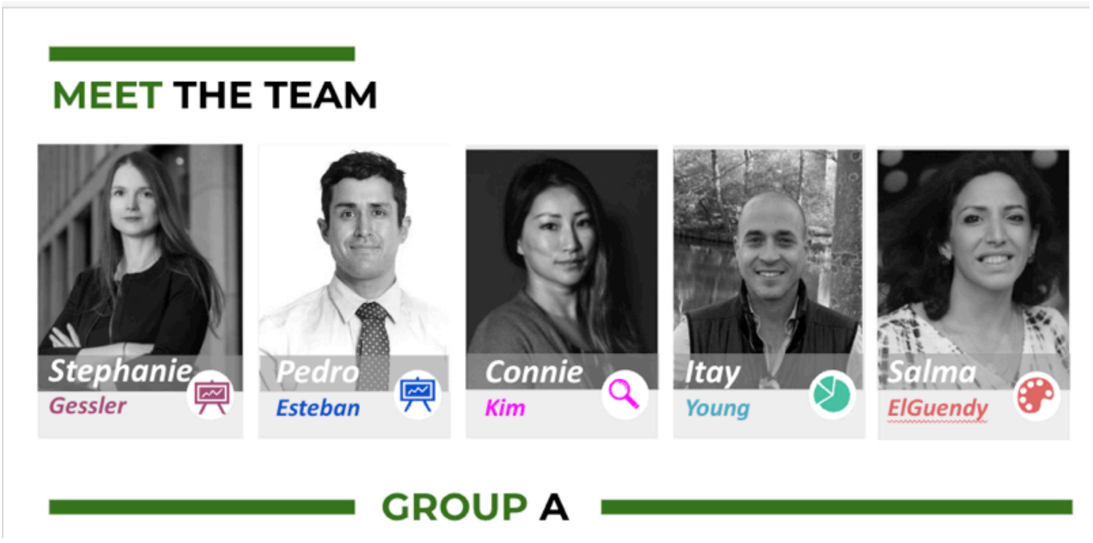
Group A:



Stephanie Gessler, Pedro V. Esteban, Itay Young, Salma ElGuendy, Connie Kim

Introduction

This is a continuation of forest_cover_type_detector_gr_a_Part3

This is a continuation of forest_cover_type_detector_gr_a_Part3.

Below we import the files created in the previous notebook so that this notebook can run independently

Table of contents is an extension of the previous notebook

Table of contents

Libraries used

In [1]:
```
#!pip install squarify
#!pip install GraphViz
#pip install pygraphviz
#!pip install pydotplus
#!pip install xgboost
#!pip install dtreeviz
#!pip install sklearn
```

In [2]:
```
import warnings
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_c
import matplotlib.pyplot as plt

import plotly.offline as py
```

```
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import seaborn as sns
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

import plotly.express as px

import pydotplus
import xgboost as xgb
import matplotlib
import squarify
```

In [3]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, class
from sklearn.model_selection import train_test_split, learning_curv

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve, ShuffleSplit
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import BernoulliNB #BernoulliNB is designe
from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz

from dtreeviz.trees import *

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import ExtraTreesClassifier, GradientBoosting
from sklearn.ensemble import AdaBoostClassifier

from sklearn import metrics
from sklearn import svm
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant_analysis import LinearDiscriminantAnalysi

from sklearn.decomposition import PCA

#from sklearn.lda import LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysi

from sklearn.svm import SVC

from sklearn.compose import make_column_transformer

from sklearn.manifold import TSNE

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

from sklearn.tree import export_graphviz

from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysi

from sklearn.manifold import TSNE

from IPython.display import Image
from IPython.core.display import display, HTML
from IPython.display import Image

from scipy.stats import uniform, randint
from scipy.stats import uniform, randint

from xgboost import XGBClassifier

from io import StringIO

from yellowbrick.classifier import ROCAUC
from yellowbrick.classifier import ClassPredictionError
from yellowbrick.style.palettes import PALETTES, SEQUENCES, color_p

warnings.simplefilter(action='ignore', category=FutureWarning)
display(HTML("<style>.container { width:100% !important; }</style>"
display(HTML("<style>.rendered_html { font-size: 16px; }</style>"))
```

```
/var/folders/ks/5bc1x9p158vgc4774v7r2tq40000gn/T/ipykernel_1877/41
04195847.py:70: DeprecationWarning:

Importing display from IPython.core.display is deprecated since IP
ython 7.14, please import from IPython display
```

1.Import the Data

1.1. Original Data

Let’s load the data

data_train = pd.read_csv("train.csv")

data_test = pd.read_csv("test.csv") pd.set_option('display.max_columns', None)

data_test.shape

```
column_list = data_train.columns
num  = [column for column in column_list if 'Soil' not in column
and 'Wilderness_Area' not in  column and 'Aspect_North' not in
column and 'Climate' not in  column and 'Family' not in  column
and 'Type' not in  column and 'complex' not in  column and
'Aspect_East' not in  column and 'Aspect_South' not in  column and
'Aspect_West' not in  column ]
cat= [column for column in column_list if column not in num]
```

```
#from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler()
scaler = StandardScaler()
```

1.2.All features & Standardization

Due to an excel import, it transforms to a new column unnamed

```
import pandas as pd
all_feat_df = pd.read_csv("all_features_data_train.csv")
all_feat_df =
all_feat_df[all_feat_df.columns.drop(list(all_feat_df.filter(regex
='Unnamed:')))]
pd.set_option('display.max_columns', None)
all_feat_df.head()
```

```
all_feat_df.shape
```

From 55 features, we engineered a total of 165 additional ones

For scaling we need to exclude the dummy variables

```
column_list = all_feat_df.columns
numerical  = [column for column in column_list if 'Soil' not in
column and 'Wilderness_Area' not in  column and 'Aspect_North' not
in  column and 'Climate' not in  column and 'Family' not in
column and 'Type' not in  column and 'complex' not in  column and
'Aspect_East' not in  column and 'Aspect_South' not in  column and
'Aspect_West' not in  column ]
categorial= [column for column in column_list if column not in
numerical]
```

1.3. Selected Model after Feature Selection & Standardization

Selected features from the feature selection are transferred to cvs and used for further analysis. Above code is not active, we import directly csv from notebook part 2

In [4]:
```
X_selected1 = pd.read_csv("X_selected.csv")
y_selected1 = pd.read_csv("y_selected.csv")
```

In [5]:
```
X_selected1 = X_selected1[X_selected1.columns.drop(list(X_selected1
y_selected1 = y_selected1[y_selected1.columns.drop(list(y_selected1
```

In [6]:
```
print(X_selected1.shape)
print(y_selected1.shape)

if X_selected1.shape[0] != y_selected1.shape[0]:
  print("X and y rows are mismatched, check dataset again")
```

