text{(Lc 1)}$$

```
from sklearn.linear_model import SGDRegressor
model =  SGDRegressor(loss='squared_loss', penalty='l2’,
                      alpha=0.0001, learning_rate=‘constant’, eta0=0.01)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs', max_iter=100, multi_class='auto’)
```

solver{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’
Algorithm to use in the optimization problem

Scikit-learn

kNN

sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, weights='uniform', p=2, metric='minkowski'*)

sklearn.neighbors.**KNeighborsRegressor**()

Decision Tree

sklearn.tree.**DecisionTreeClassifier**(*criterion='gini', max_depth=None, min_samples_split=2, class_weight=None*)

sklearn.tree.**DecisionTreeRegressor**()

Model Evaluation

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

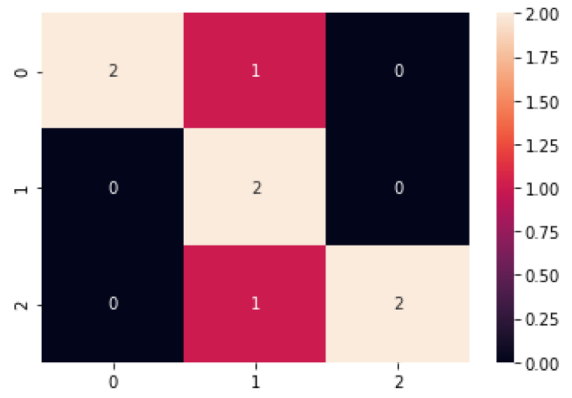
```
y_true = [0, 0, 0, 1, 1, 2, 2, 2]
y_pred = [0, 0, 1, 1, 1, 2, 1, 2]
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.50      1.00      0.67         2
           2       1.00      0.67      0.80         3

    accuracy                           0.75         8
   macro avg       0.83      0.78      0.76         8
weighted avg       0.88      0.75      0.77         8
```

```
conf = confusion_matrix(y_true, y_pred)
print(conf)
```

```
[[2 1 0]
 [0 2 0]
 [0 1 2]]
```

```
import seaborn as sns
sns.heatmap(conf, annot=True)
```

