Introduction to data science

Patrick Shafto

Department of Math and Computer Science

Plan for today

- Syllabus
- Basics of machine learning with SciKit-Learn
- Talk about the homework for Monday

Introduction to machine learning with scikit-learn

- Learning problem considers a set of n samples of data and then tries to predict properties of unknown data.

- If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features or dimensions.

    - This is a very small subset of the problems one may wish to reason about!

Categories of machine learning problems

- supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:
    - classification:
    - regression:
- Unsupervised learning

Categories of machine learning problems

- classification:
- samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data.
- An example of classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories.
- Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

Categories of machine learning problems

- regression:
- if the desired output consists of one or more continuous variables, then the task is called regression.
- An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

Categories of machine learning problems

- unsupervised learning,
- in which the training data consists of a set of input vectors x without any corresponding target values.
- The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization

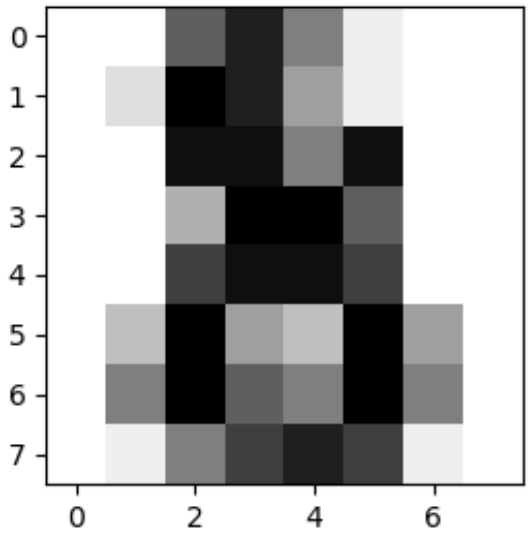
Cross-validation

- Training set and testing set
- Machine learning is about learning some properties of a data set and applying them to new data (generalization).
- This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the training set on which we learn data properties and one that we call the testing set on which we test these properties.