```
(15120, 35)
(15120, 1)
```

We need to filter out the dummy variables for the normalization

In [7]:
```
column_list = X_selected1.columns
scale_numerical  = [column for column in column_list if 'Soil' not
scale_categorial= [column for column in column_list if column not i
```

In [8]:
```
X_train_new, X_val_new, y_train_new, y_val_new = train_test_split(X
```

In [9]:
```
scaler = StandardScaler()
X_train_new[scale_numerical] = scaler.fit_transform(X_train_new[sca
X_val_new[scale_numerical] = scaler.transform(X_val_new[scale_numer
```

In [10]:
```
print("The shape of validation data:{} and {} ".format(X_val_new.sh
print("The shape of training data:{} and {} ".format(X_train_new.sh
y_val_new = y_val_new.values.ravel()
y_train_new = y_train_new.values.ravel()
```

```
The shape of validation data:(3024, 35) and (3024, 1)
The shape of training data:(12096, 35) and (12096, 1)
```

2.Re-run models with the new selected features

Some classes such as SDG classifier , Random Forest classifier and naive Bayes classifier can handle mutliple classes naively.

Others like logistic regression or Support Vector Machine classifier are stricly binary classifier. However there are various strategies to perform multiclass classification with multiple binary classifiers.

In [12]:
```
# Create a dataFrame to compare performance of Classifier Models in
classifiers_compare = pd.DataFrame(columns =['Algorithm','Mean CV S
```

8.ML Algorithms after Dimensionality Reduction

8.1. PCA Dimensionality reduction

Principal component analysis (PCA) as the transformation of any high number of variables into a smaller number of uncorrelated variables called principal components (PCs), developed to capture as much of the data’s variance as possible.

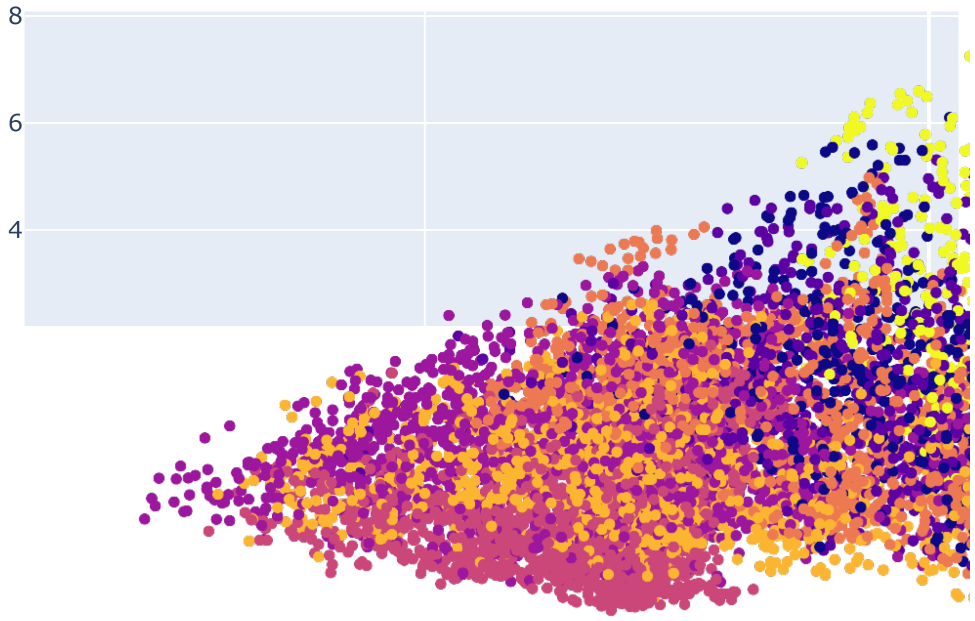
Principle components capture most of the variance of the data. The first principle components hold the most variance in the data, Each subsequent PCS is orthogonal to the last and has a lesser variance. In this way, given a set of x correlated variables over y samples, you achieve a set of uncorrelated PCS over the same y samples. We are running it on the dataset where we have all seleted features and will use PCA to identify the most important features.

For the visualization part we are using https://plotly.com/python/pca-visualization/ (https://plotly.com/python/pca-visualization/).

Using the first 2 principle components show that the data with 2 PCAs is very well deferiantiated. We are using all features to test it against the initial selected features from the previous models.

With only two PCA you can nicely see how well the data is differentiated among the different Cover Types.

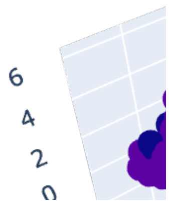
In [13]:
```
#Using a 2D Diagram for the two PCAs
import plotly.express as px
from sklearn.decomposition import PCA


pca = PCA(n_components=2)
#using the normalised dataset
components = pca.fit_transform(X_train_new)

fig = px.scatter(components, x=0, y=1, color=y_train_new)
fig.show()
```



First we will observe with a 3D Diagram the first three PCAs

In [14]:
```
pca = PCA(n_components=3)
components = pca.fit_transform(X_train_new)