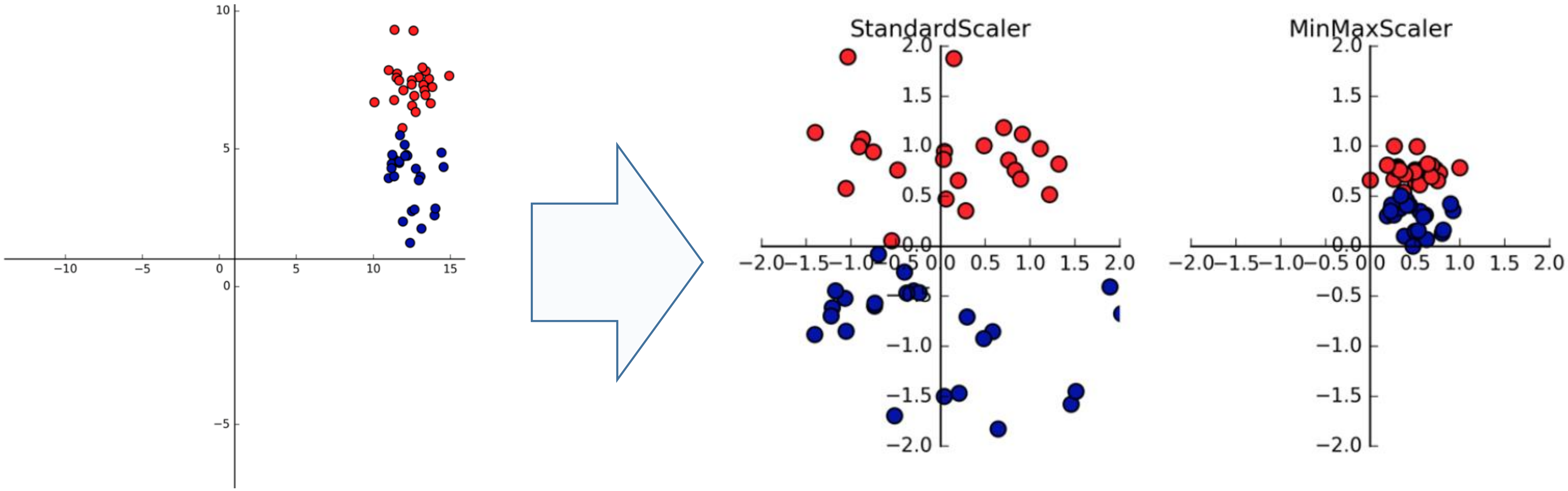


Train Test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
```

Scikit-learn

Data Normalization



sklearn.preprocessing.**StandardScaler**()

Standardize features by removing the mean and scaling to unit variance

sklearn.preprocessing.**MinMaxScaler**(*feature_range=0, 1*)

Transform features by scaling each feature to a given range

Usage:
fit(X)
transform(X)
or
fit_transform(X)

SVM in sklearn lib

sklearn.svm.**LinearSVC**()
