1 Executive Summary

“Curse of dimensionality” can dramatically impacts the performance of machine learning algorithms on data with many input features. When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the “essence” of the data. This is called dimensionality reduction.

We have some techniques for dimensionality reduction:

1) **Linear Algebra Methods (PCA, FA, SVD),**
Matrix factorization methods drawn from the field of linear algebra can be used for dimensionality
2) **Non-linear transformation methods or Manifold Learning Methods (Isomap Embedding, LLE, t-SNE),**
Manifold learning methods seek a lower-dimensional projection of high dimensional input that captures the salient properties of the input data.

The need for dimensionality reduction:
As the number of features increases, the model becomes more complex. The more the number of features, the more the chances of overfitting. Therefore, the primary aim of dimensionality reduction is to avoid overfitting.

2 Introduction

In this notebook, I will review how to use popular dimensionality reduction algorithms from the scikit-learn library. I plan to build a logistic regression model in python to estimate the attrition of employees. The dataset contains 45 features (after encoding) which can lead to making an overfitted model. I also compare my findings with dimensionality reduction through autoencoders.

3 Loading and Exploring Data

3.1 Loading libraries

In [245]:
```
1 #Loading libraries↔
```

3.2 Loading Data

In [498]:
```
1 data = pd.read_csv('Desktop/HR-Employee-Attrition-All.csv')
```

This is a data set to uncover the factors that lead to employees' attrition. The data has already been processed. It is checked for the existance of outliers and missing values. We only need to perform feature scaling.

3.3 Data Size and Structure

In [499]:
```
1 data.shape
```

Out[499]: (1470, 35)

Our dataframe consists of 10 predictors and our response variable is "AverageTemperature".

In [500]:
```
# let us look at some records of our data
data.head()
```

Out[500]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCo
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	

5 rows × 35 columns

In [501]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [502]:
```
# I will remove two unneccesary atributes from this data
data.drop(columns=['EmployeeCount','EmployeeNumber'],inplace=True)
```

In [503]:
```
data.isna().sum().sum()
```

Out[503]: 0

No missing values

In [504]:
```
# let us keep a copy of our dataset
dataset = data.copy()
```

3.4 Feature Scaling

In [505]:
```
numerical_feature = list(dataset.select_dtypes(include=['int64']).columns)
for feature in numerical_feature:
    #numpy.newaxis represents a new axis in numpy array
    fetched_val = dataset[feature].values[:, np.newaxis]
    scaler = MinMaxScaler().fit(fetched_val)
    dataset[feature] = scaler.transform(fetched_val)
```

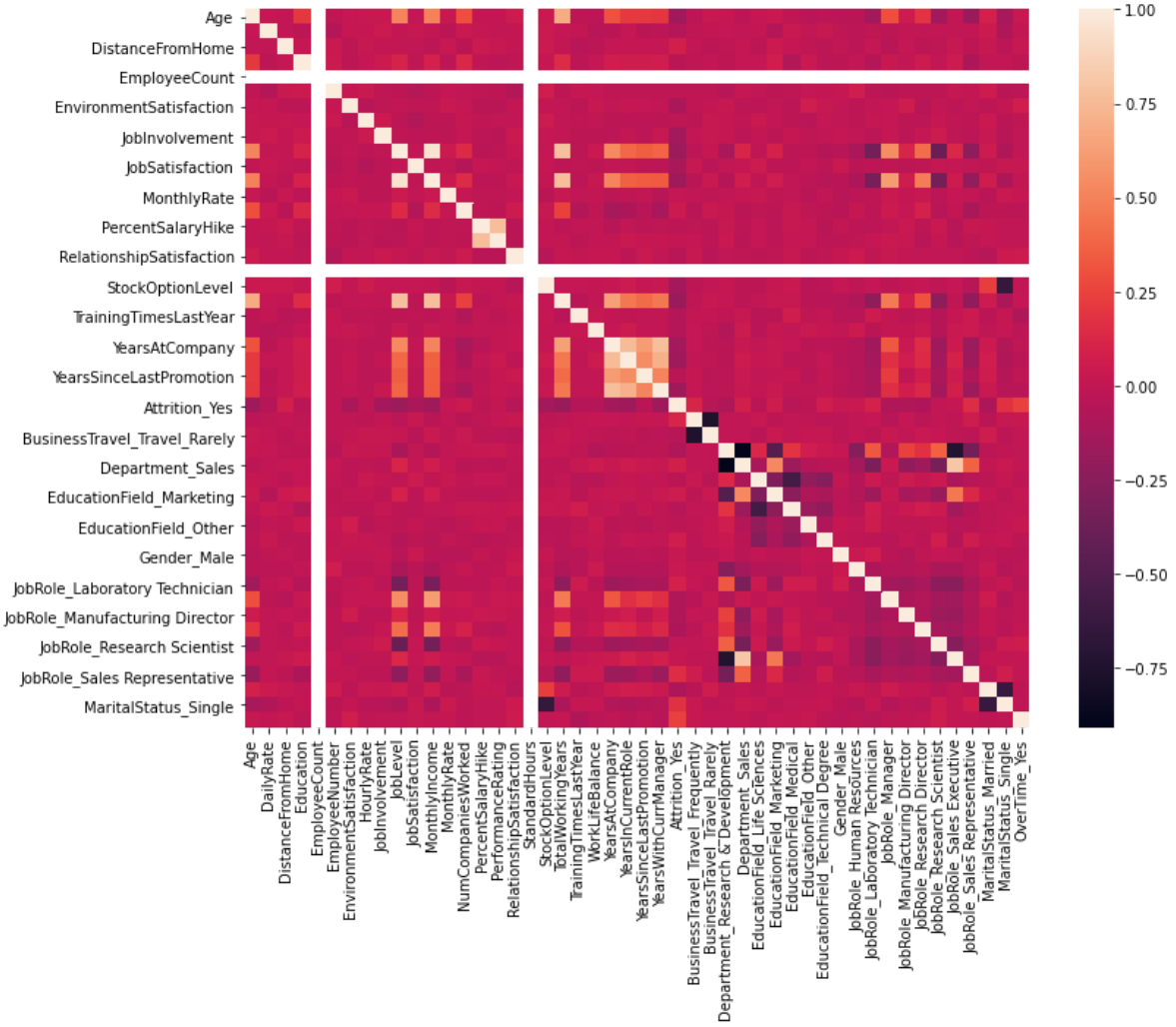
In [506]:
```
mylist = list(dataset.select_dtypes(include=['object']).columns)
```

