Review

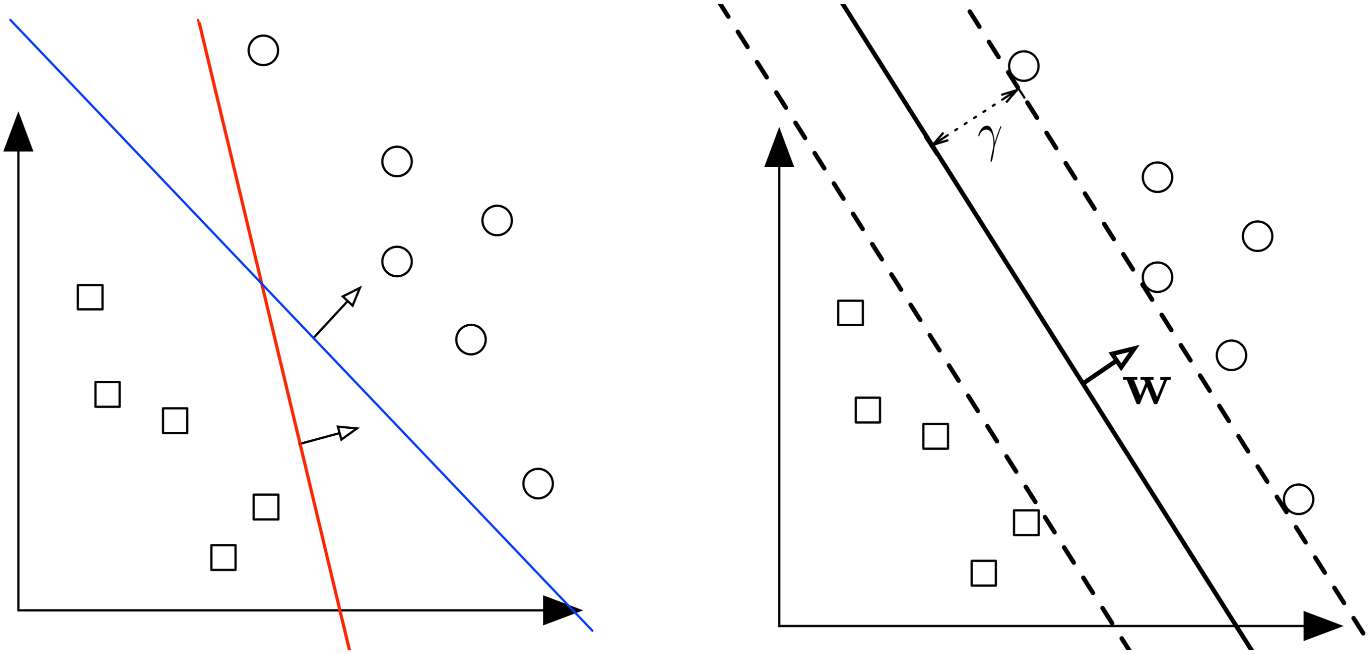
Multi-class classification

- One-vs-Rest (`sklearn.multiclass.OneVsRestClassifier`)
    - N-classes:
        - N binary classifiers
        - N-split of the training set, each includes all data points but divided into class i v.s. others)
    - Prediction: `argmax` of scores produced by a bunch of binary classifiers
- One-vs-One (`sklearn.multiclass.OneVsOneClassifier`)
    - N classes:
        - $\binom{N}{2}$ binary classifiers
        - $\binom{N}{2}$ split of the training set, each includes only data points with the label i and j
    - Prediction:
        - majority votes
        - Or the class with the most sum score is taken as the class label
- Compare:
    - the number of classifiers?
    - data used for training each classifier?
    - what if there is a specific class dominates the training set (i.e. imbalanced classes)?
    - when N is large, what will be the greatest issues for the two algorithms respectively?

SVM:

Concepts:

- difference between SVM and logistic regression?
- what is a hyperplane? what is an 'optimal' hyperplane?



