Introduction to data science

Patrick Shafto

Department of Math and Computer Science

Plan for today

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

- Rest of the semester!
- Nov 20: Project presentations
- Nov 22: CLASS CANCELED - THANKSGIVING
- Nov 27: 2016 Election polling postmortem
- Nov 29: Deep learning
- Dec 4: Big data
- Dec 6: Final project presentations
- Dec 11: Final project presentations
- Dec 13: Final project presentations

Homework for Monday

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

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!

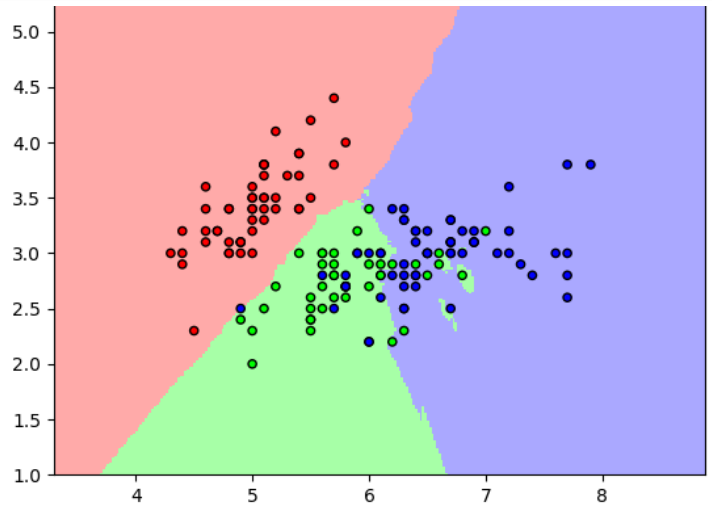
Categories of machine learning problems

- 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

Cross-validation

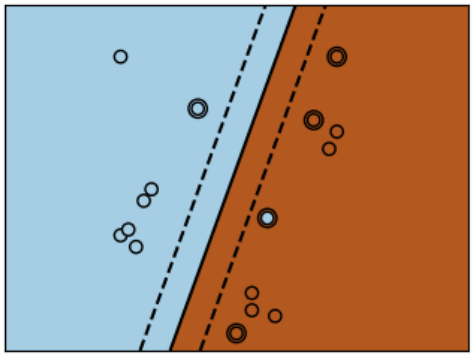
- Training set and testing set
- Machine learning is about learning some properties of a data set and applying them to new data (generalization).
- This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the training set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

```
>>> # 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])
```



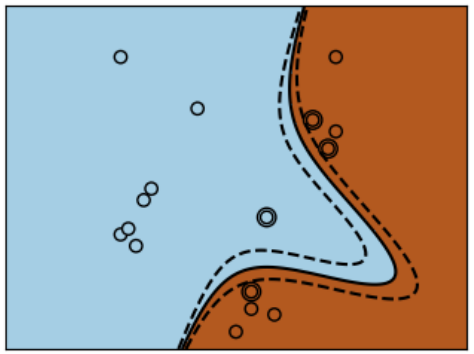
Classification: Support vector machines

```
>>> 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)
```



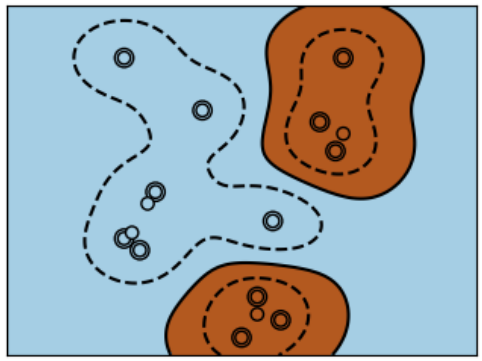
Linear

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



Polynomial

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



RBF

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

K-fold cross validation

```
>>> from sklearn.model_selection import KFold, cross_val_score
>>> X = ["a", "a", "b", "c", "c", "c"]
>>> k_fold = KFold(n_splits=3)
>>> for train_indices, test_indices in k_fold.split(X):
...       print('Train: %s | test: %s' % (train_indices, test_indices))
Train: [2 3 4 5] | test: [0 1]
Train: [0 1 4 5] | test: [2 3]
Train: [0 1 2 3] | test: [4 5]
```

