CS167: Machine Learning

Dimensionality Reduction Techniques:

Feature Selection

Feature Extraction

Thursday, March 21st, 2024

Drake UNIVERSITY

Announcements

- Project #1
    - Deadline: due next Thursday April 04 by 11:59pm
    - To submit, download the `ipynb` file from Colab

Today’s Agenda

- Dimensionality Reduction
  - Feature Selection
  - Feature Extraction
    - Principle Component Analysis (PCA)

Dimensionality of Data

- **Dimensionality:** the number of attributes (or features) that a dataset has

- **High dimensionality:** If the number of attributes (i.e. columns) is higher than the number of observations (i.e. rows) the dimensionality of the data is very high (i.e. *healthcare data*, *gene expression*,
    - Pros: more data provides more attributes to 'learn' from
    - Cons: more compute time

- **Low dimensionality:** if the number of attributes (i.e. columns) is relatively small compared to the number of rows. (i.e. *Iris Dataset*)
    - Pros: simpler data, less compute time
    - Cons: less likely to be easily separable

Curse of Dimensionality

- The **curse of dimensionality:** the more dimensions you add to a dataset, the more difficult it becomes to make predictions about that dataset.
    - each attribute added results an an exponential decrease in predictive power.

Overview

- The next two lectures (including today), we're going to be playing around with the **dimensionality** of our datasets.
    - **Principal Component Analysis**: decreases the dimensionality of our datasets
    - **Support Vector Machines**: increase dimensionality so that our data can be linearly separable

Dimensionality of Data

- When working with datasets that have many variables (i.e. columns), it is often advantageous to "reduce the dimension of your feature space"-- or in other words, focus on a smaller subset of the most important variables

- Reducing the dimension of the feature space is called **dimensionality reduction**
  - Visualize high-dimensional data in 2D or 3D
  - Reduce noise
  - Better/faster learning - removing irrelevant features

Dimensionality Reduction Techniques

- **Feature Selection/Elimination:** choose which features are important

- **Feature Extraction:** transforming raw data into numeric features that can be processed while preserving the information in the original dataset

Group Discussion

- See if you and your group can come up with some ways to tell how important a variable (i.e. column) is for making a machine learning prediction

- Ideas:
    - **Manual selection:** Try machine learning with one column at a time… pick the columns that give you the best performance
    - **Build a decision tree or Random Forest:** look at the `feature_importances_` attribute (which is built from **information gain**)
    - **Statistical tests:** *chi squared*, *F-value*, *etc*

Today's Agenda

- Dimensionality Reduction
  - Feature Selection
  - Feature Extraction
    - Principle Component Analysis (PCA)

Dimensionality Reduction Technique #1: Feature Selection

- **Feature Selection/Elimination:** choose which features are important

- **Advantages of Feature Selection/Elimination:**
  - Simplicity--easily interpretable
  - maintaining the interpretability of your variables (in comparison to feature extraction)

- **Disadvantages of Feature Selection/Elimination:**
  - you lose data by dropping columns

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]

- Goto blackboard -> make a copy of the class notebook and run the code

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import accuracy_score
```

new module for feature selection

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]

```
path = '/content/drive/MyDrive/cs167_fall23/datasets/irisData.csv'
data = pandas.read_csv(path)
predictors = ['sepal length', 'sepal width', 'petal length', 'petal width']
target = "species"

train_data, test_data, train_sln, test_sln = \
    train_test_split(data[predictors], data[target], test_size = 0.2, random_state=41)
```

```
train_data.head(5)
```

	sepal length	sepal width	petal length	petal width
79	5.7	2.6	3.5	1.0
54	6.5	2.8	4.6	1.5
106	4.9	2.5	4.5	1.7
90	5.5	2.6	4.4	1.2
145	6.7	3.0	5.2	2.3

Full columns (or features)

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]()

```
# fit your selector just like you do when training with a classifier/regressor
# only do this after splitting into train and test sets - don't let the test
# set spoil your predictions
selector = SelectKBest(k=2)
selector.fit(train_data,train_sln)

# bigger number means the feature is more important
print('Here are the scores of each feature:')
print(selector.scores_)
print(predictors)
```

create an object for feature selection

fit the feature selector on train_data

see the scores of each feature

```
Here are the scores of each feature:
[ 83.17181699  48.65999233 962.36229917 894.63459428]
['sepal length', 'sepal width', 'petal length', 'petal width']
```

best feature **second best feature**

Why only 2?

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]()