sklearn.linear_model.**SGDClassifier**(loss='hinge')
sklearn.svm.**SVC**(*kernel='rbf'*)

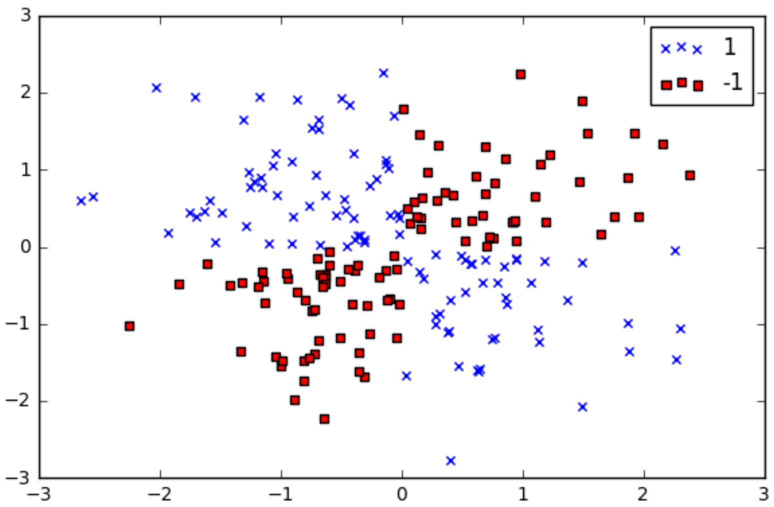
Class	Time complexity	Scaling required	Kernel trick
LinearSVC	O(mxn)	Yes	No
SGDClassifier	O(mxn)	Yes	No
SVC	O(m^2xn) to O(m^3xn)	Yes	Yes

SVMs are sensitive to the feature scaling!!!

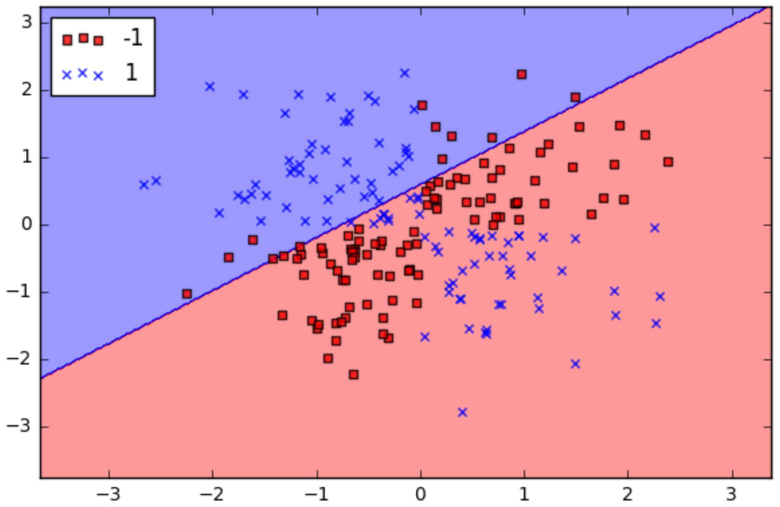
SVM software packages:

- libsvm – most commonly used implementation of kernalized svm, sklearn uses wrapper over it