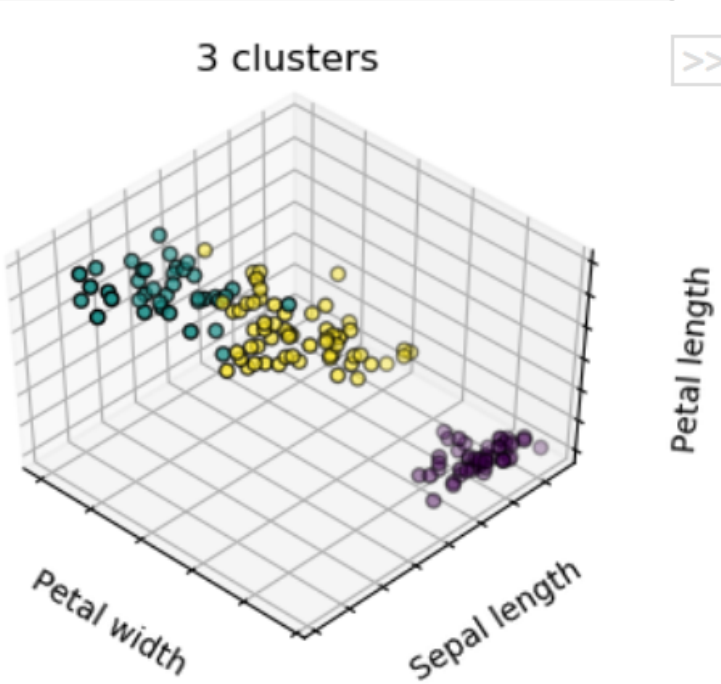
```
>>> [svc.fit(X_digits[train], y_digits[train]).score(X_digits[test], y_digits[test])
...           for train, test in k_fold.split(X_digits)]
[0.93489148580968284, 0.95659432387312182, 0.93989983305509184]
```

K-fold cross validation

```
>>> from sklearn import linear_model, datasets
>>> lasso = linear_model.LassoCV()
>>> diabetes = datasets.load_diabetes()
>>> X_diabetes = diabetes.data
>>> y_diabetes = diabetes.target
>>> lasso.fit(X_diabetes, y_diabetes)
LassoCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_intercept=True,
    max_iter=1000, n_alphas=100, n_jobs=1, normalize=False, positive=False,
    precompute='auto', random_state=None, selection='cyclic', tol=0.0001,
    verbose=False)
>>> # The estimator chose automatically its lambda:
>>> lasso.alpha_
0.01229...
```

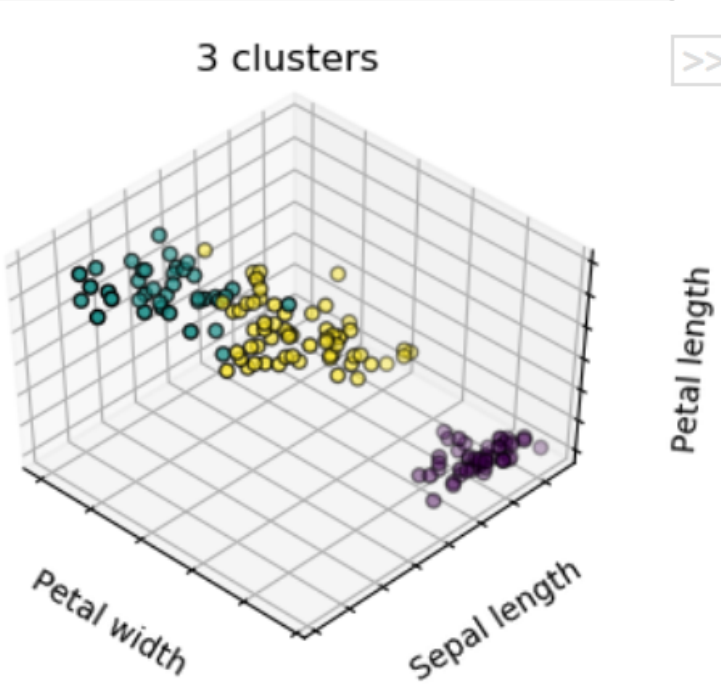
Unsupervised: K means

```
>>> from sklearn import cluster, datasets
>>> iris = datasets.load_iris()
>>> X_iris = iris.data
>>> y_iris = iris.target

>>> k_means = cluster.KMeans(n_clusters=3)
>>> k_means.fit(X_iris)
KMeans(algorithm='auto', copy_x=True, init='k-means++', ...
>>> print(k_means.labels_[::10])
[1 1 1 1 1 0 0 0 0 0 2 2 2 2 2]
>>> print(y_iris[::10])
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
```