total_var = pca.explained_variance_ratio_.sum() * 100

fig = px.scatter_3d(
    components, x=0, y=1, z=2, color=y_train_new,
    title=f'Total Explained Variance: {total_var:.2f}%',
    labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'}
)
fig.show()
```

Total Explained Variance: 73.55%

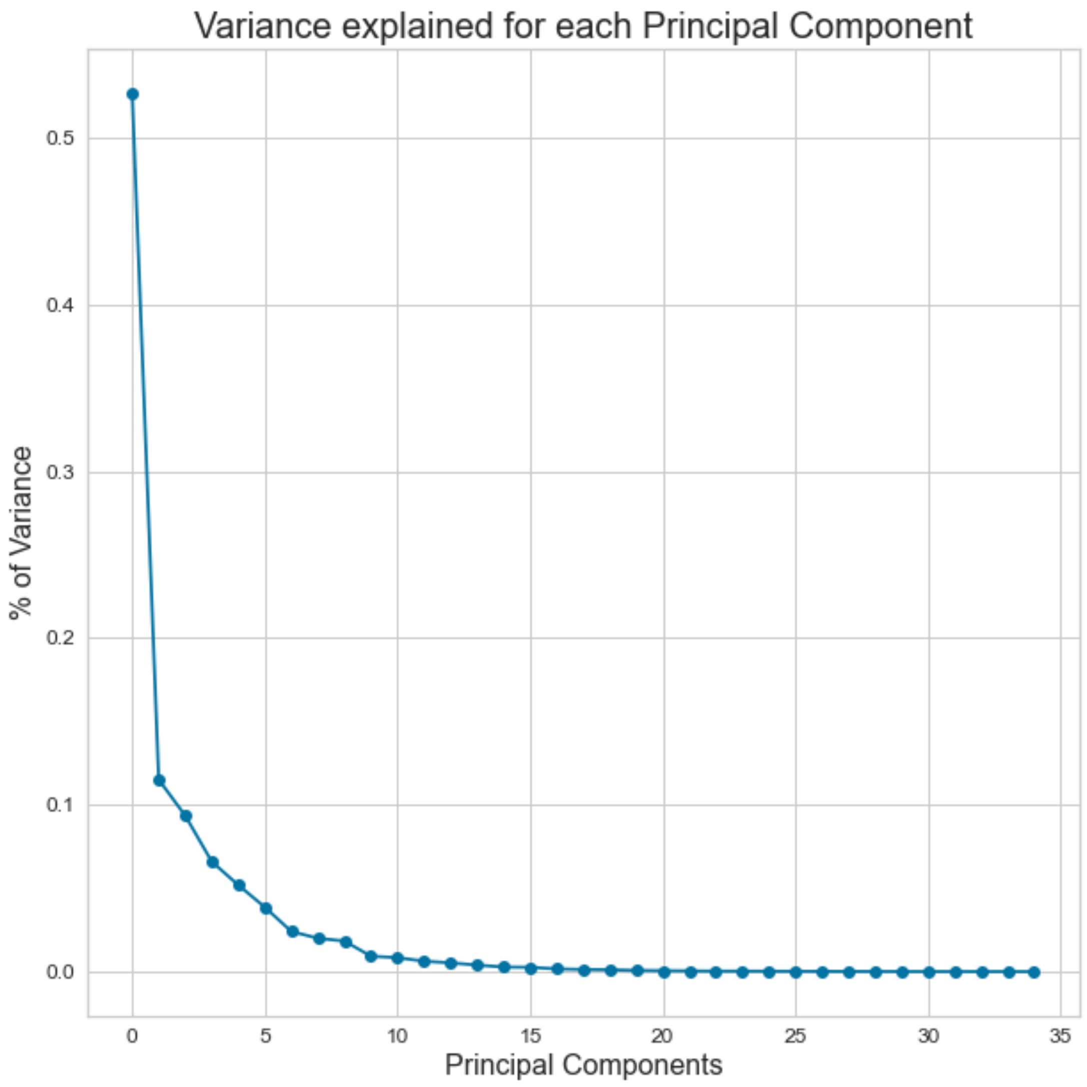


We run a PCA that holds 73% of all variance of the data

In [15]:
```
# Run PCA that holds 95% of all variance of the data
pca = PCA()
x_fit = pca.fit(X_train_new)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d=np.argmax(cumsum>=0.98)+1
```

Looking at the two first PCA to see if these can explain the dataset.You can set the components to the ratio of variance you wish to preserve

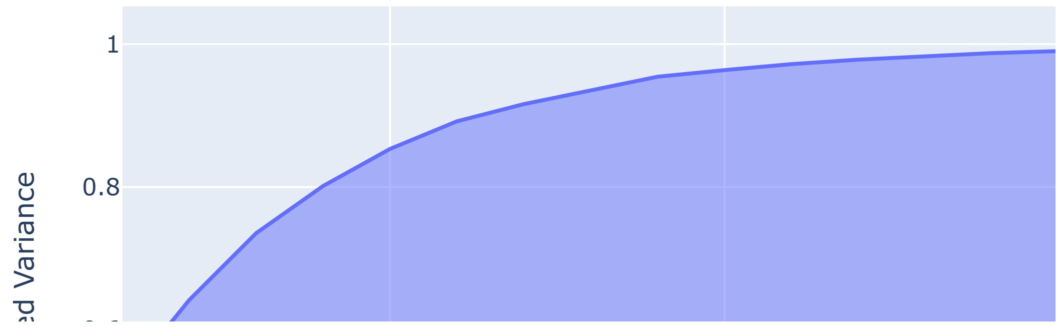
In [16]:
```
plt.figure(figsize=(10,10))
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.title("Variance explained for each Principal Component", fontsi
plt.plot(pca.explained_variance_ratio_, 'o-')
plt.xlabel("Principal Components", fontsize=16)
plt.ylabel("% of Variance", fontsize=16);
```



It is very clear that after 2 components, the curve flattens

In [17]:
```
exp_var_cumul = np.cumsum(pca.explained_variance_ratio_)
```

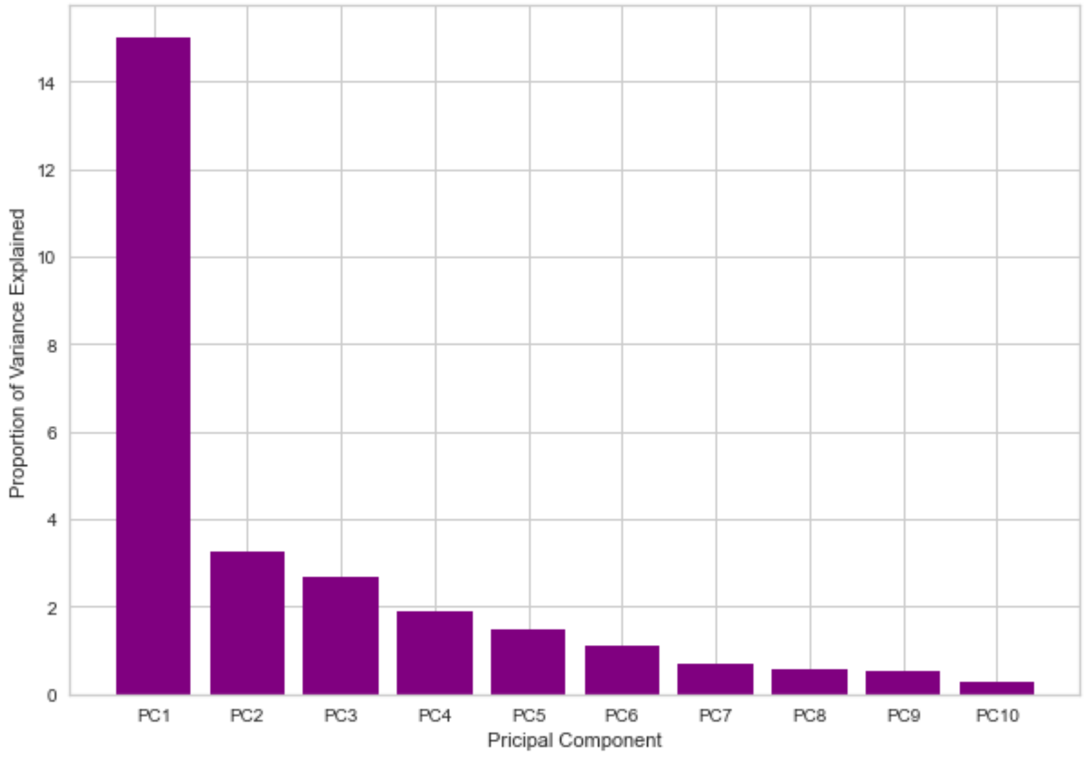
```
px.area(
    x=range(1, exp_var_cumul.shape[0] + 1),
    y=exp_var_cumul,
    labels={"x": "# of Principle Components", "y": "Cumulative Expl
```



It is a bar chart where the height of each bar is the percentage of variance explained by the associated PC.