(ref: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/images/svm/margin.png)

- what is a support vector?
- what is the margin M?
  - $\min_i \frac{2}{||w||_2}|w^T x_i + b|$
  - --> set the positive/negative plane to: $w^T x_i + b = \pm 1$
- The loss function to maximize the margin?
  - goal: $\max M(w, b)$ such that $\forall i, y_i(w^T x_i + b) \geq 0$
    - Or by definition of M: $\min w^T w$ such that $\forall i, y_i(w^T x_i + b) \geq 1$

Primal & Dual formulation

Kernel: a metric/weighting function on dot product:

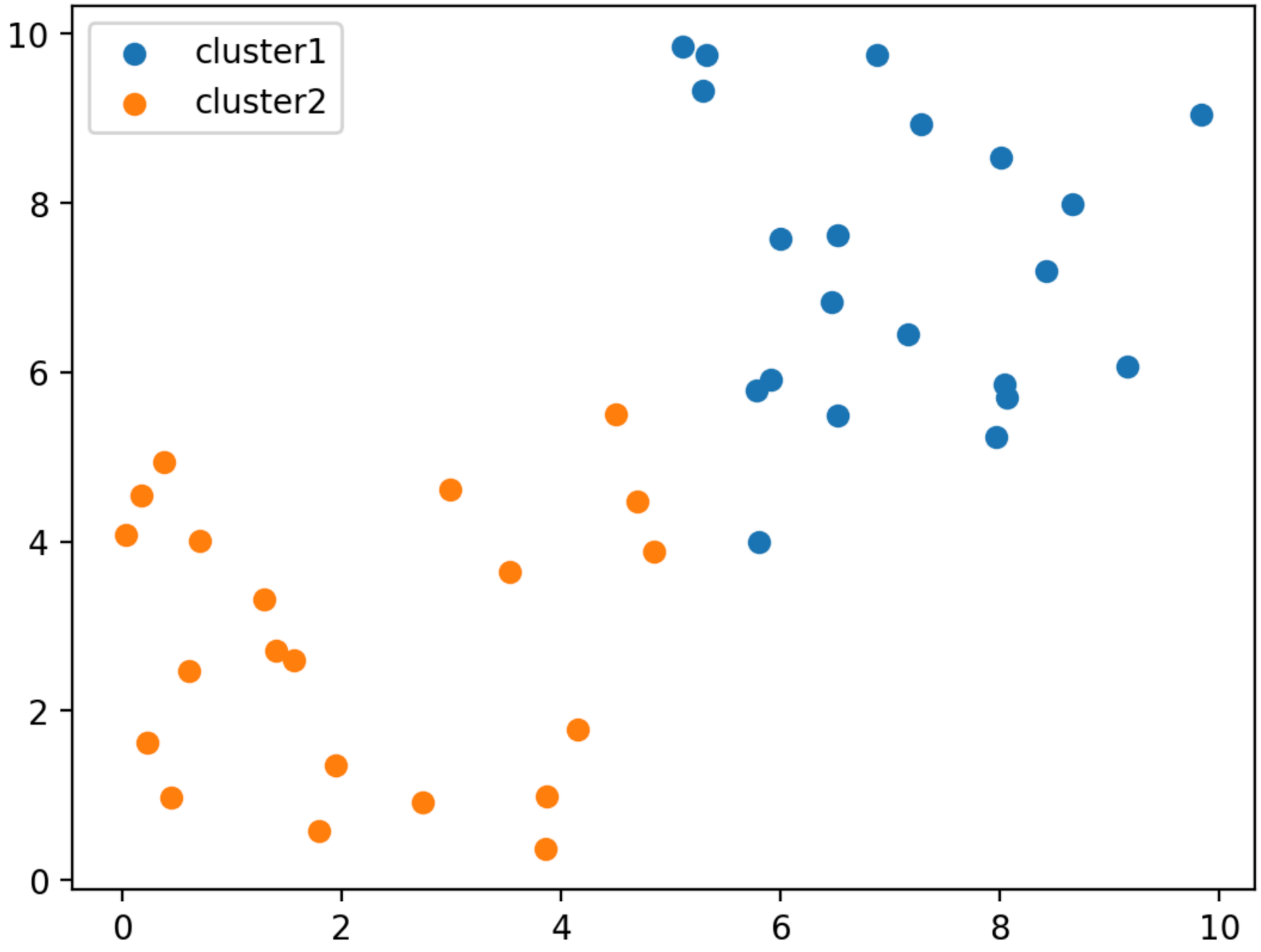
- $k(x, x') =< \phi(x), \phi(x') >$
  - linear: $K(x, x') = x^T x' + c$
  - polynomial: $K(x, x') = (ax^T x' + c)^d$
  - radial basis: $K(x, x') = exp(-\frac{||x-x'||^2}{2\sigma^2})$
  - Sigmoid: $K(x, x') = tanh(ax^T x' + c)$