In [507]:
```
#mylist = list(dataset.select_dtypes(include=['object']).columns)
dummies = pd.get_dummies(dataset[mylist], prefix= mylist, drop_first=True)
data_trans = pd.concat([dataset,dummies], axis =1 )
data_transans_copy= data_trans.copy(deep=True )
```

In [508]:
```
y_trans=data_trans.Attrition
data_trans.drop(mylist, axis=1, inplace = True)
```

In [491]:
```
1 fig, ax = plt.subplots(figsize=(12,9))
2 sns.heatmap(data_trans.corr(), ax=ax)
```

Out[491]: <AxesSubplot:>



In [511]:
```
1 print(data_trans.corr()["Attrition_Yes"])
```

```
Age                                -0.159205
DailyRate                          -0.056652
DistanceFromHome                    0.077924
Education                          -0.031373
EnvironmentSatisfaction            -0.103369
HourlyRate                         -0.006846
JobInvolvement                     -0.130016
JobLevel                           -0.169105
JobSatisfaction                    -0.103481
MonthlyIncome                      -0.159840
MonthlyRate                         0.015170
NumCompaniesWorked                  0.043494
PercentSalaryHike                  -0.013478
PerformanceRating                   0.002889
RelationshipSatisfaction           -0.045872
StandardHours                            NaN
StockOptionLevel                   -0.137145
TotalWorkingYears                  -0.171063
TrainingTimesLastYear              -0.059478
WorkLifeBalance                    -0.063939
YearsAtCompany                     -0.134392
YearsInCurrentRole                 -0.160545
YearsSinceLastPromotion            -0.033019
YearsWithCurrManager               -0.156199
Attrition_Yes                       1.000000
BusinessTravel_Travel_Frequently    0.115143
BusinessTravel_Travel_Rarely       -0.049538
Department_Research & Development  -0.085293
Department_Sales                    0.080855
EducationField_Life Sciences       -0.032703
EducationField_Marketing            0.055781
EducationField_Medical             -0.046999
EducationField_Other               -0.017898
EducationField_Technical Degree     0.069355
Gender_Male                         0.029453
JobRole_Human Resources             0.036215
JobRole_Laboratory Technician       0.098290
JobRole_Manager                    -0.083316
JobRole_Manufacturing Director     -0.082994
JobRole_Research Director          -0.088870
JobRole_Research Scientist         -0.000360
JobRole_Sales Executive             0.019774
JobRole_Sales Representative        0.157234
MaritalStatus_Married              -0.090984
MaritalStatus_Single                0.175419
OverTime_Yes                        0.246118
Name: Attrition_Yes, dtype: float64
```

From the above coefficient of each feature in the dataset with target class label “Attrition_Yes”, it is confirmed that features “OverTime_Yes”, “JobRole_Sales Representative”, "Age" and “MaritalStatus_Single” are top 4 features having maximum modular value.

Note: The result may vary if I choose different method in for the correlation method.

In [512]:
```
Feature_Name = ['Age', 'DailyRate', 'DistanceFromHome', 'Education',
    'EnvironmentSatisfaction', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome',
       'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike',
       'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
       'Department_Research & Development', 'Department_Sales',
       'EducationField_Life Sciences', 'EducationField_Marketing',
       'EducationField_Medical', 'EducationField_Other',
       'EducationField_Technical Degree', 'Gender_Male',
       'JobRole_Human Resources', 'JobRole_Laboratory Technician',
       'JobRole_Manager', 'JobRole_Manufacturing Director',
       'JobRole_Research Director', 'JobRole_Research Scientist',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_Yes']
```

In [687]:
```
Xdfcopy= data_trans.loc[:, data_trans.columns != 'Attrition_Yes']
y=np.array(data_trans.Attrition_Yes)
X=np.array(data_trans.loc[:, data_trans.columns != 'Attrition_Yes'])
```

In [514]:
```
X.shape
```

Out[514]: (1470, 45)

4 Building the Model

In [515]:
```
from sklearn.model_selection import cross_val_score

# StratifiedKFold ensure that relative class frequencies is approximately preserved
# in each train and validation fold.
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
```

In [516]:
```
from sklearn import metrics
Model = LogisticRegression()
```

Here I split the data into 10 groups, and use each of them in turn to evaluate the model fit on the other 9/10 of the data. This would be rather tedious to do by hand, and so we can use Scikit-Learn's cross_val_score.