```
In [18]: var = pca.explained_variance_[0:10] #percentage of variance explain
         labels = ['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC

         plt.figure(figsize=(10,7))
         plt.bar(labels,var,color=['purple'])
         plt.xlabel('Pricipal Component')
plt.ylabel('Proportion of Variance Explained')
```

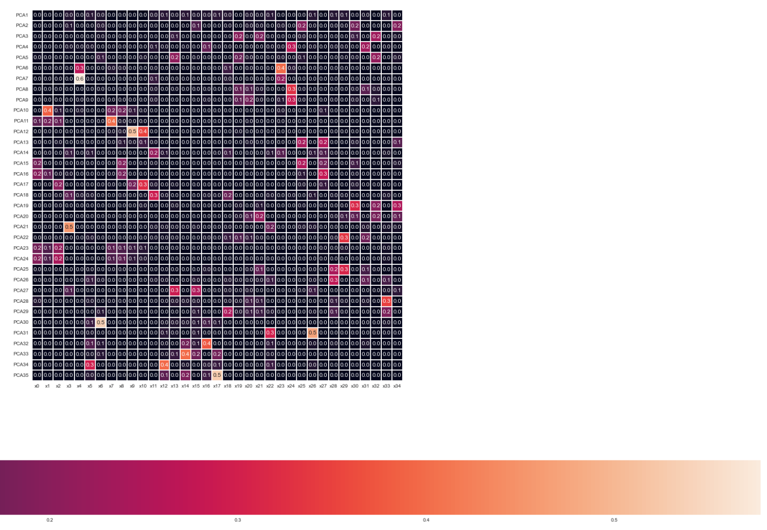
Out[18]:
```
Text(0, 0.5, 'Proportion of Variance Explained')
```



On top of that, we can also look at the combinations of variables that created each principal component with pca. We could use a heat map to showcase the correlation betweem the PCAs between each other.

In [19]:
```
#Heatmap PCA
fig, ax = plt.subplots(figsize=(40,20))
col_name = ['x' + str(idx) for idx in range(0, X_train_new.shape[1]

_ = sns.heatmap(pca.components_**2,
                 yticklabels=["PCA"+str(x) for x in range(1,pca.n_c
                 xticklabels=list(col_name),
                 annot=True,
                 fmt='.1f',
                 square=True,
                 linewidths=0.05,
                 cbar_kws={"orientation": "horizontal"})
```

```
              cbar_kws={"orientation": "horizontal"})
```



In [20]:
```
pca_ratio = pca.explained_variance_ratio_
pca_ratio_tranposed = list(map(list, zip(*[pca_ratio])))
df = pd.DataFrame(data = np.array(pca_ratio_tranposed), columns = [
df['PCA Variance Explained Cumulative'] = df['PCA Variance Explaine
df.index += 1
df
```

Out[20]:

	PCA Variance Explained	PCA Variance Explained Cumulative
1	5.268700e-01	0.526870
2	1.151225e-01	0.641992
3	9.352700e-02	0.735519
4	6.595294e-02	0.801472
5	5.189348e-02	0.853366
6	3.865947e-02	0.892025
7	2.407773e-02	0.916103
8	1.998892e-02	0.936092
9	1.832763e-02	0.954420
10	9.256484e-03	0.963676
11	8.373907e-03	0.972050
12	6.260343e-03	0.978310
13	5.260849e-03	0.983571
14	3.931265e-03	0.987502
15	2.767888e-03	0.990270
16	2.577739e-03	0.992848
17	1.641777e-03	0.994490
18	1.290626e-03	0.995780
19	1.113765e-03	0.996894
20	7.151339e-04	0.997609

21	5.299891e-04	0.998139
22	4.133632e-04	0.998553
23	2.911354e-04	0.998844
24	2.668395e-04	0.999111
25	2.411619e-04	0.999352
26	1.779521e-04	0.999530
27	1.523988e-04	0.999682
28	1.028169e-04	0.999785
29	9.209537e-05	0.999877
30	3.855903e-05	0.999916
31	3.164851e-05	0.999947
32	2.825740e-05	0.999976
33	1.971719e-05	0.999995
34	3.791688e-06	0.999999
35	8.854986e-07	1.000000

Looks Like most variance (97%) is explained once we reach 12 principal components, so let’s create training and validation data sets using the first 12 principal components

In [21]:
```
pca_mod = PCA(n_components=12)
PCA_X_train_final_selected = pca_mod.fit_transform(X_train_new)
PCA_X_val_final_selected = pca_mod.fit_transform(X_val_new)
print("Size of the dimensionality reduced training dataset", PCA_X_
print("Size of the dimensionality reduced training dataset", PCA_X_
```