Classifier:

- Primal: $f(x) = w^T x + b$
- Dual: $f(x) = \sum_i^N a_i y_i (x_i^T x) + b$
  - compare with KNN...
  - support vectors: $a_i \neq 0$

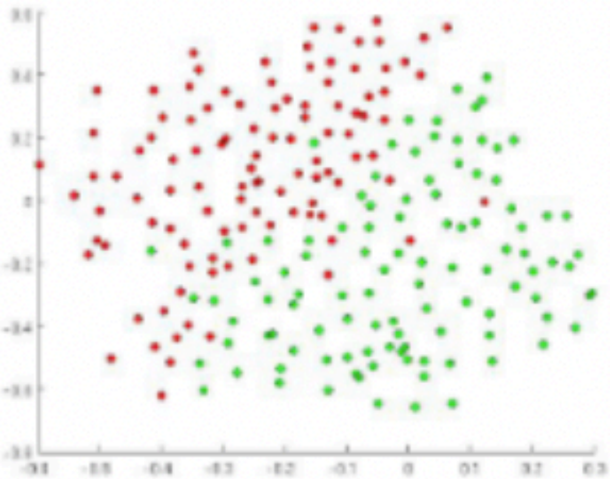
Discussion Questions

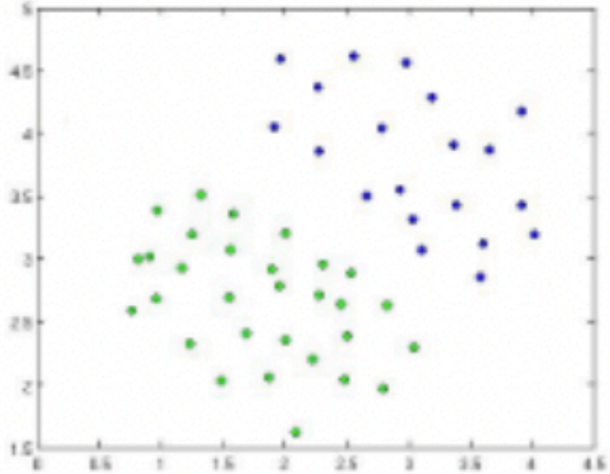
1. Consider the following dataset where the different colors indicate the different class labels a specific data point belongs to:

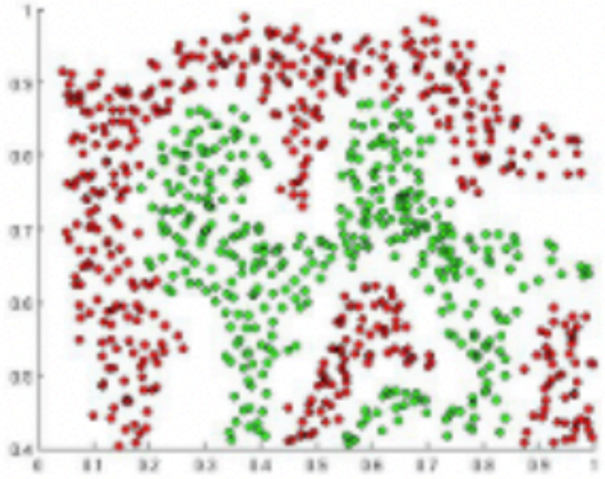


   - Is this dataset linearly separable?
   - Identify the support vectors and draw the positive margin, negative margin and decision boundary of a **hard**-SVM classifier. What is the training accuracy? Do you expect it to generalize well in the unseen dataset? What are some limitations of hard-SVM?
   - Identify the support vectors and draw the positive margin, negative margin and decision boundary of a **soft**-SVM classifier. How is that decision boundary differ from the hard-SVM classifier? What is
     the training accuracy? Do you expect it to generalize well in the unseen dataset?
   - Compare the margins of the hard SVM and soft SVM. Which of the two will will result in a weight vector of larger magnitude? Justify your reasoning

