04d_DEMO_Dimensionality_Reduction

May 24, 2022

1 Machine Learning Foundation

1.1 Course 4, Part c: Dimensionality Reduction DEMO

1.2 Introduction

We will be using customer data from a Portuguese wholesale distributor for clustering. This data file is called `Wholesale_Customers_Data`.

It contains the following features:

- Fresh: annual spending (m.u.) on fresh products
- Milk: annual spending (m.u.) on milk products
- Grocery: annual spending (m.u.) on grocery products
- Frozen: annual spending (m.u.) on frozen products
- Detergents_Paper: annual spending (m.u.) on detergents and paper products
- Delicatessen: annual spending (m.u.) on delicatessen products
- Channel: customer channel (1: hotel/restaurant/cafe or 2: retail)
- Region: customer region (1: Lisbon, 2: Porto, 3: Other)

In this data, the values for all spending are given in an arbitrary unit (m.u. = monetary unit).

[1]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

import seaborn as sns, pandas as pd, numpy as np
```

[3]:
```
import os, pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
```

[4]:
```
from colorsetup import colors, palette
sns.set_palette(palette)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_68/3229205981.py in <module>
----> 1 from colorsetup import colors, palette
      2 sns.set_palette(palette)
```

```
ModuleNotFoundError: No module named 'colorsetup'
```

1.3 Part 1

In this section, we will:

- Import the data and check the data types.
- Drop the channel and region columns as they won't be used since we focus on numeric columns for this example.
- Convert the remaining columns to floats if necessary.
- Copy this version of the data (using the `copy` method) to a variable to preserve it. We will be using it later.

[3]:
```
data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBM-ML0187EN-SkillsNetwork/labs/module%203/data/
↪Wholesale_Customers_Data.csv', sep=',')
```

[4]:
```
data.shape
```

[4]:
```
(440, 8)
```

[5]:
```
data.head()
```

[5]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

[6]:
```
data = data.drop(['Channel', 'Region'], axis=1)
```

[7]:
```
data.dtypes
```

[7]:
```
Fresh               int64
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicassen          int64
dtype: object
```

[8]:
```
# Convert to floats
for col in data.columns:
    data[col] = data[col].astype(np.float)
```

Preserve the original data.

[9]:
```python
data_orig = data.copy()
```

1.4 Part 2

As with the previous lesson, we need to ensure the data is scaled and (relatively) normally distributed.

- Examine the correlation and skew.
- Perform any transformations and scale data using your favorite scaling method.
- View the pairwise correlation plots of the new data.

[10]:
```python
corr_mat = data.corr()

# Strip the diagonal for future examination
for x in range(corr_mat.shape[0]):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

[10]:
```
                     Fresh      Milk   Grocery    Frozen  Detergents_Paper  \
Fresh             0.000000  0.100510 -0.011854  0.345881         -0.101953
Milk              0.100510  0.000000  0.728335  0.123994          0.661816
Grocery          -0.011854  0.728335  0.000000 -0.040193          0.924641
Frozen            0.345881  0.123994 -0.040193  0.000000         -0.131525
Detergents_Paper -0.101953  0.661816  0.924641 -0.131525          0.000000
Delicassen        0.244690  0.406368  0.205497  0.390947          0.069291

                  Delicassen
Fresh               0.244690
Milk                0.406368
Grocery             0.205497
Frozen              0.390947
Detergents_Paper    0.069291
Delicassen          0.000000
```

As before, the two categories with their respective most strongly correlated variable.

[11]:
```python
corr_mat.abs().idxmax()
```

[11]:
```
Fresh                         Frozen
Milk                         Grocery
Grocery             Detergents_Paper
Frozen                    Delicassen
Detergents_Paper             Grocery
Delicassen                      Milk
dtype: object
```

Examine the skew values and log transform. Looks like all of them need it.