```
Size of the dimensionality reduced training dataset (12096, 12)
Size of the dimensionality reduced training dataset (3024, 12)
```

In [22]:
```
PCA_X_train_final_selected[0]
```

Out[22]:
```
array([-3.49269467,  1.19244536, -0.13758727, -1.18074764,  1.7781
5774,
        1.78149742, -0.57081301, -2.35688513,  2.11612176, -0.4681
8732,
       -0.27141074,  0.52361574])
```

Check which most important features were selected using PCA.

In [23]:
```
pca_mod = PCA(n_components=12)
```

```
model = pca_mod.fit(X_train_new)
X_pc = model.transform(X_train_new)

# number of components
n_pcs= model.components_.shape[0]

# get the index of the most important feature on EACH component
# LIST COMPREHENSION HERE
most_important = [np.abs(model.components_[i]).argmax() for i in ra

# get the names
most_important_names = [X_train_new.columns[most_important[i]] for

# LIST COMPREHENSION HERE AGAIN
dic = {'PC{}'.format(i): most_important_names[i] for i in range(n_p

# build the dataframe
df = pd.DataFrame(dic.items())
```

In [24]:
```
df
```

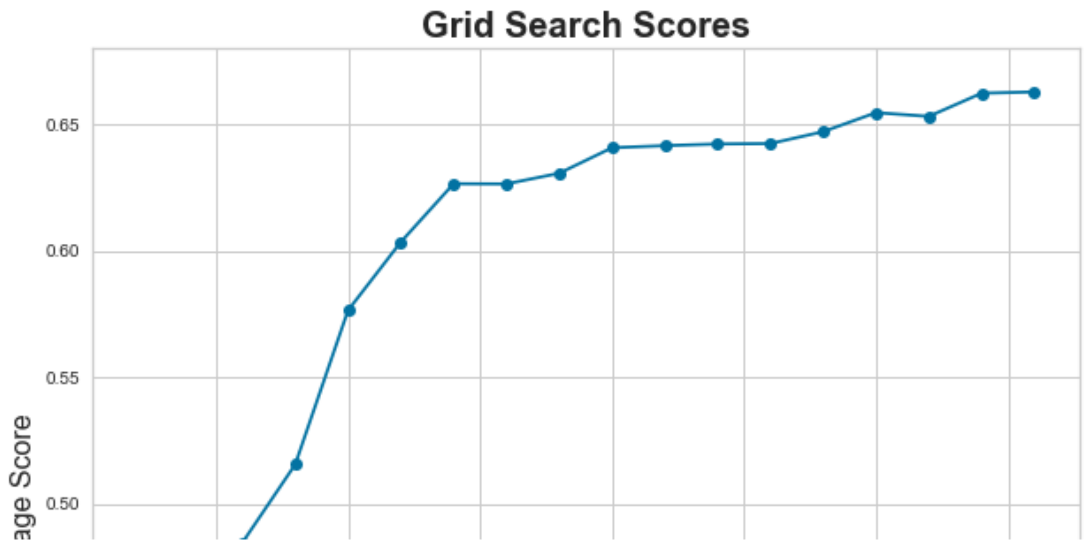
Out[24]:

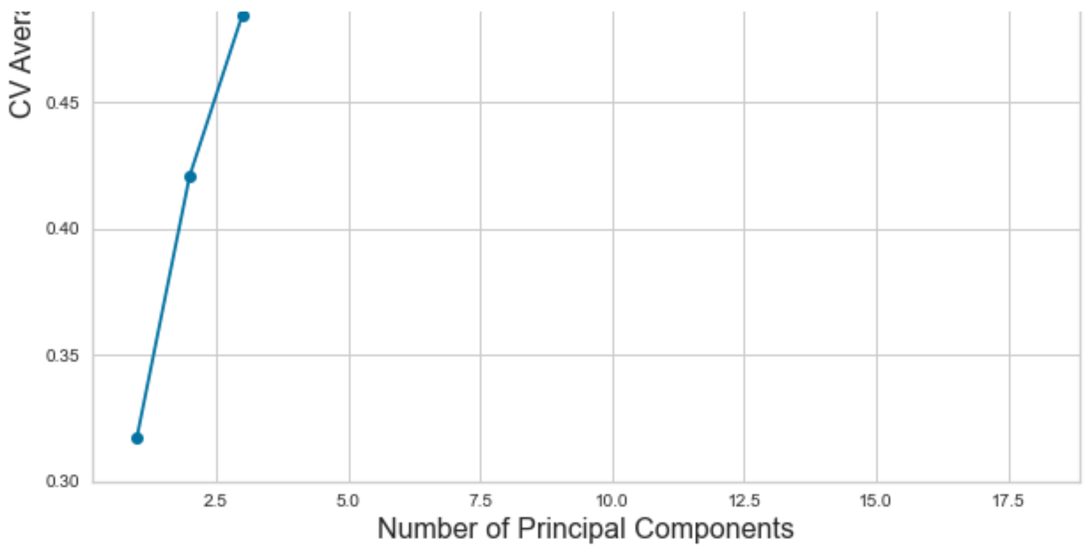
	0	1
0	PC0	sqr_Ele+Road+Fire+Hydro
1	PC1	Elevation_x_Horizontal_Distance_To_Hydrology
2	PC2	Hillshade_9am_boxcox
3	PC3	sqr_Ele-fire
4	PC4	Hillshade_9am_boxcox
5	PC5	sqr_Ele-road
6	PC6	sqr_Road-Fire
7	PC7	sqr_Ele-fire
8	PC8	sqr_Ele-fire
9	PC9	Subalpine_Climate
10	PC10	Wilderness_Area3
11	PC11	Montane_Climate

Let's use the pipeline to wrap everything together and to find the best configuration for the different hyper-parameters

In [25]:
```
pipeline = Pipeline(steps=[('scaler', StandardScaler()), ('pca', PC

n_components = list(range(1,19)) # We will try different numbers of

#Parameters of pipelines can be set using '__' separated parameter
param_grid = {"pca__n_components":n_components}

estimator = GridSearchCV(pipeline, param_grid, cv=5) # Create a gri
estimator.fit(X_train_new, y_train_new);
```

In [26]:
```
# Get Test Scores Mean and std for each grid search
scores_mean = estimator.cv_results_['mean_test_score']
scores_mean = np.array(scores_mean)

scores_sd = estimator.cv_results_['std_test_score']
scores_sd = np.array(scores_sd)