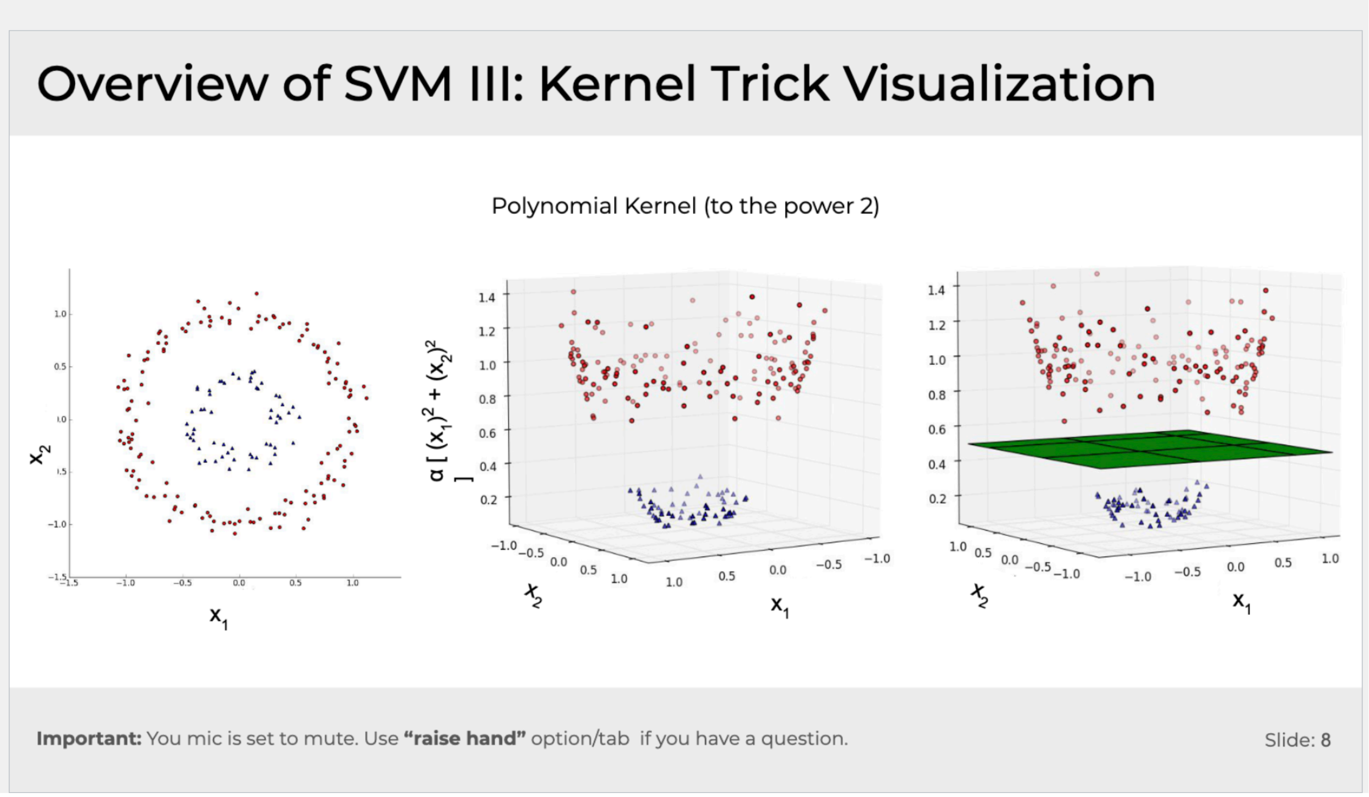
2. For the following feature space plots, identify the possible types of SVM (hard margin linear, soft margin linear, kernel) that we can use to classify the data.







3.