In [517]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=101)
n_scores = cross_val_score(Model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.884 (0.017)
```

This is the accuracy without using any dimensionality resuction techniques. Now I will capture the tradeoff between this accuracy and the reduction in the # of features.

5 Pipelines and Composite Estimators

The pipeline usually consists of a series of transforms and a final estimator. The Pipeline is built using a list of (key, value) pairs, where the **"key"** is a string containing the name I want to give to each *step* and **"value"** is an *estimator object*.

5.1 Principal Composant Analysis (PCA, IPCA)

Principal component analysis is a statistical method that uses the process of linear, orthogonal transformation to transform a higher-dimensional set of features that could be possibly correlated into a lower-dimensional set of linearly uncorrelated features. These transformed and newly created features are also known as Principal Components or PCs. In any PCA transformation, the total number of PCs is always less than or equal to the initial number of features. The first principal

component tries to capture the maximum variance of the original set of features.The PCA starts by extracting maximum variance and puts them into the first factor. Then it starts with the second factor after removing the first variance and the analysis goes on until the last factor.

The incremental principal component analysis is a variant of the PCA, It only keeps the most significant singular vectors to project the data into a space to reduced size.

In [534]:
```
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA
from sklearn.model_selection import GridSearchCV

pca = PCA()
ipca = IncrementalPCA(batch_size=100)
logistic = LogisticRegression()

# define the pipeline
steps = [('reduce_dim', pca), (('logistic', logistic))]
Model_pip = Pipeline(steps=steps)

ipcasteps = [('reduce_dim', ipca), (('logistic', logistic))]
ipca_Model_pip = Pipeline(steps=ipcasteps)
```

In [535]:
```
# I want to evaluate pca with a logistic regression model for the sake of classification
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=101)
n_scores = cross_val_score(Model_pip, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
ipca_n_scores = cross_val_score(ipca_Model_pip, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

# report performance
print('Accuracy of PCA: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
print('Accuracy od IPCA: %.3f (%.3f)' % (mean(ipca_n_scores), std(ipca_n_scores)))
```

```
Accuracy of PCA: 0.885 (0.017)
Accuracy od IPCA: 0.885 (0.017)
```

I did not observe a significant difference. The reducton in the accuracy also is negligible. Now let us define our hyperparameter ranges using any of the instantiated pipeline's param keys.

In [537]:
```
param_grid = {
    'reduce_dim__n_components': [5, 10, 15, 20, 22, 25, 27, 30, 32, 35, 40, 45, 50]
}
search = GridSearchCV(Model_pip, param_grid, n_jobs=-1)
ipca_search = GridSearchCV(ipca_Model_pip, param_grid, n_jobs=-1)
search.fit(X, y)
ipca_search.fit(X, y)

print("Best parameter PCA(CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
print("Best parameter IPCA(CV score=%0.3f):" % ipca_search.best_score_)
print(ipca_search.best_params_)