```
#transforming the predictor columns of the training set
train_transformed = selector.transform(train_data)

print("Here's what the training predictors look like after the transformation. \
Notice that it's just the last two columns from the original data.")
train_transformed[0:5]
```

apply the feature selector on train_data

```
[[3.5, 1. ],
 [4.6, 1.5],
 [4.5, 1.7],
 [4.4, 1.2],
 [5.2, 2.3]])
```

second best feature

best feature

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: sklearn.feature_selection.SelectKBest()

```
#take a look at the training data
train_data[0:5]
```

	sepal length	sepal width	petal length	petal width
79	5.7	2.6	3.5	1.0
54	6.5	2.8	4.6	1.5
106	4.9	2.5	4.5	1.7
90	5.5	2.6	4.4	1.2
145	6.7	3.0	5.2	2.3

best feature

second best feature

```
[[3.5, 1. ],
 [4.6, 1.5],
 [4.5, 1.7],
 [4.4, 1.2],
 [5.2, 2.3]])
```

best feature

second best feature

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]

```
#Now we transform the predictor columns in the test set as well.
#Notice that we're using the selector that we trained using the training set.
#Do not re-fit it to the test data.
test_transformed = selector.transform(test_data)

#Now we can use our transformed data with a classifier just like always:
clf = KNeighborsClassifier()
clf.fit(train_transformed,train_sln)
predictions = clf.predict(test_transformed)
print('Accuracy:',accuracy_score(test_sln,predictions))
```

apply the feature selector on test_data

```
Accuracy: 0.9333333333333333
```

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

Documentation: [sklearn.feature_selection.SelectKBest()]

- Let's compare it to a model trained on all of the data:

```
clf = KNeighborsClassifier()
clf.fit(train_data,train_sln)
predictions = clf.predict(test_data)
print('Accuracy:',accuracy_score(test_sln,predictions))
```

```
Accuracy: 0.9666666666666667
```

Dimensionality Reduction Technique #1: Feature Selection Code

- **Feature Selection/Elimination:** choose which features are important

  Documentation: [sklearn.feature_selection.SelectKBest()]()

- Feature Selection Code (all together)

```
# create an instance of 'SelectKBest'
selector = SelectKBest(k=2)

# fit it on your train data and solutions
selector.fit(train_data,train_sln)

# transform your traning data to only have the k best attributes
train_transformed = selector.transform(train_data)

#build a model
clf = KNeighborsClassifier()
clf.fit(train_transformed,train_sln)
predictions = clf.predict(test_transformed)
print('Accuracy:',accuracy_score(test_sln,predictions))
```

```
Accuracy: 0.9333333333333333
```

Accuracy went down (from 96.67% to 93.33%) after feature selection, but we have fewer features now

Group Exercise #1

- Let's give it a shot:
    - below, I went ahead and loaded in the penguin dataset 🐧
    - Using `species` as the target variable, what are the 3 best attributes?
    - Build a default Random Forest using only the 3 best attributes. How does the performance compare to a default random forest that uses all of the predictor variables?

- Finish the empty block on your class notebook