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

```
>>> clf.fit(digits.data[:-1], digits.target[:-1])
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

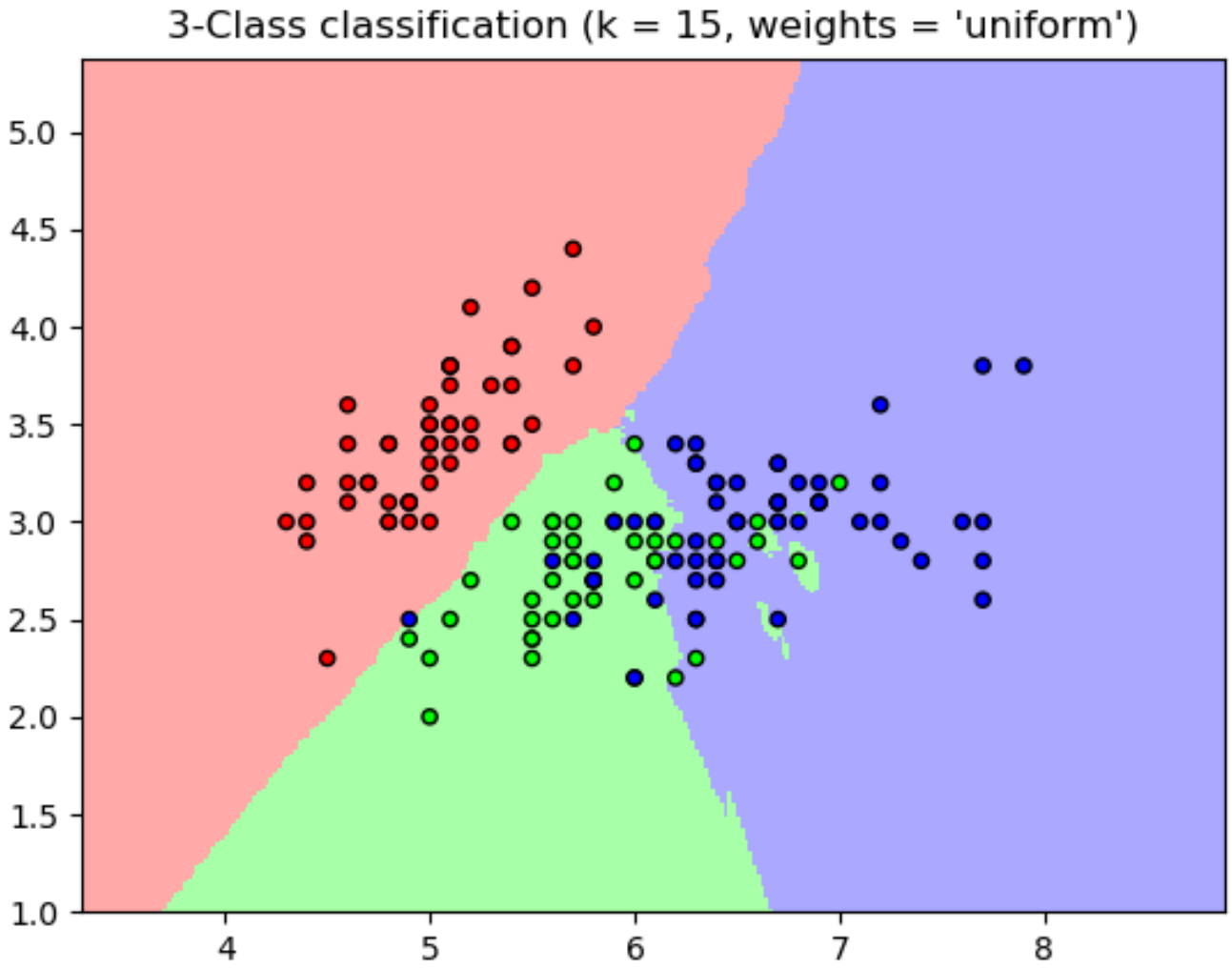
```
>>> clf.predict(digits.data[-1:])
array([8])
```



Model persistence: Pickling

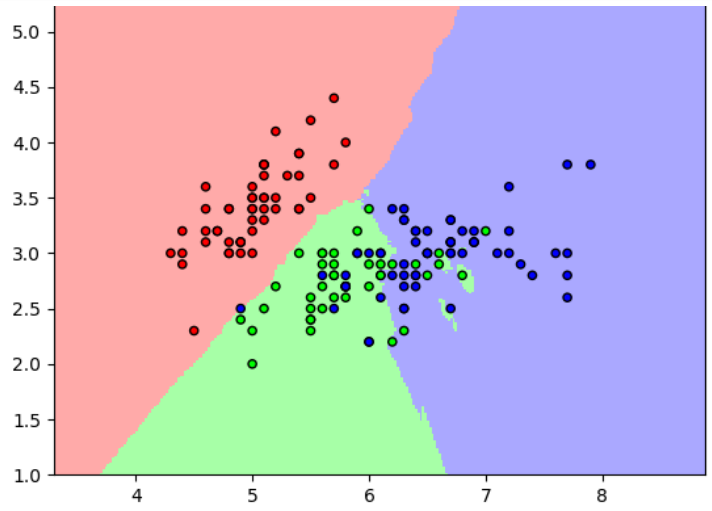
```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC()
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0:1])
array([0])
>>> y[0]
0
```

Supervised learning

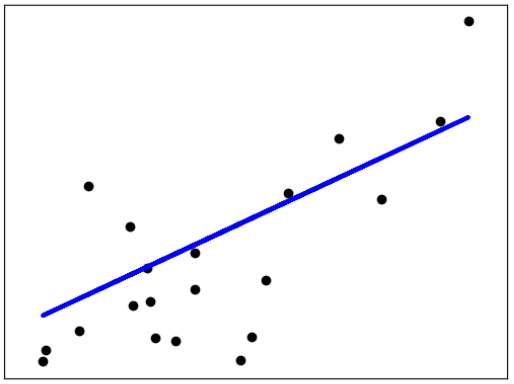


k-nearest neighbors

```
>>> # Split iris data in train and test data
>>> # A random permutation, to split the data randomly
>>> np.random.seed(0)
>>> indices = np.random.permutation(len(iris_X))
>>> iris_X_train = iris_X[indices[:-10]]
>>> iris_y_train = iris_y[indices[:-10]]
>>> iris_X_test  = iris_X[indices[-10:]]
>>> iris_y_test  = iris_y[indices[-10:]]
>>> # Create and fit a nearest-neighbor classifier
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier()
>>> knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
>>> knn.predict(iris_X_test)
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
>>> iris_y_test
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```