# Then i will fit the PCA and IPCA instances on our data.
pca.fit(X)
ipca.fit(X)
```

```
Best parameter PCA(CV score=0.888):
{'reduce_dim__n_components': 35}
Best parameter IPCA(CV score=0.888):
{'reduce_dim__n_components': 35}
```

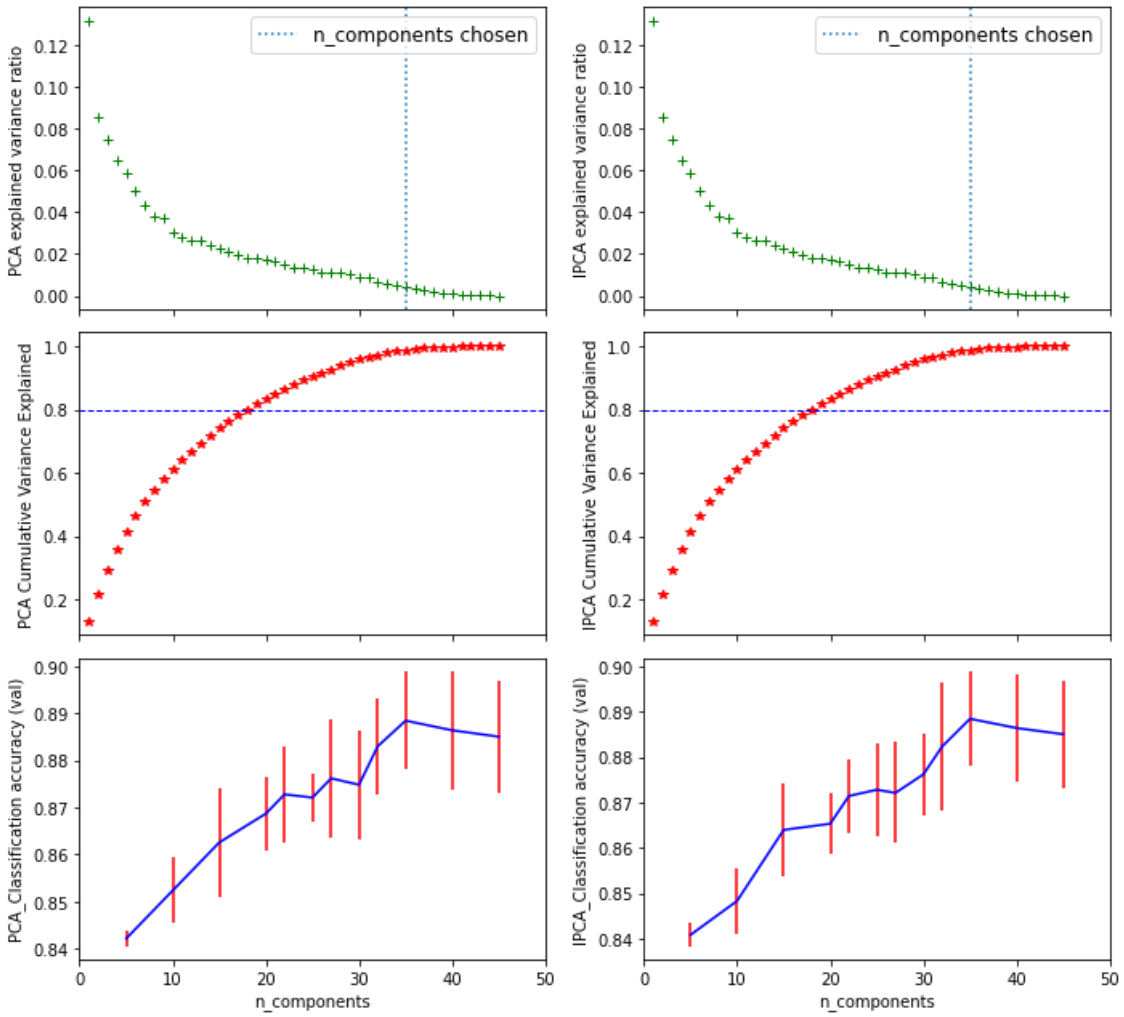
Out[537]:
```
IncrementalPCA(batch_size=100)
```

The ideal number of components is identified to be 35 but I need to find out for which accuracy and also aligh it with the business requirements.

In [521]:
```
# I can access the parameter keys of the individual as follow:
search.get_params().keys()
```

Out[521]:
```
dict_keys(['cv', 'error_score', 'estimator__memory', 'estimator__steps', 'estimator__verbose',
'estimator__reduce_dim', 'estimator__logistic', 'estimator__reduce_dim__copy', 'estimator__reduc
e_dim__iterated_power', 'estimator__reduce_dim__n_components', 'estimator__reduce_dim__random_st
ate', 'estimator__reduce_dim__svd_solver', 'estimator__reduce_dim__tol', 'estimator__reduce_dim_
_whiten', 'estimator__logistic__C', 'estimator__logistic__class_weight', 'estimator__logistic__d
ual', 'estimator__logistic__fit_intercept', 'estimator__logistic__intercept_scaling', 'estimator
__logistic__l1_ratio', 'estimator__logistic__max_iter', 'estimator__logistic__multi_class', 'est
imator__logistic__n_jobs', 'estimator__logistic__penalty', 'estimator__logistic__random_state',
'estimator__logistic__solver', 'estimator__logistic__tol', 'estimator__logistic__verbose', 'esti
mator__logistic__warm_start', 'estimator', 'iid', 'n_jobs', 'param_grid', 'pre_dispatch', 'refi
t', 'return_train_score', 'scoring', 'verbose'])
```

In [541]:
```
#Let us plot the results↔
```



Assume, as a data scientist, my requirement is that 80% of the variance must be explained in the PCA transformed data. This level of cumulative variance is explained by including almost 18 principal components. Therefore, there is a tradeoff here among the desired accuracy, number of identified components and business requirements.

In [523]:
```
# This part shows the correlation between the identified components and features of the ori
comp=pd.DataFrame(pca.fit(X).components_, columns=Feature_Name)
#print(comp)
```

Note:The highest correlation (in absolute terms) between the identified principal component and the original dataset features determines that the feature with a large value for 'pc1' in the transformed dataset will have a high value for that specific feature in the original data as well.

5.2 Factor Analysis (FA)

It is a technique which is used to reduce a large number of variables into fewer numbers of factors. The values of observed data are expressed as functions of a number of possible causes in order to find which are the most important. The observations are assumed to be caused by a linear transformation of lower dimensional latent factors and added Gaussian noise.
Factor analysis explains correlations among multiple outcomes as a result of one or more factors. As it attempts to represent a set of variables by a smaller number, it involves data reduction. It explores unexplained factors that represent underlying concepts that cannot be adequately measured by a single variable. It is most popular nowadays in survey research where the responses to each question represent an outcome. It is because multiple questions are often related and the underlying factor may influence the subject responses. The reduction technique of factor analysis in reducing a large number of variables into a fewer number of factors enables to extract maximum common variance. The common score from all variables as an index can be used for further analysis.
It assumes several assumptions including:
· There exists a linear relationship.
· There is no multicollinearity.
· Includes relevant variables into the analysis.
· No true correlation between variables and factors.

In [698]:
```
from sklearn.decomposition import FactorAnalysis
```

In [700]:
```
fa = FactorAnalysis()
logistic = LogisticRegression()
steps = [('reduce_dim', fa), (('logistic', logistic))]
Model_pip = Pipeline(steps=steps)
```

In [701]:
```
# evaluate pca with logistic regression algorithm for classification
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3 #, random_state=101
                             )
# Data shgould be in the form of a dataframe not array
n_scores = cross_val_score(Model_pip, Xdfcopy, data_trans['Attrition_Yes'], scoring='accura

# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.863 (0.018)
```

In [702]:
```
param_grid = {
    'reduce_dim__n_components': [5, 10, 15, 20, 22, 25, 27, 30, 32, 35, 40, 45, 50]
}
search = GridSearchCV(Model_pip, param_grid, n_jobs=-1)
search.fit(Xdfcopy, data_trans['Attrition_Yes'])

print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)

fa.fit(Xdfcopy)
```

```
Best parameter (CV score=0.872):
{'reduce_dim__n_components': 40}
```

Out[702]:
```
FactorAnalysis()
```

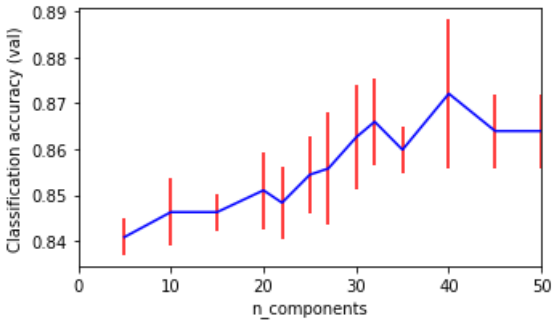
In [703]:
```
fig, ax1 = plt.subplots(nrows=1, sharex=True, figsize=(5, 3))

# For each number of components, find the best classifier results
results = pd.DataFrame(search.cv_results_)
components_col = 'param_reduce_dim__n_components'
best_clfs = results.groupby(components_col).apply(
    lambda g: g.nlargest(1, 'mean_test_score'))

best_clfs.plot(x=components_col, y='mean_test_score', yerr='std_test_score',
               legend=False, ax=ax1, color="blue", ecolor="red")
ax1.set_ylabel('Classification accuracy (val)')
ax1.set_xlabel('n_components')

plt.xlim(0, 50)

plt.tight_layout()
plt.show()
```



5.3 Singular Value Decomposition (SVD)

Singular Value Decomposition, or SVD, is the most popular technique for dimensionality reduction when data is sparse (Recommender Systems, One Hot Encoding, Text Classification). SVD can be applied even on rectangular matrices; whereas, eigenvalues are defined only for square matrices. The equivalent of eigenvalues obtained through the SVD method are called singular values, and vectors obtained equivalent to eigenvectors are known as singular vectors. However, as they are rectangular in nature, we need to have left singular vectors and right singular vectors respectively for their dimensions.

In [560]:
```
from sklearn.decomposition import TruncatedSVD
tsvd = TruncatedSVD()
Model_logistic = LogisticRegression()
steps = [('reduce_dim', tsvd), (('Model', Model_logistic))]
Model_pip = Pipeline(steps=steps)
```

In [561]:
```
param_grid = {
    'reduce_dim__n_components': [5, 10, 15, 20, 22, 25, 27, 30, 32, 35, 40, 45, 50]
}
search = GridSearchCV(Model_pip, param_grid, n_jobs=-1)
search.fit(X, y)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
tsvd.fit(X)
```

```
Best parameter (CV score=0.888):
{'reduce_dim__n_components': 35}
```

Out[561]:
```
TruncatedSVD()
```

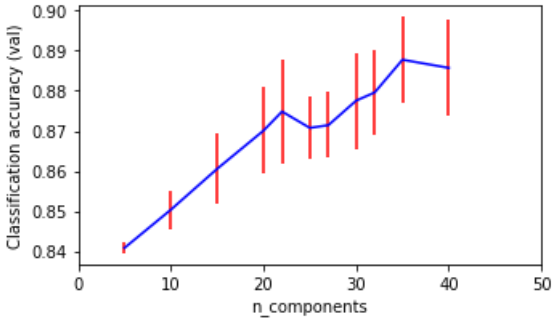
In [562]:
```
fig, ax1= plt.subplots(nrows=1, sharex=True, figsize=(5, 3))

# For each number of components, find the best classifier results
results = pd.DataFrame(search.cv_results_)
components_col = 'param_reduce_dim__n_components'
best_clfs = results.groupby(components_col).apply(
    lambda g: g.nlargest(1, 'mean_test_score'))

best_clfs.plot(x=components_col, y='mean_test_score', yerr='std_test_score',
               legend=False, ax=ax1, color="blue", ecolor="red")
ax1.set_ylabel('Classification accuracy (val)')
ax1.set_xlabel('n_components')

plt.xlim(0, 50)

plt.tight_layout()
plt.show()
```



5.4 Linear Discriminant Analysis(LDA)

LDA projects data in a way that the class separability is maximised. Examples from same class are put closely together by the projection. Examples from different classes are placed far apart by the projection
LDA is limited in the number of components used in the dimensionality reduction to between the number of classes minus one, in this case we are dealing witha binary classification problem so the #of components = 1

In [707]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda= LinearDiscriminantAnalysis()
Model_logistic = LogisticRegression()
steps = [('reduce_dim', lda), (('Model', Model_logistic))]
Model_pip = Pipeline(steps=steps)
```

In [709]:
```
param_grid = {
    #n_components cannot be larger than min(n_features, n_classes - 1)
    'reduce_dim__n_components': [1]
}
search = GridSearchCV(Model_pip, param_grid, n_jobs=-1)
search.fit(Xdfcopy, data_trans['Attrition_Yes'])
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
lda.fit(Xdfcopy, data_trans['Attrition_Yes'])
```

```
Best parameter (CV score=0.882):
{'reduce_dim__n_components': 1}
```

Out[709]:
```
LinearDiscriminantAnalysis()
```

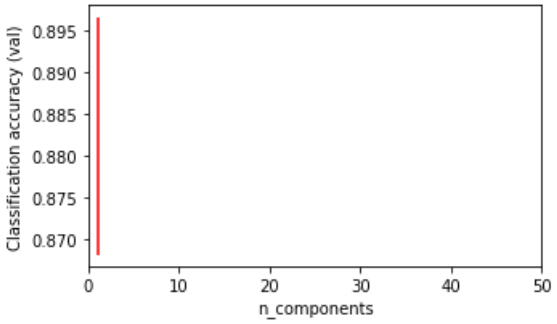
In [710]:
```
fig, ax1 = plt.subplots(nrows=1, sharex=True, figsize=(5, 3))

# For each number of components, find the best classifier results
results = pd.DataFrame(search.cv_results_)
components_col = 'param_reduce_dim__n_components'
best_clfs = results.groupby(components_col).apply(↔

best_clfs.plot(x=components_col, y='mean_test_score', yerr='std_test_score',
               legend=False, ax=ax1, color="blue", ecolor="red")
ax1.set_ylabel('Classification accuracy (val)')
ax1.set_xlabel('n_components')

plt.xlim(0, 50)

plt.tight_layout()
plt.show()
```