[12]:
```
log_columns = data.skew().sort_values(ascending=False)
log_columns = log_columns.loc[log_columns > 0.75]

log_columns
```

[12]:
```
Delicassen          11.151586
Frozen               5.907986
Milk                 4.053755
Detergents_Paper     3.631851
Grocery              3.587429
Fresh                2.561323
dtype: float64
```

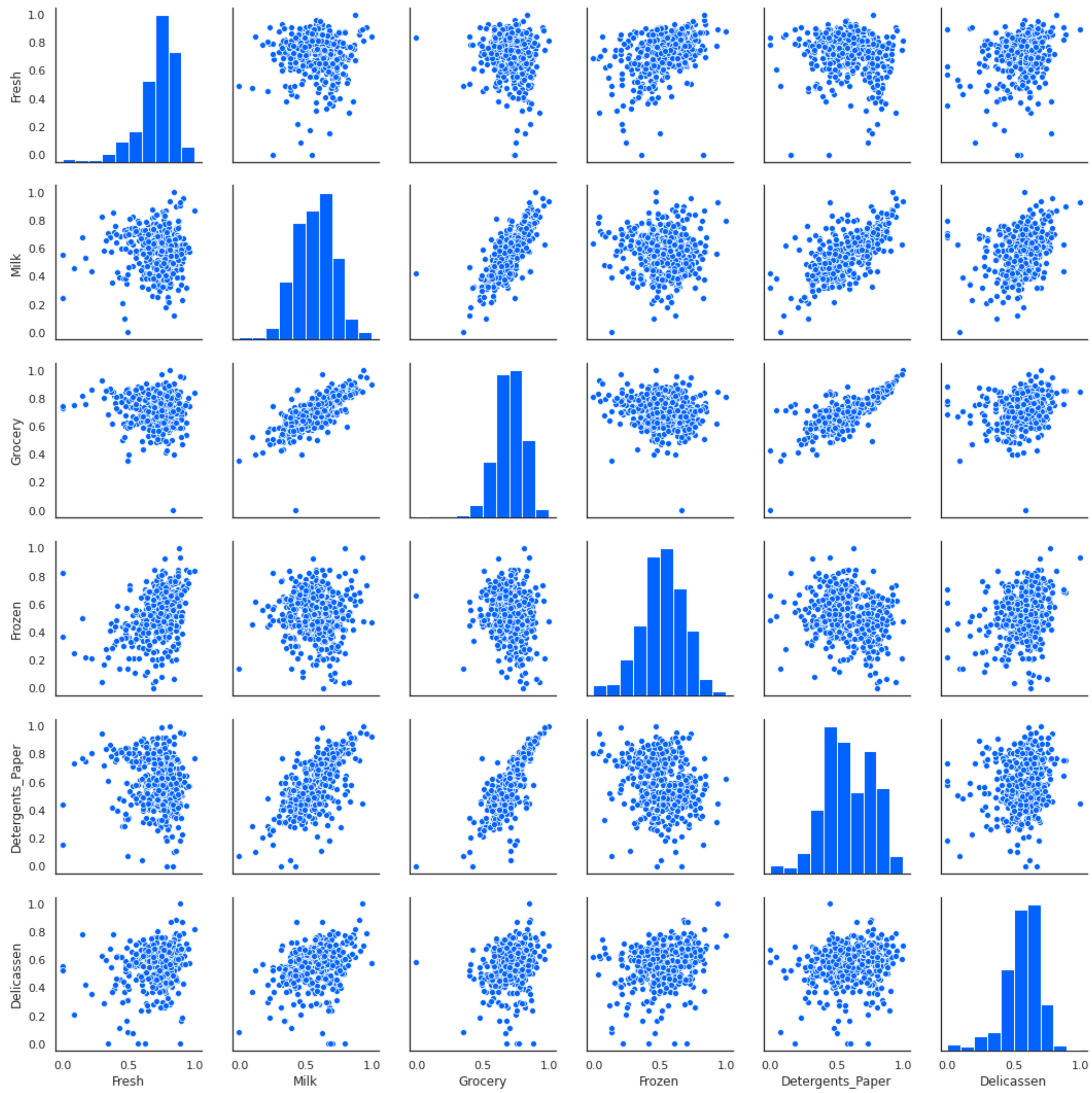
[13]:
```
# The log transformations
for col in log_columns.index:
    data[col] = np.log1p(data[col])
```

Scale the data again. Let's use `MinMaxScaler` this time just to mix things up.