```
# create an instance of 'SelectKBest' with 3 seleted features

# fit it on your train data and solutions

# transform your train and test data to only have the k best attributes

# train a random forest classifier with the k best attributes

# evalute your trained random forest in accuracy metric
```

Today’s Agenda

- Dimensionality Reduction
  - Feature Selection
  - Feature Extraction
    - Principle Component Analysis (PCA)

Dimensionality Reduction Technique #2: Feature Extraction

- **Feature extraction** takes the existing (usually high dimensional) dataset, and returns a dataset such that there are new columns of data that are ordered from most important to least important.

- If you are working with high dimensional data, it is often advantageous to do some **feature extraction** before building and testing your machine learning models.

- **Big Idea**: Find new (or *latent* features) made up of combinations of existing features.
  - Maybe multiplying `sepal length*petal width` is more helpful in identifying the species of an iris than either `sepal length` or `petal width` are on their own

Dimensionality Reduction Technique #2: Feature Extraction

- **Feature extraction:** measurable feature vs. latent features

- Imagine we are attempting to predict the price of a house based on the following measurable features:
    - *house square footage*
    - *number of rooms*
    - *school district test scores*
    - *neighborhood crime rates*

Group Discussion

- **claim**: In the 4 *measurable features*, there are really only **2 latent features** which explain these four measurable features. In other words, there are two composite features that more directly probe the underlying phenomenon of the data

    - *house square footage*
    - *number of rooms*
    - *school district test scores*
    - *neighborhood crime rates*

- Can you see the pattern and guess what these **two latent features** are?

Group Discussion

- **claim**: In the 4 *measurable features*, there are really only **two latent features** which explain these four measurable features. In other words, there are two composite features that more directly probe the underlying phenomenon of the data

- Can you see the pattern and guess what these **two latent features** are?

  - Size of house:
    - *house square footage*
    - *number of rooms*

  - Location of house:
    - *school district test scores*
    - *neighborhood crime rates*

Today's Agenda

- Dimensionality Reduction
  - Feature Selection
  - Feature Extraction
    - Principle Component Analysis (PCA)

Principle Component Analysis (PCA)

- Principal Component Analysis is an **unsupervised** algorithm as it doesn't use a target column:
  - PCA is a **feature extraction** technique
  - PCA is also a **preprocessing** technique (something that you do during data prep, before building/running your model)

- **Big Idea:** Can we extract information from the data that might prove to be more useful?
  - Reduce dimensions of inputs to learning algorithm
  - Easier to understand and graph
  - Reduce noise

Principle Component Analysis (PCA)

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

- **Advantages of Feature Feature Extraction (PCA):**
  - Minimal data loss
  - Output is a transformed data ordered by how well each component predicts the dependent variable

- **Disadvantages of Feature Extraction (PCA):**
  - data becomes much less interpretable

Principle Component Analysis (PCA)

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

- Calculating PCA requires a relatively deep background in linear algebra-- calculating the eigenvectors and their corresponding eigenvalues of covariance matrices. So... we're going to stick to a practical level of understanding.

https://setosa.io/ev/principal-component-analysis/

When should we use PCA?

- Ask yourself these questions:
  - Do you want to **reduce the number of variables**, but aren't able to identify variables to completely remove from consideration?
  - Are you comfortable making your independent variables **less interpretable**?
  - Do you want to ensure your variables are **independent of one another**?
    - **independence**: variables are independent if and only if the occurrence of one does not affect the probability of the occurrence of the other

- If the answers to the above questions are yes, then doing a PCA on your data before you build/run your model is probably a good idea

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]()

```
import pandas
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
```

new module for PCA

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]

```
path = '/content/drive/MyDrive/cs167_fall23/datasets/irisData.csv'
data = pandas.read_csv(path)
predictors = ['sepal length', 'sepal width', 'petal length', 'petal width']
target = "species"

train_data, test_data, train_sln, test_sln = \
    train_test_split(data[predictors], data[target], test_size = 0.2, random_state=41)
```

```
train_data.head(5)
```

	sepal length	sepal width	petal length	petal width
79	5.7	2.6	3.5	1.0
54	6.5	2.8	4.6	1.5
106	4.9	2.5	4.5	1.7
90	5.5	2.6	4.4	1.2
145	6.7	3.0	5.2	2.3

Full columns (or features)

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]

```
# whiten = True is important for uncorrelated
# attributes, and is False by default
pca_extractor = PCA(n_components=2, whiten=True)
# When fitting with PCA, you do not use the target column -
#               this is an unsupervised learning algorithm
pca_extractor.fit(train_data)

print('this is the variance/importance of each component')
print(pca_extractor.explained_variance_ratio_)
```

```
this is the variance/importance of each component
[0.92185361 0.05522532]
```

create an object for PCA

fit the PCA on train_data

see the importance (variance) of each component

Variance of first principle component

Variance of second principle component

Why only 2?

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]()

```
train_transformed = pca_extractor.transform(train_data)

print("Here's what the training predictors look \
like after the transformation.")
train_transformed[0:5]
```

apply the learned principle components on train_data

```
[[-0.14247446, -0.74168385],
 [ 0.52354614,  0.15216911],
 [ 0.25775022, -2.45414817],
 [ 0.22837852, -1.3551112 ],
 [ 0.93305811,  0.32233582]]
```

Transformed train_data along the first principle component

Transformed train_data along the second principle component

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]()