- liblinear – gradient descent based implementation of linear SVM

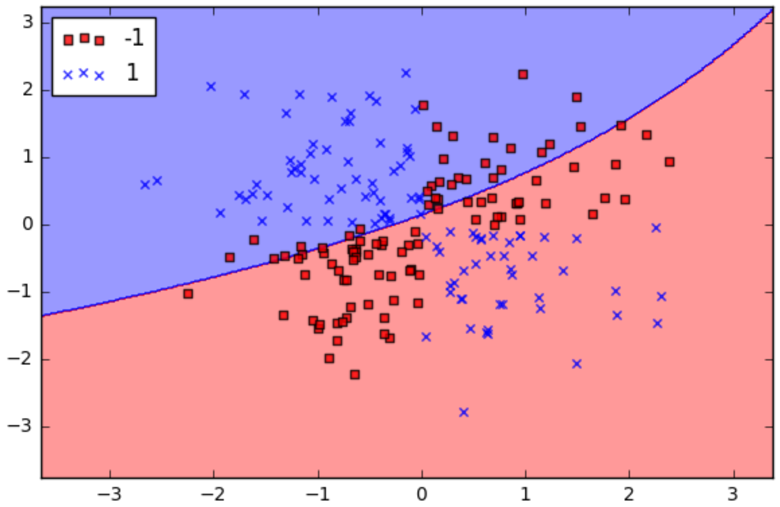
SVM Example



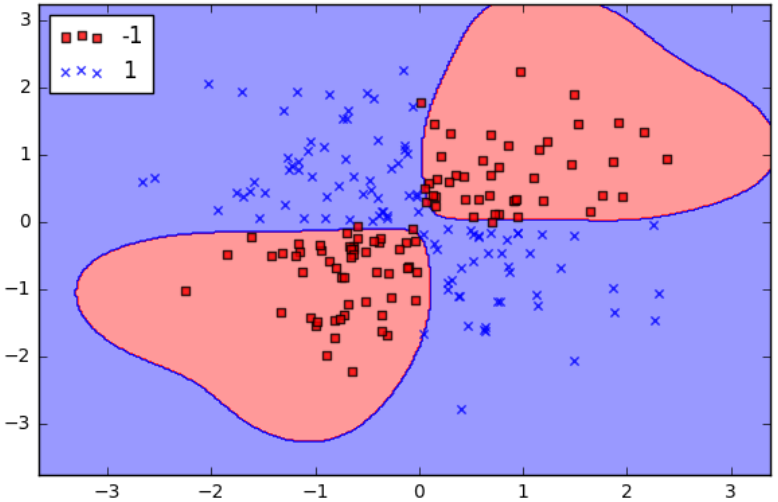
SVC(kernel='linear', C=1)



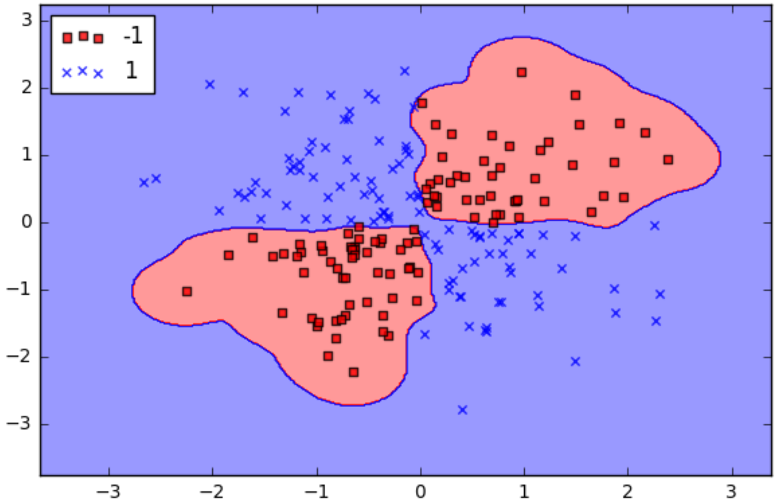
SVC(kernel='rbf', gamma=.01, C=1)



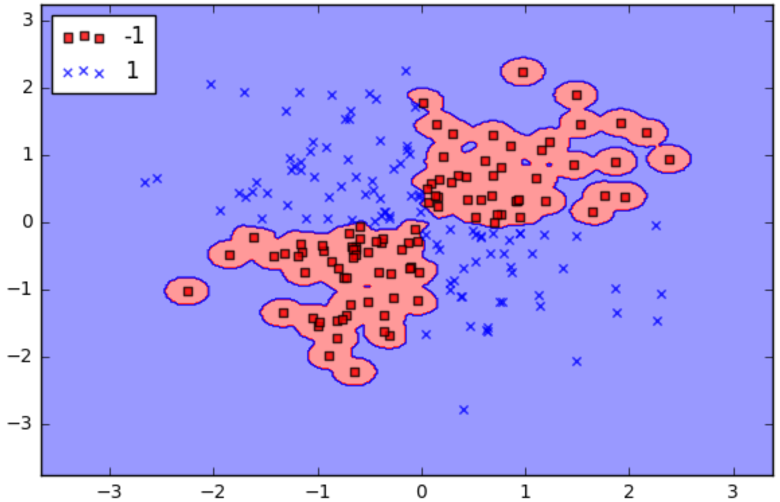
SVC(kernel='rbf', gamma=1, C=1)



SVC(kernel='rbf', gamma=10, C=1)

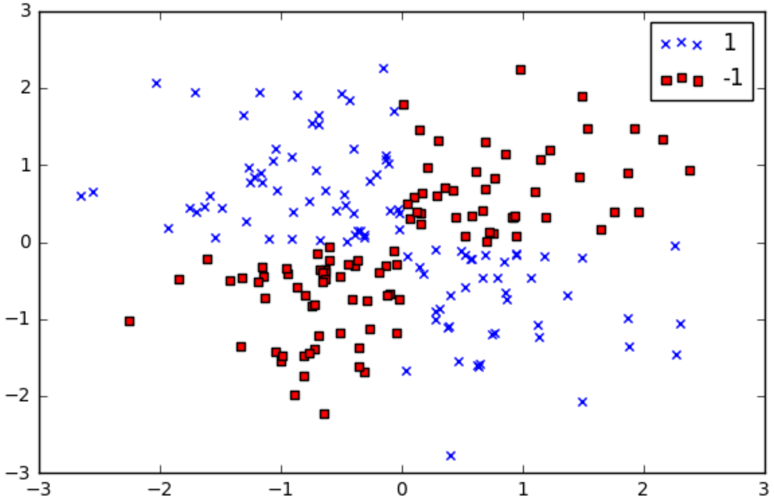


SVC(kernel='rbf', gamma=100, C=1)

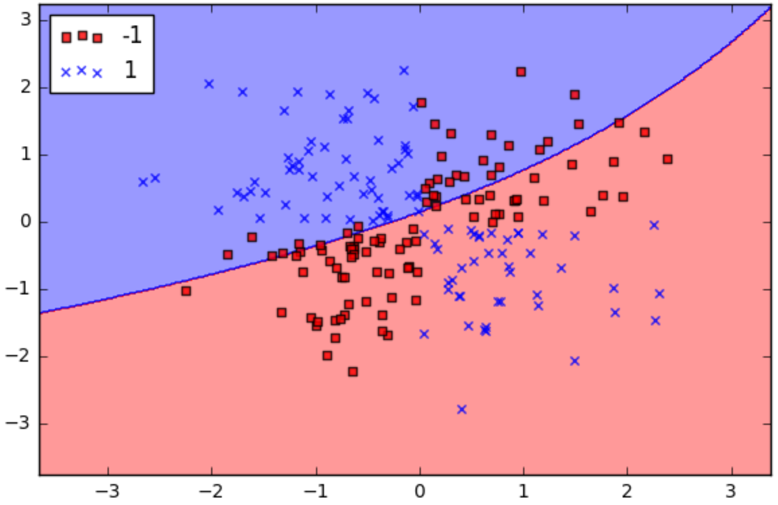