[14]:
```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

for col in data.columns:
    data[col] = mms.fit_transform(data[[col]]).squeeze()
```

Visualize the relationship between the variables.

[15]:
```
sns.set_context('notebook')
sns.set_style('white')
sns.pairplot(data);
```



1.5 Part 3

In this section, we will:

- Using Scikit-learn's pipeline function, recreate the data pre-processing scheme above (transformation and scaling) using a pipeline. If you used a non-Scikit learn function to transform the data (e.g. NumPy's log function), checkout the custom transformer class called `FunctionTransformer`.
- Use the pipeline to transform the original data that was stored at the end of question 1.
- Compare the results to the original data to verify that everything worked.

Note: Scikit-learn has a more flexible `Pipeline` function and a shortcut version called `make_pipeline`. Either can be used. Also, if different transformations need to be performed on the data, a `FeatureUnion` can be used.

[16]:
```python
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline

# The custom NumPy log transformer
log_transformer = FunctionTransformer(np.log1p)

# The pipeline
estimators = [('log1p', log_transformer), ('minmaxscale', MinMaxScaler())]
pipeline = Pipeline(estimators)

# Convert the original data
data_pipe = pipeline.fit_transform(data_orig)
```

The results are identical. Note that machine learning models and grid searches can also be added to the pipeline (and in fact, usually are.)

[17]:
```python
np.allclose(data_pipe, data)
```

[17]: True

1.6 Part 4

In this section, we will:

- Perform PCA with `n_components` ranging from 1 to 5.
- Store the amount of explained variance for each number of dimensions.
- Also store the feature importance for each number of dimensions. *Hint:* PCA doesn't explicitly provide this after a model is fit, but the `components_` properties can be used to determine something that approximates importance. How you decided to do so is entirely up to you.
- Plot the explained variance and feature importances.

[18]:
```python
from sklearn.decomposition import PCA

pca_list = list()
feature_weight_list = list()

# Fit a range of PCA models

for n in range(1, 6):

    # Create and fit the model
    PCAmod = PCA(n_components=n)
    PCAmod.fit(data)

    # Store the model and variance
    pca_list.append(pd.Series({'n':n, 'model':PCAmod,
                               'var': PCAmod.explained_variance_ratio_.sum()}))
```

```
    # Calculate and store feature importances
    abs_feature_values = np.abs(PCAmod.components_).sum(axis=0)
    feature_weight_list.append(pd.DataFrame({'n':n,
                                              'features': data.columns,
                                              'values':abs_feature_values/
 ↪abs_feature_values.sum()}))

pca_df = pd.concat(pca_list, axis=1).T.set_index('n')
pca_df
```

[18]:

n	model	var
1	PCA(copy=True, iterated_power='auto', n_compon…	0.448011
2	PCA(copy=True, iterated_power='auto', n_compon…	0.72099
3	PCA(copy=True, iterated_power='auto', n_compon…	0.827534
4	PCA(copy=True, iterated_power='auto', n_compon…	0.923045
5	PCA(copy=True, iterated_power='auto', n_compon…	0.979574

Create a table of feature importances for each data column.