5.5 Isomap Embedding

Isomap should be used when there is a non-linear mapping between our higher-dimensional data and our lower-dimensional manifold.
Euclidean distances should not be used for approximating the distance between two points in non-linear manifolds while geodesic distances can be used. Isomap uses the above principle to create a similarity matrix for eigenvalue decomposition. Unlike other non-linear dimensionality reduction like LLE which only use local information, isomap uses the local information to create a global similarity matrix. The isomap algorithm uses euclidean metrics to prepare the neighborhood graph. Then, it approximates the geodesic distance between two points by measuring shortest path between these points using graph distance. Thus, it approximates both global as well as the local structure of the dataset in the low dimensional embedding.

Isomap is better than linear methods when dealing with almost all types of real image and motion tracking.

In [567]:
```
from sklearn.manifold import Isomap
iso= Isomap()
Model_logistic = LogisticRegression()
steps = [('reduce_dim', iso), (('Model', Model_logistic))]
Model_pip = Pipeline(steps=steps)
```

In [568]:
```
param_grid = {
    'reduce_dim__n_components': [5, 10, 15, 20, 22, 25, 27, 30, 32, 35, 40, 45, 50]
}
search = GridSearchCV(Model_pip, param_grid, n_jobs=-1)
search.fit(X, y)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
iso.fit(X)
```

```
Best parameter (CV score=0.844):
{'reduce_dim__n_components': 15}
```

Out[568]:
```
Isomap()
```

In [569]:
```
iso.get_params().keys()
```

Out[569]:
```
dict_keys(['eigen_solver', 'max_iter', 'metric', 'metric_params', 'n_components', 'n_jobs', 'n_n
eighbors', 'neighbors_algorithm', 'p', 'path_method', 'tol'])
```

In [570]:

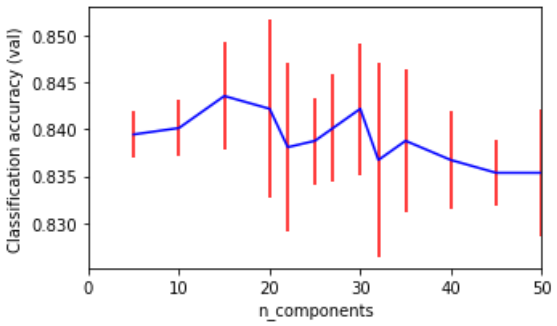
```
fig, ax1 = plt.subplots(nrows=1, sharex=True, figsize=(5, 3))

# For each number of components, find the best classifier results
results = pd.DataFrame(search.cv_results_)
components_col = 'param_reduce_dim__n_components'
best_clfs = results.groupby(components_col).apply(
    lambda g: g.nlargest(1, 'mean_test_score'))

best_clfs.plot(x=components_col, y='mean_test_score', yerr='std_test_score',
               legend=False, ax=ax1, color="blue", ecolor="red")
ax1.set_ylabel('Classification accuracy (val)')
ax1.set_xlabel('n_components')

plt.xlim(0, 50)

plt.tight_layout()
plt.show()
```



5.6 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a new technique for dimension reduction and data visualization. t-SNE not only captures the local structure of the higher dimension but also preserves the global structures of the data like clusters. It has stunning ability to produce well-defined segregated clusters. t-SNE is based on stochastic neighbor embedding(SNE).

The approach of SNE is:

1)Construct a probability distribution to represent the dataset, where similar points have a higher probability of being picked, and dissimilar points have a lower probability of being picked.
2)Create a low dimensional space that replicates the properties of the probability distribution from Step 1 as close as possible.

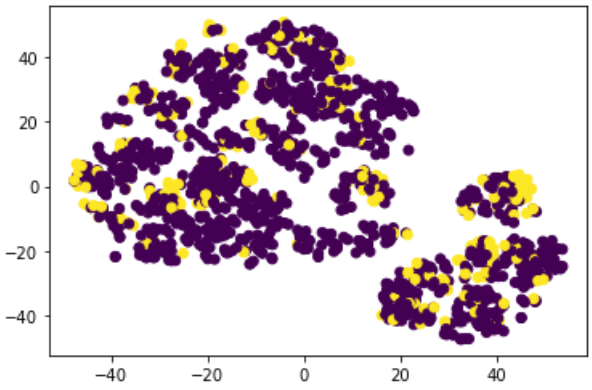
In [579]:

```
from sklearn.manifold import TSNE

#n_component:s (default: 2), perplexity : (default: 30)
tsne = TSNE(random_state=101)
tsne_results = tsne.fit_transform(X)

tsne_results=pd.DataFrame(tsne_results, columns=['comp1', 'comp2'])

# for the sake of visualization, the color assigned as the label value (2 values for the em
plt.scatter(tsne_results['comp1'], tsne_results['comp2'], c=y)
plt.show()
```



I visualized the identified dimensions in a 2D space. but 3d space also was possible if I initialize the TSNE with 3 components. It is obvious that the data is not seperable with only two dimensions.