# Plot Grid search scores
plt.figure(figsize=(10,10))
plt.plot(n_components, scores_mean, '-o')

plt.title("Grid Search Scores", fontsize=20, fontweight='bold')
plt.xlabel("Number of Principal Components", fontsize=16)
plt.ylabel('CV Average Score', fontsize=16)
plt.grid('on')
```





As can be seen in the graph, the optimal number of PCs to select will 12 as this gives us the optimal CV score. The optimum is 12 and increasing the number of features does not add much value.

Source codes: https://www.kaggle.com/code/maniyar2jaimin/interactive-plotly-guide-to-pca-lda-t-sne (https://www.kaggle.com/code/maniyar2jaimin/interactive-plotly-guide-to-pca-lda-t-sne), https://github.com/dasaditi/machineLearning/blob/master/forestCover/ForestC (https://github.com/dasaditi/machineLearning/blob/master/forestCover/ForestC

8.1.1. Random Forest

Fine tuning the model and checking the performance of the PCA method

In [27]:
```
n_estimators = list(range(50, 250,10))
criterion=['gini','entropy']
#min_samples_split = list(range(5, 25))
max_depth = list(range(1, 10,1))
# create a parameter grid: map the parameter names to the values th
param_grid_rf = dict(n_estimators=n_estimators,criterion=criterion,

# instantiate and fit the grid
rf = RandomForestClassifier()
grid_rf = RandomizedSearchCV(rf, param_grid_rf, cv=5, scoring='accu

grid_rf.fit(PCA_X_train_final_selected, y_train_new)
```

Out[27]:
```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

Here we get the best combination of hyperparameters yielding the best score

In [28]:
```
#Mean cross-validated score of the best_estimator
print("The best score: ",grid_rf.best_score_.round(4))
#Parameter setting that gave the best results on the hold out data.
print("The best parameter: ",grid_rf.best_params_)
```

```
The best score:  0.7765
The best parameter:  {'n_estimators': 130, 'max_depth': 9, 'criter
ion': 'gini'}
```

In [29]:
```
rf_final = RandomForestClassifier(criterion=grid_rf.best_params_['c
                            n_estimators=grid_rf.best_params_['n_es
rf_final.fit(PCA_X_train_final_selected,y_train_new)
rf_score=rf_final.score(PCA_X_val_final_selected,y_val_new)

print("The accuracy score of the RandomForest with reduced feature
```

```
The accuracy score of the RandomForest with reduced feature set:
0.68
```

With cross validation you can see it overfits the train data und the result of the validation data is lower

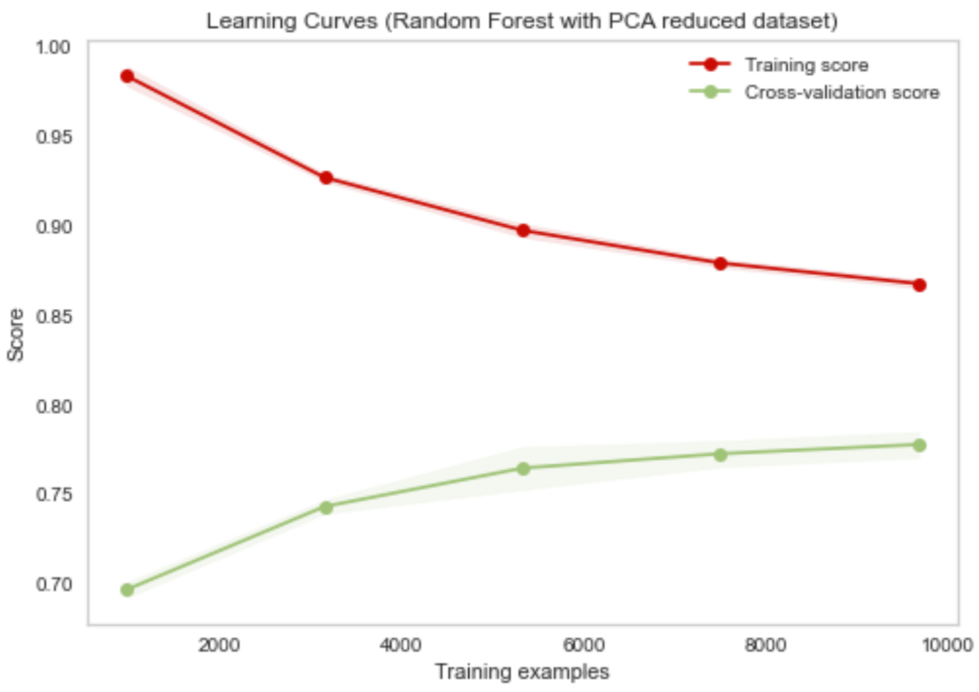
In [30]:
```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(rf_final,
```

```
Accuracy = 0.7447
```

As you can see the model is overitting on the Training Set and has a lower score on the cross validation score.

In [31]:
```
#the plot learning curve from sklearn works only if you add the def


def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0,

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_si
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_
                     train_scores_mean + train_scores_std, alpha=0.
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_st
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

In [32]:
```
#cv = ShuffleSplit(n_splits=500, test_size=0.2, random_state=0)
title = "Learning Curves (Random Forest with PCA reduced dataset)"
plot_learning_curve(rf_final, title, PCA_X_train_final_selected, y_
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0,

plt.show()
```



8.1.2 XGBoost

We continue following the same dynamic, fitting the pca df and finiding the best hyperparameters

In [33]:
```
le = LabelEncoder()
y_train_PCA = le.fit_transform(y_train_new)
y_val_PCA = le.fit_transform(y_val_new)
```

In [34]:
```
n_estimators = [20,50,100]
#min_samples_split = list(range(5, 25))
max_depth = list(range(1, 10,5))
learning_rate = [0.05,0.1,0.5]
# create a parameter grid: map the parameter names to the values th
param_grid_xgb_model = dict(n_estimators=n_estimators,max_depth=max

# instantiate and fit the grid
xgb_model = XGBClassifier()
grid_xgb_model = RandomizedSearchCV(xgb_model, param_grid_xgb_model

grid_xgb_model.fit(PCA_X_train_final_selected, y_train_PCA)
```

Out[34]:
```
▸    RandomizedSearchCV
▸ estimator: XGBClassifier
       ▸ XGBClassifier
```

In [35]:
```
#Mean cross-validated score of the best_estimator
print("The best score: ",grid_xgb_model.best_score_.round(4))
#Parameter setting that gave the best results on the hold out data.
print("The best parameter: ",grid_xgb_model.best_params_)
```