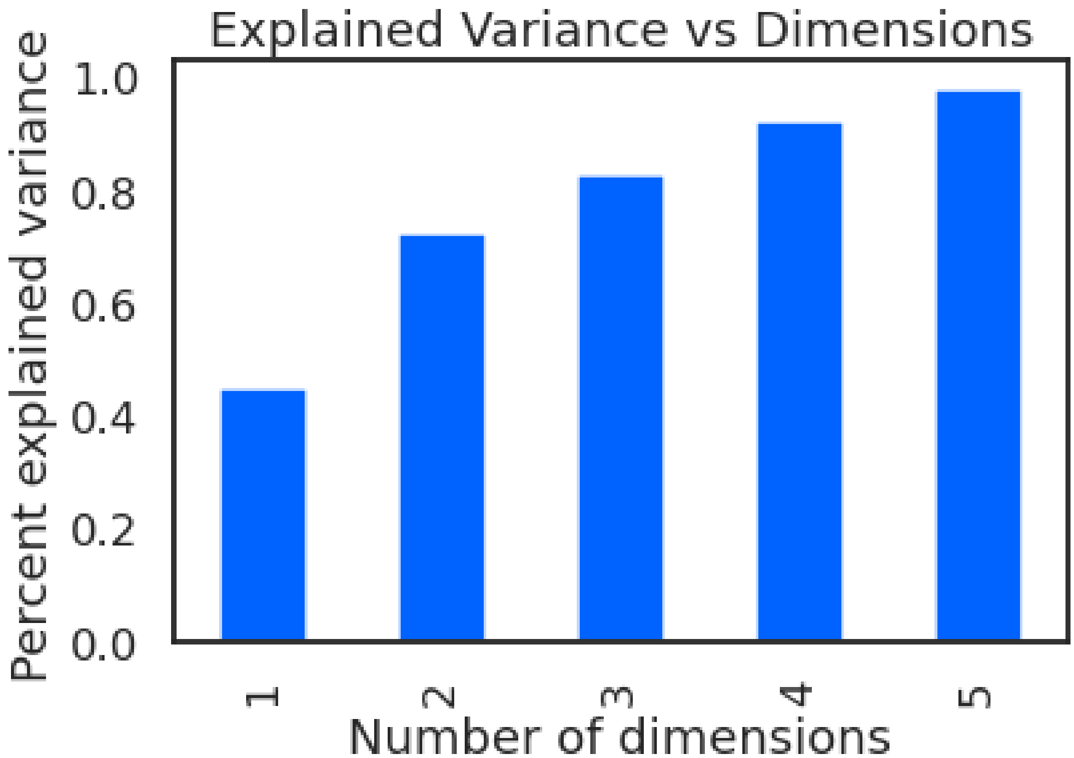
[19]:
```
features_df = (pd.concat(feature_weight_list)
               .pivot(index='n', columns='features', values='values'))

features_df
```

[19]:

features n	Delicassen	Detergents_Paper	Fresh	Frozen	Grocery	Milk
1	0.071668	0.335487	0.060620	0.095979	0.190236	0.246010
2	0.151237	0.177519	0.158168	0.222172	0.112032	0.178872
3	0.165518	0.145815	0.211434	0.268363	0.084903	0.123967
4	0.224259	0.149981	0.239527	0.214275	0.070971	0.100987
5	0.211840	0.182447	0.196382	0.178104	0.067338	0.163888