Perplexity is a parameter for the number of nearest neighbors based on which t-SNE will determine the potential neighbors. Generally, the larger the dataset the larger should be the perplexity value. Let us do some sensitivity analysis on the "Perplexity" parameter.

In [580]:

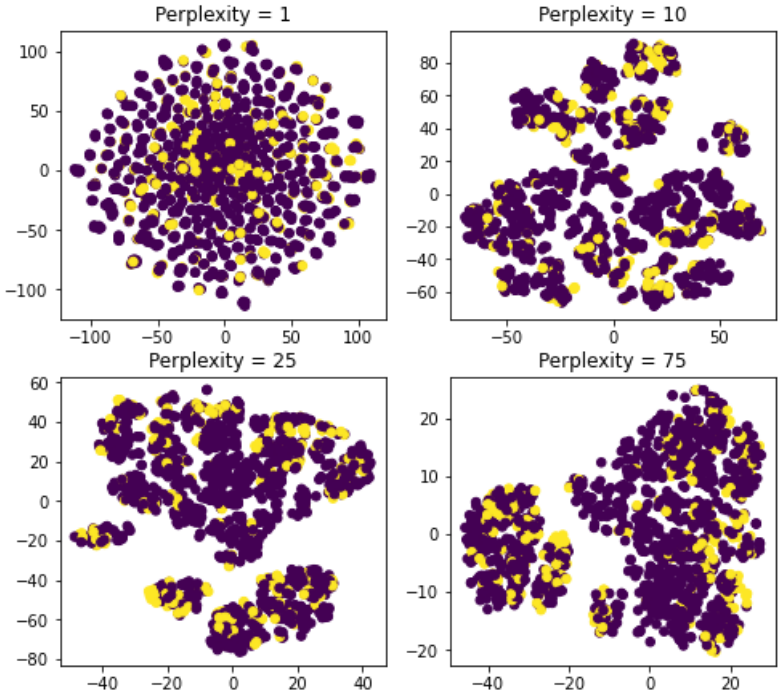
```
plt.figure(figsize = (8,4))
plt.subplots_adjust(top = 1.5)

for index, p in enumerate([1, 10, 25, 75]):

    tsne = TSNE(n_components = 2, perplexity = p, random_state=101)
    tsne_results = tsne.fit_transform(X)

    tsne_results=pd.DataFrame(tsne_results, columns=['comp1', 'comp2'])

    plt.subplot(2,2,index+1)
    plt.scatter(tsne_results['comp1'], tsne_results['comp2'], c=y, s=30)
    plt.title('Perplexity = '+ str(p))
plt.show()
```



As we can see, with smaller values of perplexitity (Perplexity = 1) the local variations dominate and with larger values (Perplexity = 75) global variations dominate. The results seem to be more or less similar with perplexity values in between.

Note:Today two interesting practical applications of autoencoders are data denoising, and dimensionality reduction for data visualization. **PCA** creates embedding by finding the axis that maximizes the variance. The result of a fitted PCA are the k components attribute in sklearn). We can then transform our test data by applying the linear combination to the test points.

TSNE TSNE on the other hand creates low dimension embedding that tries to respect the distance between the points in the real dimensions. TSNE doesn't look at points given their position in the high dimension space it just looks at the distance between that point and its neighbors. Furthermore, we cannot transform data using TSNE, TSNE transform function needs to fit the entire data first. In this case buliding a machine learning model would be useless since we leaked our test data.

5.7 Auto-encoders

Another popular dimensionality reduction method are auto-encoders, which is a type of artificial neural network. They compress the input into a latent-space representation, and then reconstructs the output from this representation.

An autoencoder is composed of two parts :
Encoder: compresses the input into a latent-space representation.
Decoder: reconstruct the input from the latent space representation.
In almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks.
With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

In [615]:
```
from keras.layers import Input, Dense
from keras.models import Model
```

In [650]:
```
# Number of inputs = Number of Features
ncol = X.shape[1]
```

In [651]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, train_size = 0.7, random_state =
```

In [652]:
```
input_dim = Input(shape = (ncol, ))
encoding_dim = 10
# Encoder Layers
encoded1 = Dense(40, activation = 'relu')(input_dim)
encoded2 = Dense(35, activation = 'relu')(encoded1)
encoded3 = Dense(30, activation = 'relu')(encoded2)
encoded4 = Dense(25, activation = 'relu')(encoded3)
encoded5 = Dense(20, activation = 'relu')(encoded4)
encoded6 = Dense(15, activation = 'relu')(encoded5)
encoded7 = Dense(encoding_dim, activation = 'relu')(encoded6)

# Decoder Layers
decoded1 = Dense(15, activation = 'relu')(encoded7)
decoded2 = Dense(20, activation = 'relu')(decoded1)
decoded3 = Dense(25, activation = 'relu')(decoded2)
decoded4 = Dense(30, activation = 'relu')(decoded3)
decoded5 = Dense(35, activation = 'relu')(decoded4)
decoded6 = Dense(40, activation = 'relu')(decoded5)
decoded7 = Dense(ncol, activation = 'sigmoid')(decoded6)

# Combine Encoder and Deocder Layers
autoencoder = Model(inputs = input_dim, outputs = decoded7)

# Compile the Model
autoencoder.compile(optimizer = 'adadelta', loss = 'binary_crossentropy')
```

In [653]:
```
1 autoencoder.summary()
```