```
#take a look at the training data
train_data[0:5]
```

	old dimension$_1$	old dimension$_2$	old dimension$_3$	old dimension$_4$
	sepal length	**sepal width**	**petal length**	**petal width**
79	5.7	2.6	3.5	1.0
54	6.5	2.8	4.6	1.5
106	4.9	2.5	4.5	1.7
90	5.5	2.6	4.4	1.2
145	6.7	3.0	5.2	2.3

new dimension$_1$ new dimension$_2$

```
[[-0.14247446, -0.74168385],
 [ 0.52354614,  0.15216911],
 [ 0.25775022, -2.45414817],
 [ 0.22837852, -1.3551112 ],
 [ 0.93305811,  0.32233582]]
```

Transformed train_data along the first principle component

Transformed train_data along the second principle component

Each training sample is 4 dimensional data (shown above); each is then **transformed** into a 2 dimensional data (shown left). Dimension is reduced (4D —>2D)

PCA Code

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

Documentation: [sklearn.decomposition.PCA()]()

```python
#Now we transform the predictor columns in the test set as well.
#Notice that we're using the extractor that we trained using the training set.
#Do not re-fit it to the test data.
test_transformed = pca_extractor.transform(test_data)

#Now we can use our transformed data with a classifier just like always.
clf = KNeighborsClassifier()
clf.fit(train_transformed,train_sln)
predictions = clf.predict(test_transformed)
print('Accuracy:',accuracy_score(test_sln,predictions))
```

```
Accuracy: 0.9333333333333333
```

apply the learned principle components on test_data

Two PCA Axes (each 4D) Learned from Train Data

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

Documentation: [sklearn.decomposition.PCA()]()

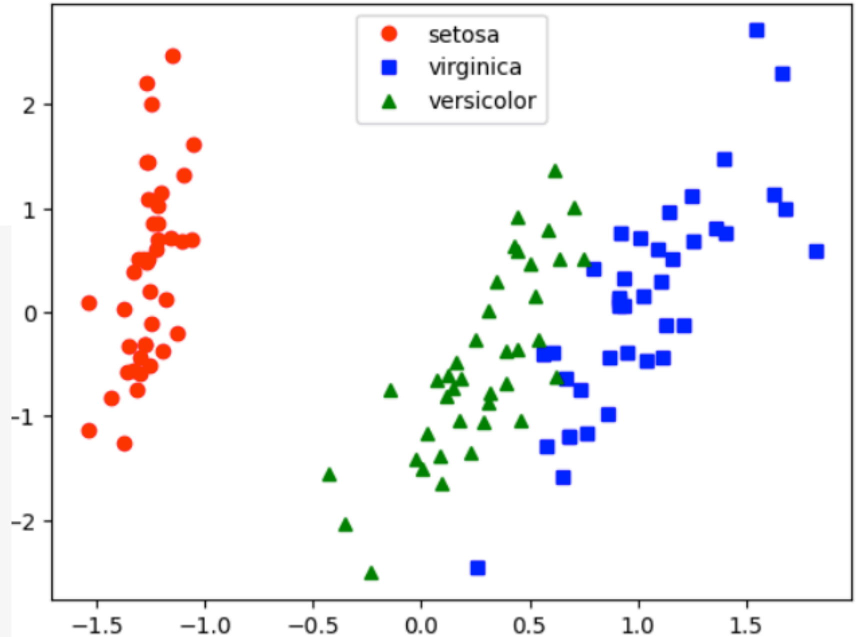
```python
print('Here are the two vectors (in the original space) that define our 2 new axes:')
print(pca_extractor.components_[0])
print(pca_extractor.components_[1])
```

```
Here are the two vectors (in the original space) that define our 2 new axes:
[ 0.35503041 -0.09364147  0.85845905  0.35809601]
[ 0.6991275   0.68599282 -0.16756386 -0.11205774]
```

Visualizing the PCA-Transformed Train Data

Documentation: [sklearn.decomposition.PCA()]

- Another benefit: we can also visualize the lower-dimensional data after it has been transformed via PCA.




```
import matplotlib.pyplot as plt
%matplotlib inline

#visualizing the new axes
#PCA gives it back as numpy array
tdf = pandas.DataFrame(train_transformed)
#next line: probably not the best way
tdf['species'] = pandas.Series(list(train_sln))


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',label='setosa')
plt.plot(virginica_series[0],virginica_series[1],'bs',label='virginica')
plt.plot(versicolor_series[0],versicolor_series[1],'g^',label='versicolor')
plt.legend(loc='upper center')
plt.show()
```