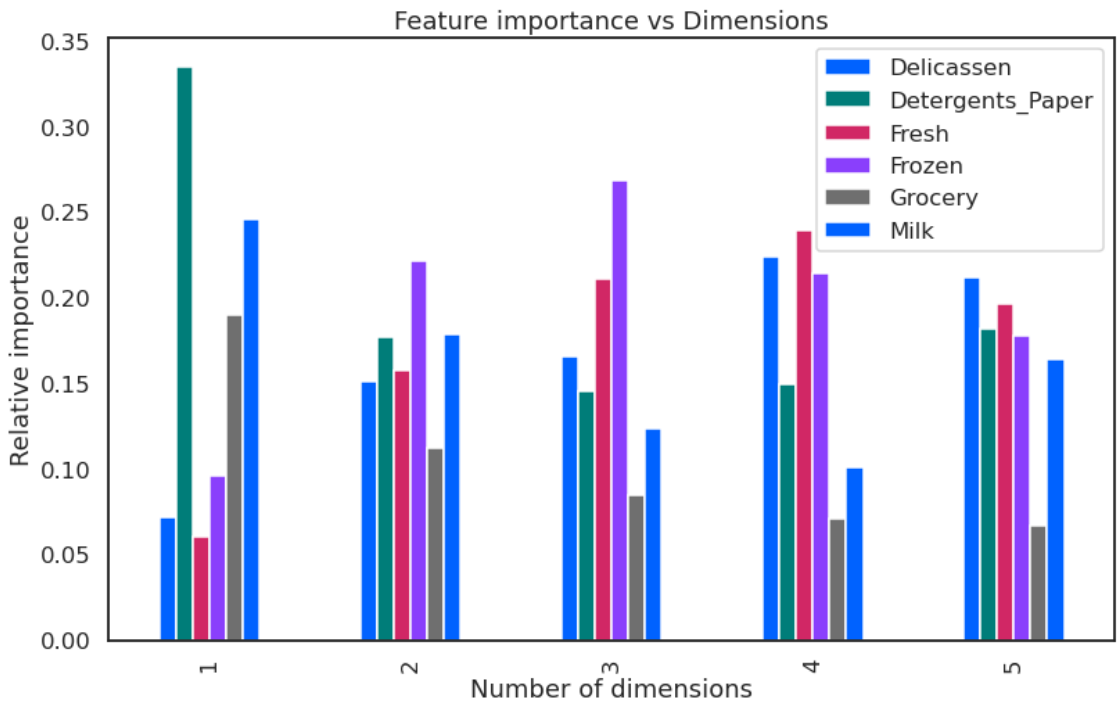
Create a plot of explained variances.

[20]:
```
sns.set_context('talk')
ax = pca_df['var'].plot(kind='bar')

ax.set(xlabel='Number of dimensions',
       ylabel='Percent explained variance',
       title='Explained Variance vs Dimensions');
```



And here's a plot of feature importances.

[21]:
```
ax = features_df.plot(kind='bar', figsize=(13,8))
ax.legend(loc='upper right')
ax.set(xlabel='Number of dimensions',
       ylabel='Relative importance',
       title='Feature importance vs Dimensions');
```



1.7 Part 5

In this section, we will:

- Fit a `KernelPCA` model with `kernel='rbf'`. You can choose how many components and what values to use for the other parameters (`rbf` refers to a Radial Basis Function kernel, and the `gamma` parameter governs scaling of this kernel and typically ranges between 0 and 1). Several other kernels can be tried, and even passed ss cross validation parameters (see this example).
- If you want to tinker some more, use `GridSearchCV` to tune the parameters of the `KernelPCA` model.

The second step is tricky since grid searches are generally used for supervised machine learning methods and rely on scoring metrics, such as accuracy, to determine the best model. However, a custom scoring function can be written for `GridSearchCV`, where larger is better for the outcome of the scoring function.

What would such a metric involve for PCA? What about percent of explained variance? Or perhaps the negative mean squared error on the data once it has been transformed and then inversely transformed?

[22]:
```
from sklearn.decomposition import KernelPCA
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Custom scorer--use negative rmse of inverse transform
def scorer(pcamodel, X, y=None):
```

```python
    try:
        X_val = X.values
    except:
        X_val = X

    # Calculate and inverse transform the data
    data_inv = pcamodel.fit(X_val).transform(X_val)
    data_inv = pcamodel.inverse_transform(data_inv)

    # The error calculation
    mse = mean_squared_error(data_inv.ravel(), X_val.ravel())

    # Larger values are better for scorers, so take negative value
    return -1.0 * mse

# The grid search parameters
param_grid = {'gamma':[0.001, 0.01, 0.05, 0.1, 0.5, 1.0],
              'n_components': [2, 3, 4]}

# The grid search
kernelPCA = GridSearchCV(KernelPCA(kernel='rbf', fit_inverse_transform=True),
                         param_grid=param_grid,
                         scoring=scorer,
                         n_jobs=-1)


kernelPCA = kernelPCA.fit(data)

kernelPCA.best_estimator_
```

[22]:
```
KernelPCA(alpha=1.0, coef0=1, copy_X=True, degree=3, eigen_solver='auto',
     fit_inverse_transform=True, gamma=0.5, kernel='rbf',
     kernel_params=None, max_iter=None, n_components=4, n_jobs=None,
     random_state=None, remove_zero_eig=False, tol=0)
```

1.8 Part 6

Let's explore how our model accuracy may change if we include a `PCA` in our model building pipeline. Let's plan to use sklearn's `Pipeline` class and create a pipeline that has the following steps:

A scaler

`PCA(n_components=n)`

`LogisticRegression`

- Load the Human Activity data from the datasets.