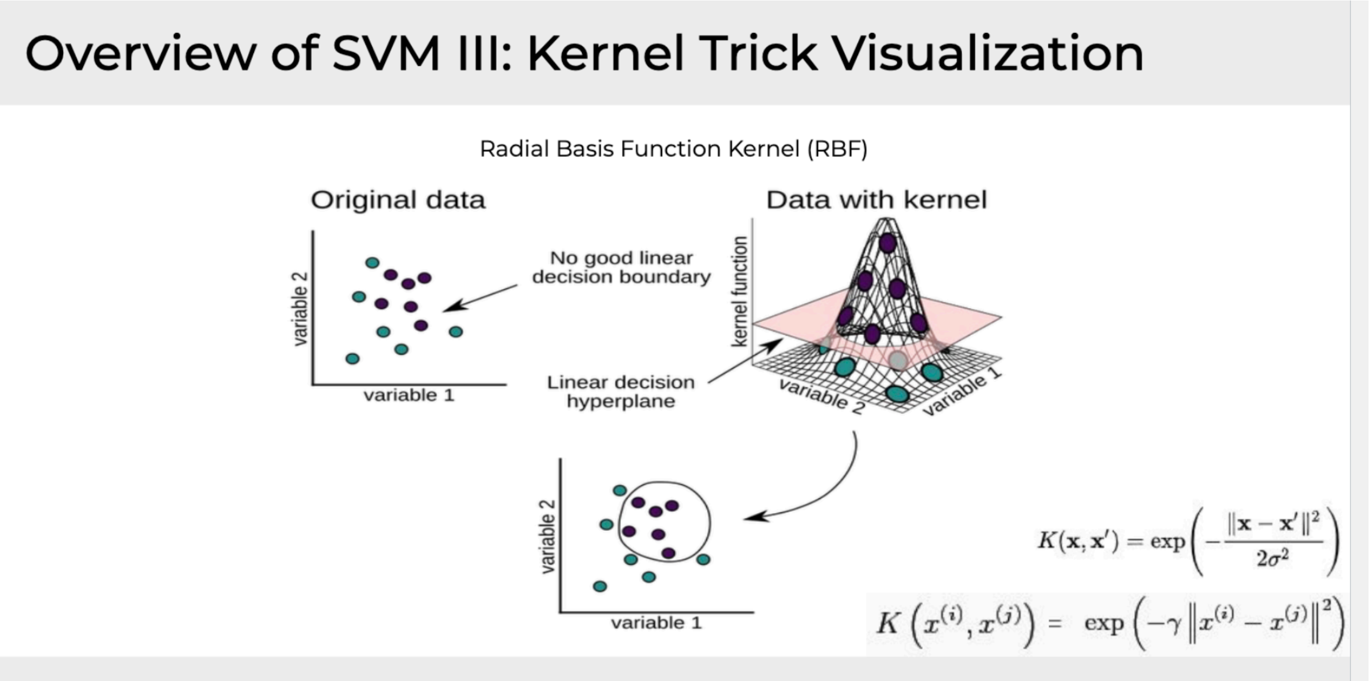


Overview of SVM III: Kernel Trick Visualization

Polynomial Kernel (to the power 2)

Important: You mic is set to mute. Use **“raise hand”** option/tab if you have a question.

4.



Overview of SVM III: Kernel Trick Visualization

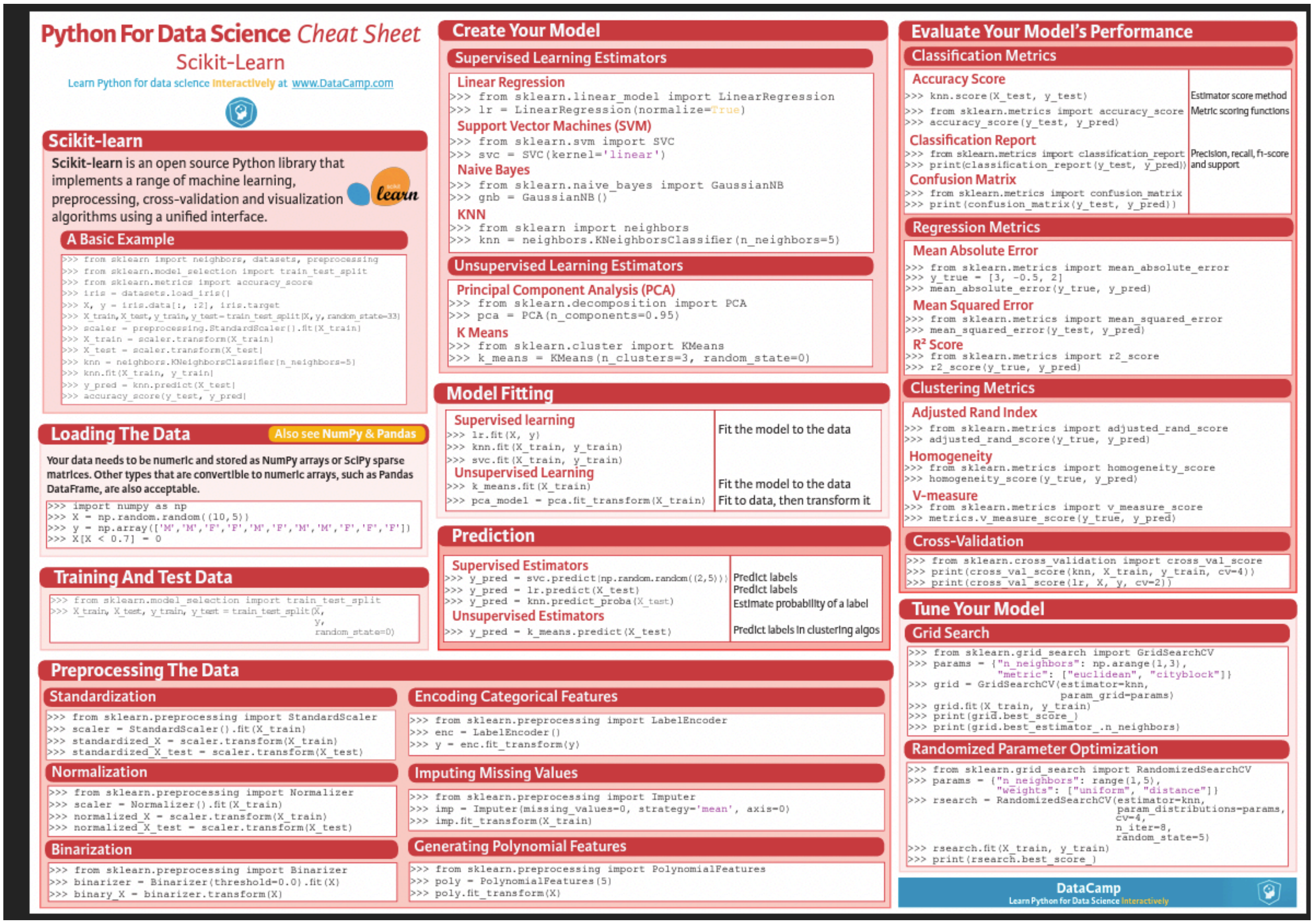
Radial Basis Function Kernel (RBF)