Linear regression

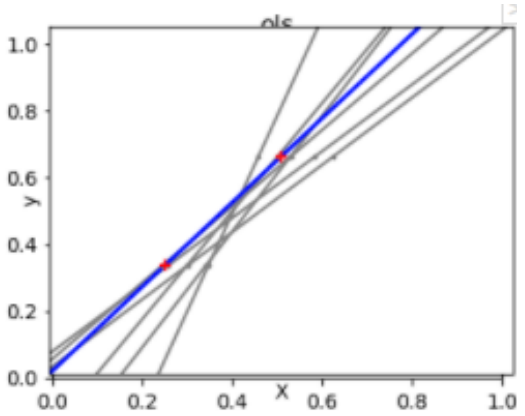


```
>>> diabetes = datasets.load_diabetes()
>>> diabetes_X_train = diabetes.data[:-20]
>>> diabetes_X_test  = diabetes.data[-20:]
>>> diabetes_y_train = diabetes.target[:-20]
>>> diabetes_y_test  = diabetes.target[-20:]
```

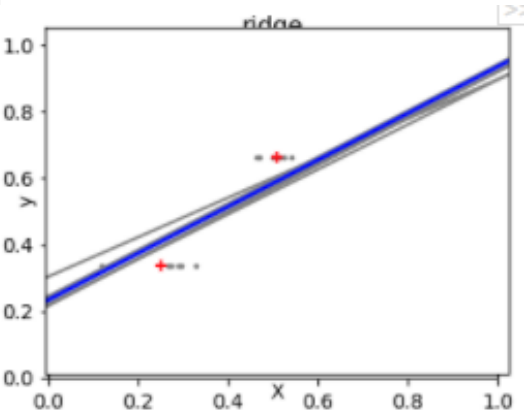
```
>>> from sklearn import linear_model
>>> regr = linear_model.LinearRegression()
>>> regr.fit(diabetes_X_train, diabetes_y_train)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, nor
malize=False)
>>> print(regr.coef_)
[   0.30349955 -237.63931533  510.53060544  327.73698041 -814.1
3170937
  492.81458798  102.84845219  184.60648906  743.51961675   76.0
9517222]

>>> # The mean square error
>>> np.mean((regr.predict(diabetes_X_test)-diabetes_y_test)**2)
2004.56760268...

>>> # Explained variance score: 1 is perfect prediction
>>> # and 0 means that there is no linear relationship
>>> # between X and y.
>>> regr.score(diabetes_X_test, diabetes_y_test)
0.5850753022690...
```

Shrinkage: Ridge regression

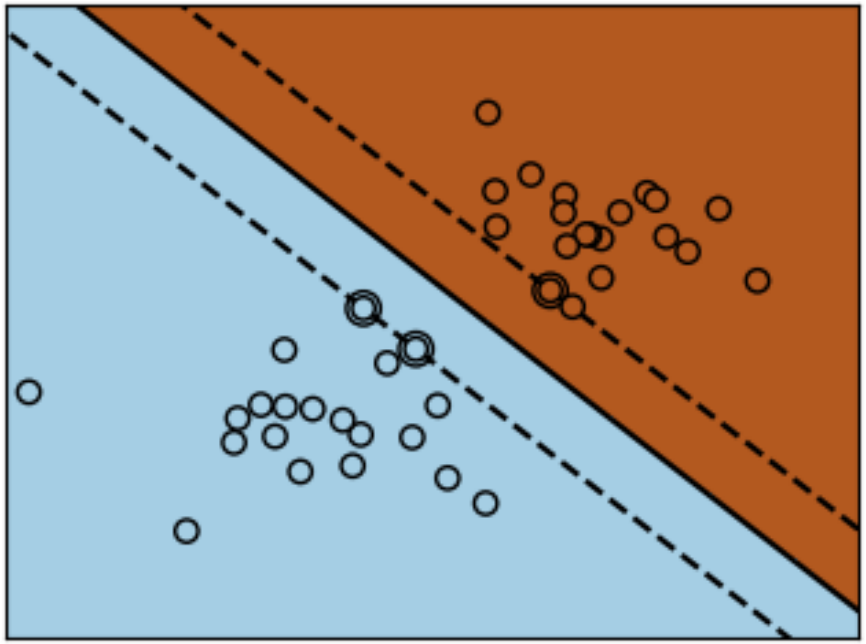
```
>>> X = np.c_[ .5, 1].T
>>> y = [.5, 1]
>>> test = np.c_[ 0, 2].T
>>> regr = linear_model.LinearRegression()

>>> import matplotlib.pyplot as plt
>>> plt.figure()

>>> np.random.seed(0)
>>> for _ in range(6):
...     this_X = .1*np.random.normal(size=(2, 1)) + X
...     regr.fit(this_X, y)
...     plt.plot(test, regr.predict(test))
...     plt.scatter(this_X, y, s=3)
```