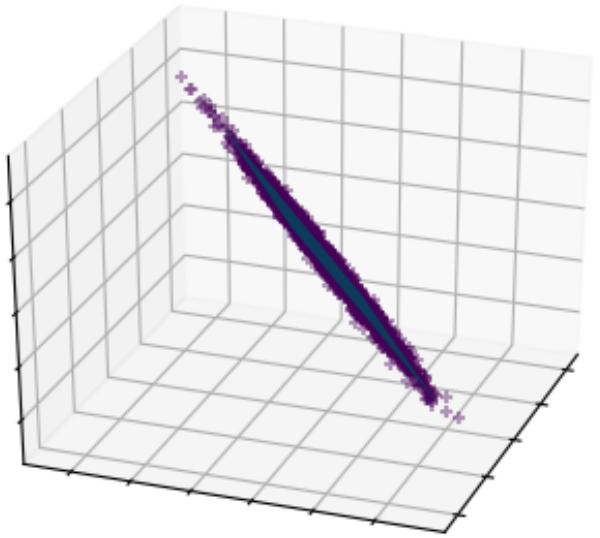
Unsupervised: K means

```
>>> from sklearn import cluster, datasets
>>> iris = datasets.load_iris()
>>> X_iris = iris.data
>>> y_iris = iris.target

>>> k_means = cluster.KMeans(n_clusters=3)
>>> k_means.fit(X_iris)
KMeans(algorithm='auto', copy_x=True, init='k-means++', ...
>>> print(k_means.labels_[::10])
[1 1 1 1 1 0 0 0 0 0 2 2 2 2 2]
>>> print(y_iris[::10])
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
```



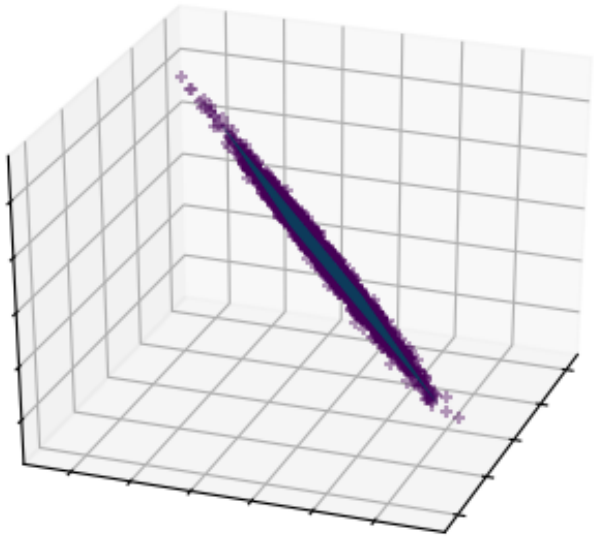
Unsupervised: PCA

```
>>> # Create a signal with only 2 useful dimensions
>>> x1 = np.random.normal(size=100)
>>> x2 = np.random.normal(size=100)
>>> x3 = x1 + x2
>>> X = np.c_[x1, x2, x3]

>>> from sklearn import decomposition
>>> pca = decomposition.PCA()
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
>>> print(pca.explained_variance_)
[  2.18565811e+00   1.19346747e+00   8.43026679e-32]

>>> # As we can see, only the 2 first components are useful
>>> pca.n_components = 2
>>> X_reduced = pca.fit_transform(X)
>>> X_reduced.shape
(100, 2)
```



Unsupervised: PCA

```
>>> # Create a signal with only 2 useful dimensions
>>> x1 = np.random.normal(size=100)
>>> x2 = np.random.normal(size=100)
>>> x3 = x1 + x2
>>> X = np.c_[x1, x2, x3]

>>> from sklearn import decomposition
>>> pca = decomposition.PCA()
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
>>> print(pca.explained_variance_)
[  2.18565811e+00   1.19346747e+00   8.43026679e-32]

>>> # As we can see, only the 2 first components are useful
>>> pca.n_components = 2
>>> X_reduced = pca.fit_transform(X)
>>> X_reduced.shape
(100, 2)
```



Precision and recall

$$P = \frac{T_p}{T_p + F_p}$$

True positive over true positive plus false positive

$$R = \frac{T_p}{T_p + F_n}$$

True positive over true positive plus false negative

		Truth: T	Truth: F
Decision	P	T_P	F_P
	N	F_N	T_N

Pipelining!

http://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html

Extra work: For fun

- Apply K nearest neighbors and a support vector machine to the digits data and the iris data.
- Which algorithm performs better on each?
  - Show pictures representing performance
  - Also summarize performance using appropriate numbers
- Explain the results, what they mean, and any performance differences.

Homework for Monday

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

Format for final presentations

- Evaluate your classmates using the rubric!
- You must attend class to get credit