No good linear decision boundary

Linear decision hyperplane

$$K(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$$

$$K\left(x^{(i)},x^{(j)}\right) = \exp\left(-\gamma\left\|x^{(i)}-x^{(j)}\right\|^2\right)$$



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                         y,
                                                         random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	

`>>> lr.fit(X, y)`
`>>> knn.fit(X_train, y_train)`
`>>> svc.fit(X_train, y_train)`
	Fit the model to the data

Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score	
`>>> knn.score(X_test, y_test)`	Estimator score method

`>>> from sklearn.metrics import accuracy_score`
`>>> accuracy_score(y_test, y_pred)`
	Metric scoring functions

Classification Report	

`>>> from sklearn.metrics import classification_report`
`>>> print(classification_report(y_test, y_pred))`
	Precision, recall, f1-score and support

Confusion Matrix	

`>>> from sklearn.metrics import confusion_matrix`
`>>> print(confusion_matrix(y_test, y_pred))`	

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

Assignment 6

Package: `sklearn.model_selection.GridSearchCV`

`cv_results_`: (example: 4 sets of hyperparameters configuration to test)

```
{
'param_kernel': masked_array(data = ['poly', 'poly', 'rbf', 'rbf'],
                             mask = [False False False False]...)
'param_gamma': masked_array(data = [-- -- 0.1 0.2],
                            mask = [ True  True False False]...),
'param_degree': masked_array(data = [2.0 3.0 -- --],
                             mask = [False False  True  True]...),
'split0_test_score'  : [0.80, 0.70, 0.80, 0.93],
'split1_test_score'  : [0.82, 0.50, 0.70, 0.78],
'mean_test_score'    : [0.81, 0.60, 0.75, 0.85],
'std_test_score'     : [0.01, 0.10, 0.05, 0.08],
'rank_test_score'    : [2, 4, 3, 1],
'split0_train_score' : [0.80, 0.92, 0.70, 0.93],
'split1_train_score' : [0.82, 0.55, 0.70, 0.87],
'mean_train_score'   : [0.81, 0.74, 0.70, 0.90],
'std_train_score'    : [0.01, 0.19, 0.00, 0.03],
'mean_fit_time'      : [0.73, 0.63, 0.43, 0.49],
'std_fit_time'       : [0.01, 0.02, 0.01, 0.01],
'mean_score_time'    : [0.01, 0.06, 0.04, 0.04],
'std_score_time'     : [0.00, 0.00, 0.00, 0.01],
'params'             : [{'kernel': 'poly', 'degree': 2}, ...],
}
```