```
Model: "model_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_16 (InputLayer)        (None, 45)                0
_________________________________________________________________
dense_149 (Dense)            (None, 40)                1840
_________________________________________________________________
dense_150 (Dense)            (None, 35)                1435
_________________________________________________________________
dense_151 (Dense)            (None, 30)                1080
_________________________________________________________________
dense_152 (Dense)            (None, 25)                775
_________________________________________________________________
dense_153 (Dense)            (None, 20)                520
_________________________________________________________________
dense_154 (Dense)            (None, 15)                315
_________________________________________________________________
dense_155 (Dense)            (None, 10)                160
_________________________________________________________________
dense_156 (Dense)            (None, 15)                165
_________________________________________________________________
dense_157 (Dense)            (None, 20)                320
_________________________________________________________________
dense_158 (Dense)            (None, 25)                525
_________________________________________________________________
dense_159 (Dense)            (None, 30)                780
_________________________________________________________________
dense_160 (Dense)            (None, 35)                1085
_________________________________________________________________
dense_161 (Dense)            (None, 40)                1440
_________________________________________________________________
dense_162 (Dense)            (None, 45)                1845
=================================================================
Total params: 12,285
Trainable params: 12,285
Non-trainable params: 0
_________________________________________________________________
```

In [654]:
```
1 autoencoder.fit(X_train, X_train, nb_epoch = 10, batch_size = 32, shuffle = False, validati
```

```
Train on 1029 samples, validate on 441 samples
Epoch 1/10
1029/1029 [==============================] - 2s 2ms/step - loss: 0.6835 - val_loss: 0.6602
Epoch 2/10
1029/1029 [==============================] - 0s 208us/step - loss: 0.5792 - val_loss: 0.5314
Epoch 3/10
1029/1029 [==============================] - 0s 219us/step - loss: 0.5330 - val_loss: 0.5268
Epoch 4/10
1029/1029 [==============================] - 0s 222us/step - loss: 0.5304 - val_loss: 0.5248
Epoch 5/10
1029/1029 [==============================] - 0s 221us/step - loss: 0.5286 - val_loss: 0.5233
Epoch 6/10
1029/1029 [==============================] - 0s 196us/step - loss: 0.5271 - val_loss: 0.5221
Epoch 7/10
1029/1029 [==============================] - 0s 232us/step - loss: 0.5260 - val_loss: 0.5212
Epoch 8/10
1029/1029 [==============================] - 0s 213us/step - loss: 0.5252 - val_loss: 0.5204
Epoch 9/10
1029/1029 [==============================] - 0s 214us/step - loss: 0.5245 - val_loss: 0.5197
Epoch 10/10
1029/1029 [==============================] - 0s 219us/step - loss: 0.5238 - val_loss: 0.5190
```

Out[654]:
```
<keras.callbacks.callbacks.History at 0x2722be102c8>
```

In [655]:
```
1 encoder = Model(inputs = input_dim, outputs = encoded7)
2 encoded_input = Input(shape = (encoding_dim, ))
```

In [656]:
```
encoded_train = pd.DataFrame(encoder.predict(X))
encoded_train = encoded_train.add_prefix('feature_')
```

In [ ]:
```
# if our latent space was two-dimensional, we could visualize them like bellow ↔
```

5.7.1 Building the Model

In [658]:
```
from sklearn.model_selection import cross_val_score

# StratifiedKFold ensure that relative class frequencies is approximately preserved
# in each train and validation fold.
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
```

In [659]:
```
from sklearn import metrics
Model = LogisticRegression()
```

Now I will give the "encoded_train" data for the cross validation.

In [660]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=101)
n_scores = cross_val_score(Model, encoded_train, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.839 (0.003)
```

Well it seems with this method and with the help of 10 components we could maintain the accuracy of 84%.

6 Conclusion

The results of my analysis indicate how different dimensionality reduction techniques can be used to captures the “essence” of the data (prominent features) while maintaining the precitive power of a model. I observed that the choise of the algorithms depends heavily on the application use case and the type of data used. For example, Factor Analysis reduces the observed variables into a few unobserved variables or identifies the groups of inter-related variables, which help the market researchers to compress the market situations and find the hidden relationship among the features of their dataset. Isomap is better than linear methods when dealing with almost all types of real image and motion tracking. t_SNE is suitable for data compression (image, audio), data denoising (signal) etc. Some algorithms may perform better in presence of homoscedastic noise (noise variance is the same for each feature) or heteroscedastic noise (noise variance is the different for each feature). SVD, is usually used when data is sparse (Recommender Systems, One Hot Encoding, Text Classification). There are a lot of issues that should be considered while picking a dimensionality reduction techniques.

7 References

https://medium.com/analytics-vidhya/locally-linear-embedding-lle-data-mining-b956616d24e9 (https://medium.com/analytics-vidhya/locally-linear-embedding-lle-data-mining-b956616d24e9)
https://blog.paperspace.com/dimension-reduction-with-isomap/ (https://blog.paperspace.com/dimension-reduction-with-isomap/)
https://pyshark.com/visualization-of-multidimensional-datasets-using-t-sne-in-python/ (https://pyshark.com/visualization-of-multidimensional-datasets-using-t-sne-in-python/)
https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_fa_model_selection.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-fa-model-selection-py (https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_fa_model_selection.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-fa-model-selection-py)
https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/#:~:text=6%20Dimensionality%20Reduction%20Algorithms%20With%20Python.%20Dimensionality%20reduction,regr (https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/#:~:text=6%20Dimensionality%20Reduction%20Algorithms%20With%20Python.%20Dimensionality%20reduction,regr

In [ ]:

```
1
```