```
The best score:  0.8382
The best parameter:  {'n_estimators': 100, 'max_depth': 6, 'learni
ng_rate': 0.5}
```

In [36]:
```
xgb_final = XGBClassifier(learning_rate=grid_xgb_model.best_params_
                            n_estimators=grid_xgb_model.best_params
xgb_final.fit(PCA_X_train_final_selected,y_train_PCA)
xgb_score=xgb_final.score(PCA_X_val_final_selected,y_val_PCA)

print("The accuracy score of the XGBClassifier with reduced feature
```

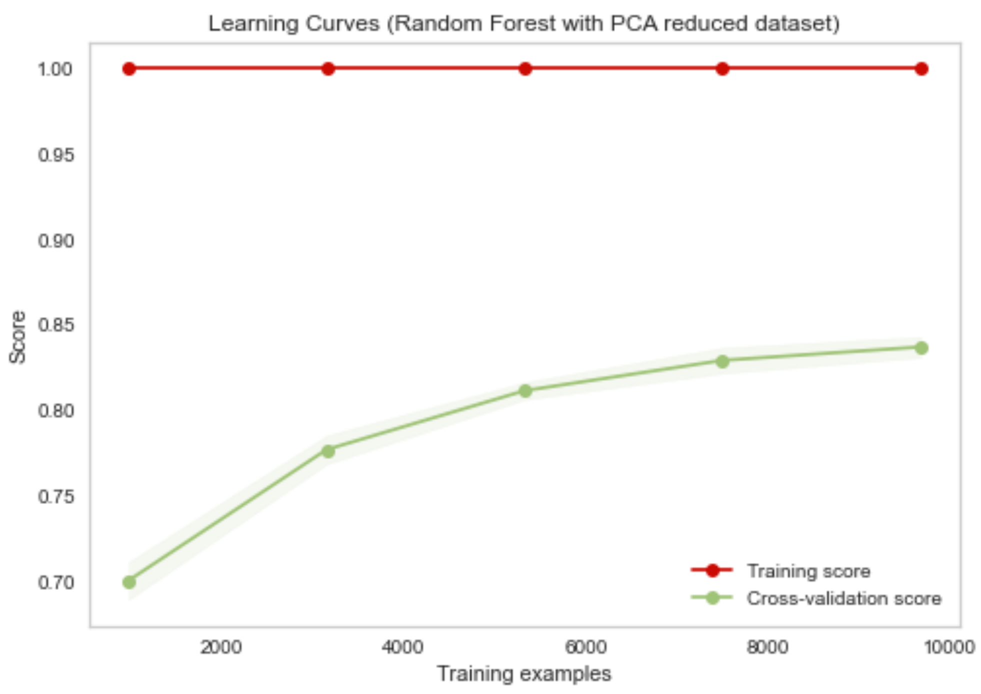
```
The accuracy score of the XGBClassifier with reduced feature set:
0.65
```

In [37]:
```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(xgb_final
```

```
Accuracy = 0.7659
```

The resulting score is in line with the ones achieved so far

In [38]:
```
title = "Learning Curves (Random Forest with PCA reduced dataset)"
plot_learning_curve(xgb_final, title, PCA_X_train_final_selected, y
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0,
plt.show()
```



8.1.3 Extra Trees Classifier

In [39]:
```
# Build a forest and compute the feature importances
n_estimators = list(range(50, 250,5))
criterion=['gini','entropy']
#min_samples_leaf = list(range(5, 25))
#min_samples_split = list(range(5, 25))
max_depth = list(range(1, 10))
# create a parameter grid: map the parameter names to the values th
param_grid = dict(n_estimators=n_estimators, criterion=criterion,ma
forest = ExtraTreesClassifier(random_state=0)
grid_etc = RandomizedSearchCV(forest, param_grid, cv=5, scoring="ac
grid_etc.fit(PCA_X_train_final_selected, y_train_new)
print("The best score: ",grid_etc.best_score_.round(4))
#Parameter setting that gave the best results on the hold out data.
print("The best parameter: ",grid_etc.best_params_)
grid_etc.best_estimator_
```

```
The best score:  0.7159
The best parameter:  {'n_estimators': 55, 'max_depth': 8, 'criteri
on': 'gini'}
```

Out[39]:

```
▾                    ExtraTreesClassifier
ExtraTreesClassifier(max_depth=8, n_estimators=55, rando
m_state=0)
```

In [40]:

```
 #Find Feature importance
etc_selected = ExtraTreesClassifier(n_estimators=grid_etc.best_para
                            criterion=grid_etc.best_params_['criter
etc_selected.fit(PCA_X_train_final_selected, y_train_new)
etc_sel_acc = etc_selected.score(PCA_X_val_final_selected, y_val_ne
print("The accuracy score of the ExtraTreesClassifier with the sele
print("CV Accuracy = {0:.4f}".format(np.mean(cross_val_score(etc_se
```

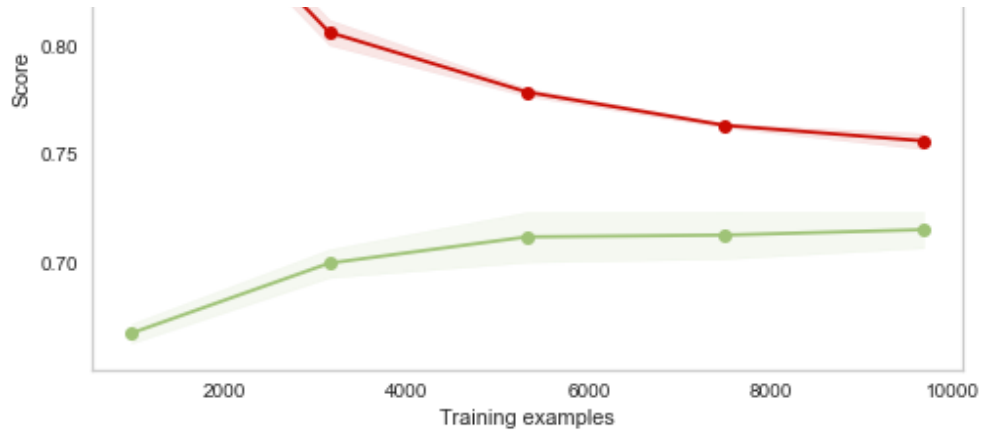
```
The accuracy score of the ExtraTreesClassifier with the selected f
eatures:  0.64
CV Accuracy = 0.6974
```

The CV score is somehow smaller than with previous methods

In [41]:

```
title = "Learning Curves (Random Forest with PCA reduced dataset)"
plot_learning_curve(etc_selected, title, PCA_X_train_final_selected
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0,

plt.show()
```

Learning Curves (Random Forest with PCA reduced dataset)
0.90
0.85
Training score
Cross-validation score



8.1.4 Ensemble Methods

Voting Classifier

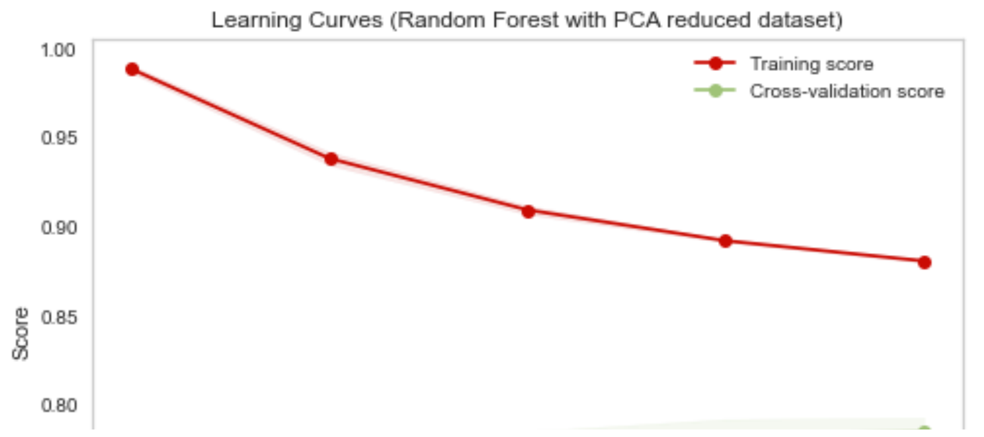
A very simple way to create an even better classifier is to aggregate the best predictions of each classifier and predict the class that gets the most votes. Normally this method is a better predictor than the single models.

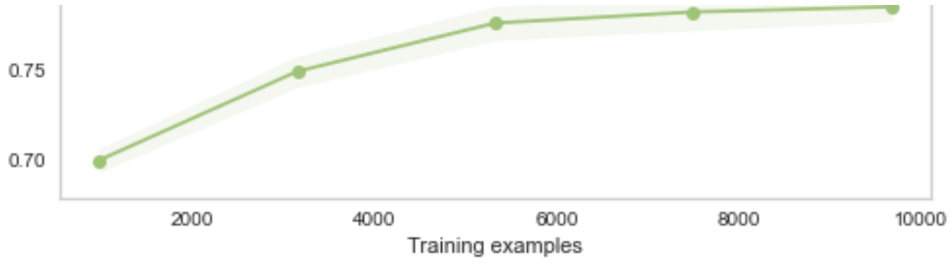
In [42]:
```
eclf = VotingClassifier(estimators=[
    ('Extra Tree', etc_selected), ('Random Forest',rf_final),('xgbo
eclf.fit(PCA_X_train_final_selected, y_train_PCA)
y_val_pred = eclf.predict(PCA_X_val_final_selected)
print("CV Accuracy = {0:.4f}".format(np.mean(cross_val_score(eclf,
```

```
CV Accuracy = 0.7513
```

Because we have more models scoring lower than 80%, it brings the score to 75%

In [43]:
```
title = "Learning Curves (Random Forest with PCA reduced dataset)"
plot_learning_curve(eclf, title, PCA_X_train_final_selected, y_trai
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0,

plt.show()
```





8.2. Linear Discriminant Analysis (LDA)

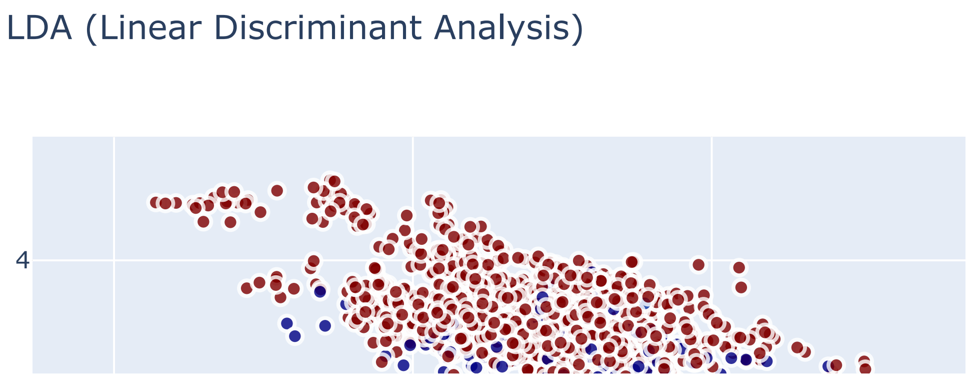
LDA, much like PCA is also a linear transformation method commonly used in dimensionality reduction tasks. However unlike the latter which is an unsupervised learning algorithm, LDA falls into the class of supervised learning methods.

As such the goal of LDA is that with available information about class labels, LDA will seek to maximise the separation between the different classes by computing the component axes (linear discriminants) which does this.

In [44]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysi
lda= lda(n_components=5) # n_components in this case is 1 less than
# Taking in as second argument the Target as labels
X_LDA_2D = lda.fit_transform(X_train_new, y_train_new )
```

In [45]:
```
# Create a classifier: a Fisher's LDA classifier
lda = LinearDiscriminantAnalysis(n_components=2)

# Train lda on the first half of the digits
X_LDA_2D = lda.fit_transform(X_train_new, y_train_new,)
```

In [46]:
```
# To produce an interactive chart
traceLDA = go.Scatter(
    x = X_LDA_2D[:,0],
    y = X_LDA_2D[:,1],
    name = '',
    mode = 'markers',
#     text = Target.unique(),
    showlegend = True,
    marker = dict(
        size = 8,
        color = y_train_new,
        colorscale ='Jet',
        showscale = False,
        line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        ),
        opacity = 0.8
    )
)
data = [traceLDA]

layout = dict(title = 'LDA (Linear Discriminant Analysis)',
              hovermode= 'closest',
              yaxis = dict(zeroline = False),
              xaxis = dict(zeroline = False),
              showlegend= True
             )

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-scatter')
```



Machine Learning II
Group Assignment

Group A