- Write a function that takes in a value of `n` and makes the above pipeline, then predicts the "Activity" column over a 5-fold StratifiedShuffleSplit, and returns the average test accuracy
- For various values of n, call the above function and store the average accuracies.
- Plot the average accuracy by number of dimensions.

[23]:
```python
data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBM-ML0187EN-SkillsNetwork/
↪Human_Activity_Recognition_Using_Smartphones_Data.csv', sep=',')
```

[24]:
```python
data.columns
```

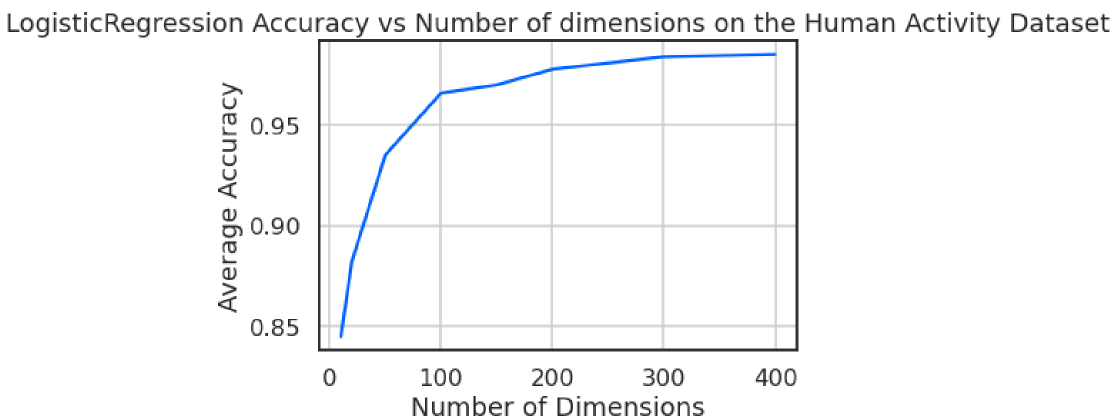
[24]:
```
Index(['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',
       'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X',
       …
       'fBodyBodyGyroJerkMag-skewness()', 'fBodyBodyGyroJerkMag-kurtosis()',
       'angle(tBodyAccMean,gravity)', 'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity'],
      dtype='object', length=562)
```

[25]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = data.drop('Activity', axis=1)
y = data.Activity
sss = StratifiedShuffleSplit(n_splits=5, random_state=42)

def get_avg_score(n):
    pipe = [
        ('scaler', StandardScaler()),
        ('pca', PCA(n_components=n)),
        ('estimator', LogisticRegression(solver='liblinear'))
    ]
    pipe = Pipeline(pipe)
    scores = []
    for train_index, test_index in sss.split(X, y):
        X_train, X_test = X.loc[train_index], X.loc[test_index]
        y_train, y_test = y.loc[train_index], y.loc[test_index]
        pipe.fit(X_train, y_train)
        scores.append(accuracy_score(y_test, pipe.predict(X_test)))
    return np.mean(scores)
```

```
ns = [10, 20, 50, 100, 150, 200, 300, 400]
score_list = [get_avg_score(n) for n in ns]
```

[26]:

```
sns.set_context('talk')

ax = plt.axes()
ax.plot(ns, score_list)
ax.set(xlabel='Number of Dimensions',
       ylabel='Average Accuracy',
       title='LogisticRegression Accuracy vs Number of dimensions on the Human␣
 ↪Activity Dataset')
ax.grid(True)
```



1.8.1 Machine Learning Foundation (C) 2020 IBM Corporation