https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/

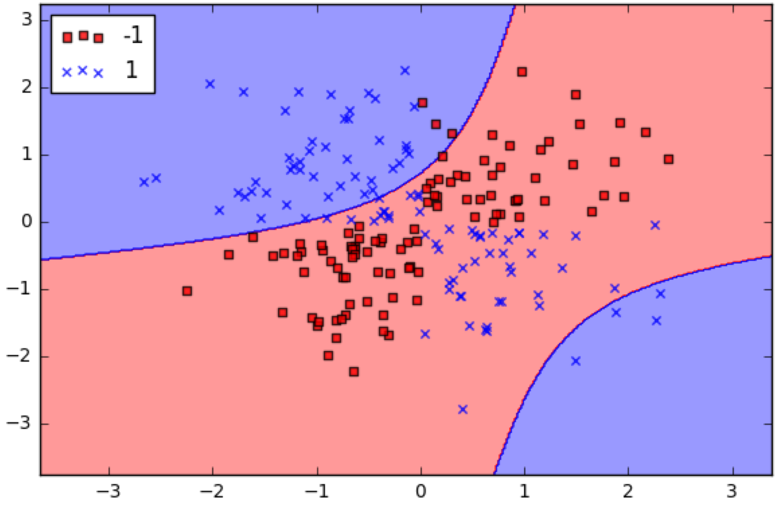
SVM Example



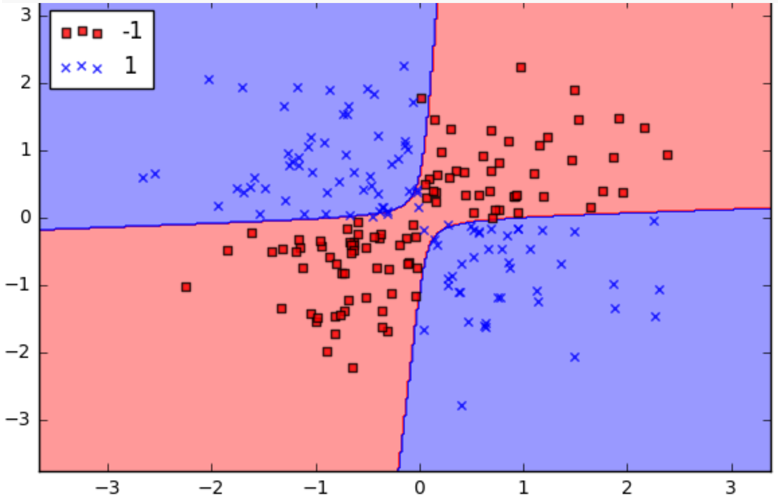
SVC(kernel='rbf', gamma=.01, C=1)



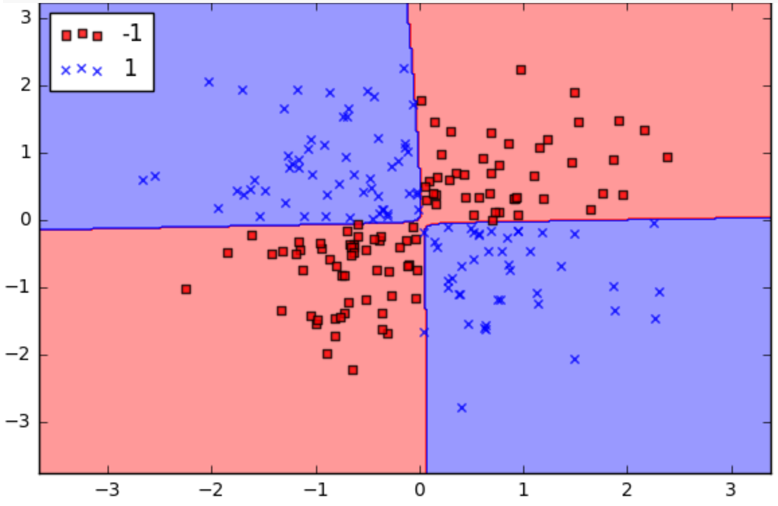
SVC(kernel='rbf', gamma=.01, C=10)



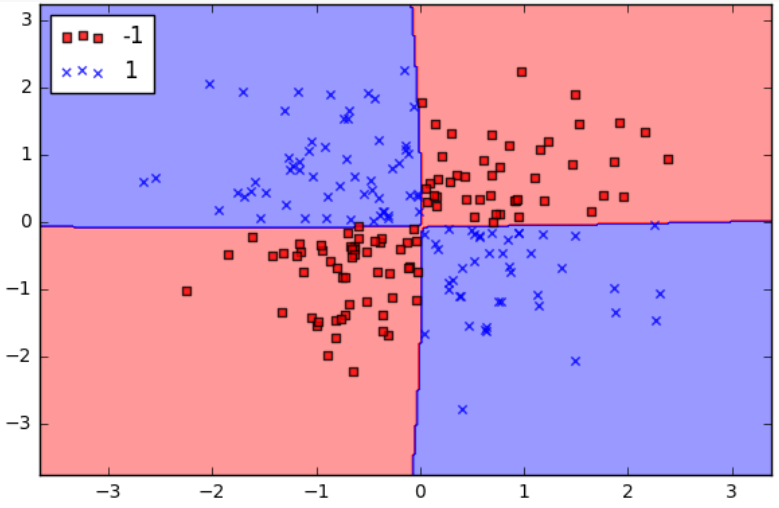
SVC(kernel='rbf', gamma=.01, C=1000)



SVC(kernel='rbf', gamma=.01, C=10000)



SVC(kernel='rbf', gamma=.01, C=100000)



SVM in sklearn lib

Regression:

```
sklearn.svm.LinearSVR()
Sklearn.svm.SVR()
```

GridSearch:

```
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
grid.fit(X,y)
print(grid.best_params_)
```

Ensemble Classifiers

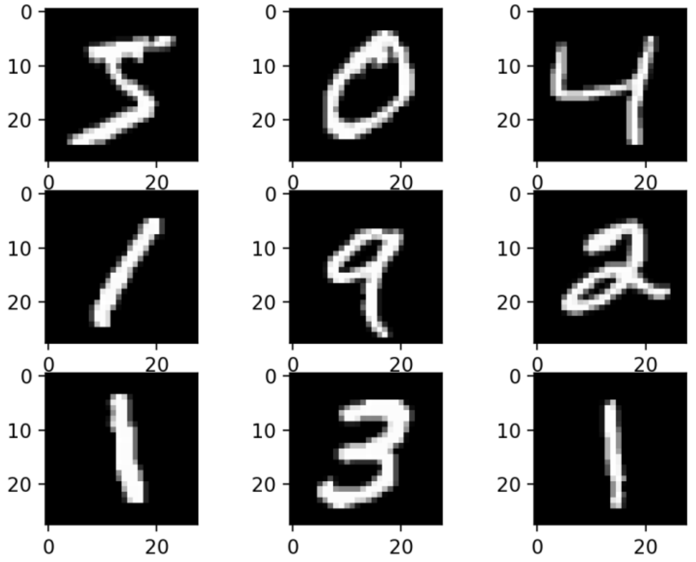
sklearn.ensembles.AdaBoostClassifier

sklearn.ensembles.RandomForestClassifier

Feature importance with Random Forest:

- A single DT: important features are likely to appear closer to the root of tree, while unimportant features often appear closer to the leaves
- RF: feature importance as average depth at which it appears across all trees in the forest

RandomForestClassifier().feature_importances_



The MNIST database is a large database of handwritten digits (28x28 pixels, 60,000 training images and 10,000 testing images)

MNIST pixel importance with accordance to RF classifier