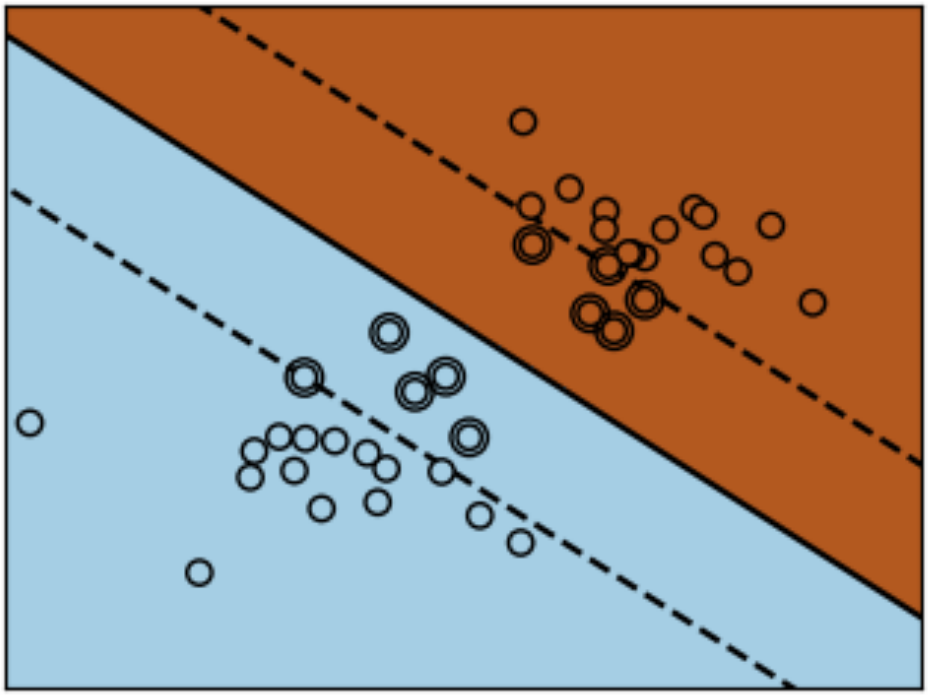

```
>>> regr = linear_model.Ridge(alpha=.1)

>>> plt.figure()

>>> np.random.seed(0)
>>> for _ in range(6):
...     this_X = .1*np.random.normal(size=(2, 1)) + X
...     regr.fit(this_X, y)
...     plt.plot(test, regr.predict(test))
...     plt.scatter(this_X, y, s=3)
```



Classification: Support vector machines



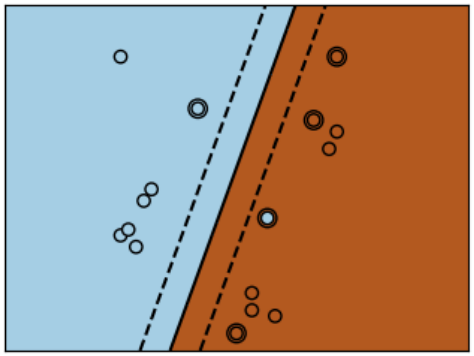
Unregularized



Regularized

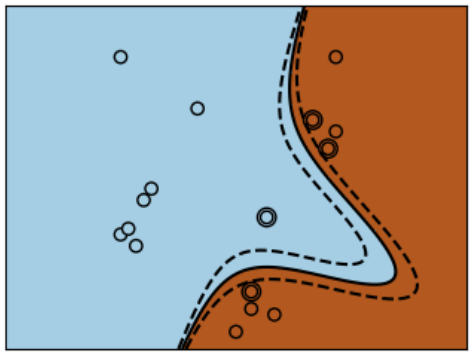
Classification: Support vector machines

```
>>> from sklearn import svm
>>> svc = svm.SVC(kernel='linear')
>>> svc.fit(iris_X_train, iris_y_train)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```



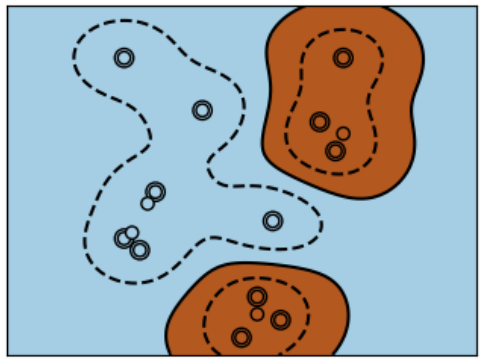
Linear

>>> svc = svm.SVC(kernel='linear')



Polynomial

>>> svc = svm.SVC(kernel='poly', degree=3)



RBF

>>> svc = svm.SVC(kernel='rbf')

Homework for Monday

- Speed talks: 3 minutes each + 1 question
- (No slides!)