Lecture 15: Principal Component Analysis

CS 167: Machine Learning

Upcoming Schedule

Exam: Thursday, November 17th in class.

Project 4: It's posted. Due Tuesday, November 22nd (end of the day).

Project 5: Not posted yet. Will be due last week of class or finals week.

Review: Let's continue with SVMs for a minute

True/False: SVM uses a linear separator.

When trying to avoid overfitting with an SVM, which parameters should I tweak?

Recall: Some common kernels

Polynomial:

$$K(X, Y) = (X \cdot Y + r)^d \quad \text{(for } d,\ r \text{ constants)}$$

Gaussian radial basis function:

$$K(X, Y) = \exp(-\gamma \|X - Y\|^2) \quad \text{(for } \gamma > 0\text{)}$$

Hyperbolic tangent (or sigmoid):

$$K(X, Y) = \tanh(\gamma X \cdot Y + r) \quad \text{(for } \gamma > 0,\ r < 0\text{)}$$

SVM in scikit-learn

The documentation on the SVM classifier in scikit-learn is here:
`http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`
Answer the following:

- Which kernels are supported? What is the default?
- What are the default values of C, γ, r, and d? Do they all have meaning for all kernels?

Exercises:

- Use the linear kernel (i.e., just dot product) with the iris data. How does it do?
- Try the quadratic kernel (i.e., polynomial kernel with $d = 2$, $r = 1$). How does it do?
- Try it on the LFW data set.

Review: Supervised vs. Unsupervised Learning

Supervised Learning: Training examples are all labeled with the right target values: e.g., classification and regression

Unsupervised Learning: no labels, analyze/cluster examples. Example: what voting blocks exist in Congress?

Principal Component Analysis is an *unsupervised learner* - it doesn't use a target column

Reducing Dimensionality

Sometimes, we want to *reduce the dimensionality* of the data

- Visualize high-dimensional data in 2D or 3D
- Reduce noise
- Better/faster learning - removing irrelevant features

When have we already done this?

Examples

Let's look at ways of doing this with some plots on the whiteboard.

Measurable vs. Latent Attributes

Regression problem: predict the price of a house based on the following *measurable attributes*

- house square footage
- number of rooms
- school quality
- neighborhood safety

My Claim: There are really two *latent attributes* which explain patterns in these four *measurable attributes*.

Discuss: What are they?

How it works

Let's go over how it works with some whiteboard plots.

PCA Take-Aways

- Transform data into new attributes: the *principal components*
- **principal components**: axes that maximize variance (minimize information loss) when you project onto them
- highest variance is **first principal component**
- second highest variance that doesn't overlap with first is **second principal component**
- and so on until you have the desired number of attributes
  - all PCs are *orthogonal* (perpendicular) to each other, so are *linearly uncorrelated*

Try it in scikit-learn: Front Matter

```
import pandas
#from sklearn import cross_validation as cv
from sklearn import model_selection as cv
from sklearn import metrics
from sklearn.svm import SVC
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA #new one!

iris_data = pandas.read_csv('irisData.csv')

predictors = ["sepal length","sepal width",\
              "petal length","petal width"]

(train, test) = cv.train_test_split(iris_data, test_size = 0.2)
```

PCA in scikit-learn

```
#whiten = True is important for uncorrelated
#attributes, and is False by default
pca = PCA(n_components=2, whiten=True)
pca.fit(train[predictors])
transformed_train_data = pca.transform(train[predictors])
transformed_test_data = pca.transform(test[predictors])

#this is the variance/importance of each component
print(pca.explained_variance_ratio_)

print(pca.components_[0])
print(pca.components_[1])
```


Visualizing the new attributes

```
#PCA gives it back as numpy array
tdf = pandas.DataFrame(transformed_train_data)
#next line: probably not the best way
tdf['species'] = pandas.Series(list(train['species']))
print(tdf)


setosa_series = tdf[ tdf['species'] == 'Iris-setosa' ]
virginica_series = tdf[ tdf['species'] == 'Iris-virginica' ]
versicolor_series = tdf[ tdf['species'] == 'Iris-versicolor']

plt.plot(setosa_series[0],setosa_series[1],'ro')
plt.plot(virginica_series[0],virginica_series[1],'bs')
plt.plot(versicolor_series[0],versicolor_series[1],'g^')
plt.show()
```


Can we learn with the new attributes?

```
clf = SVC(kernel='linear')
clf.fit(transformed_train_data,train['species'])
predictions = clf.predict(transformed_test_data)
print(metrics.accuracy_score(test['species'], predictions))
```