final_code

February 15, 2019

1 Workflow

1. Objective - Define Problem and Metrics
2. Import and Store Data
3. Data Exploration and Data Cleaning
4. Feature Engineering from Raw Data
5. Classification (Modelling)
6. Communicating Results (visualizations included)

1.1 1. Objective

Implement a "Hand Geometry based Person Identification" system using supervised learning. Training & Testing data with labels is provided. The classification system will be measured by accuracy.

```
In [635]: #Libraries
          import numpy as np
          import pandas as pd
          import seaborn as sns
          import matplotlib.pyplot as plt
          plt.rcParams['figure.figsize'] = [24, 10]
          sns.set(style="darkgrid")
          plt.rcParams['figure.figsize'] = [24, 10]
```

2 2. Import & Store Data

```
In [72]: def getxLabel(x):
             return "{:02d}x".format(x)

         def getyLabel(y):
             return "{:02d}y".format(y)
```

```
In [706]: NPOINTS = 23
          data_cols = ['label']
          for p in range(1,NPOINTS+1):
              data_cols.append(getxLabel(p))
              data_cols.append(getyLabel(p))
```

```
df_train = pd.read_csv('Assignment2-handtrainfile.txt',delimiter=' ',header=None,name
df_test = pd.read_csv('Assignment2-handtestfile.txt',delimiter=' ',header=None,names=

print("TRAIN DATA:", df_train.shape, "#ofUniqIDs:",len(df_train.label.unique()))
print("TEST DATA:", df_test.shape)
```

```
TRAIN DATA: (100, 47) #ofUniqIDs: 20
TEST DATA: (100, 47)
```

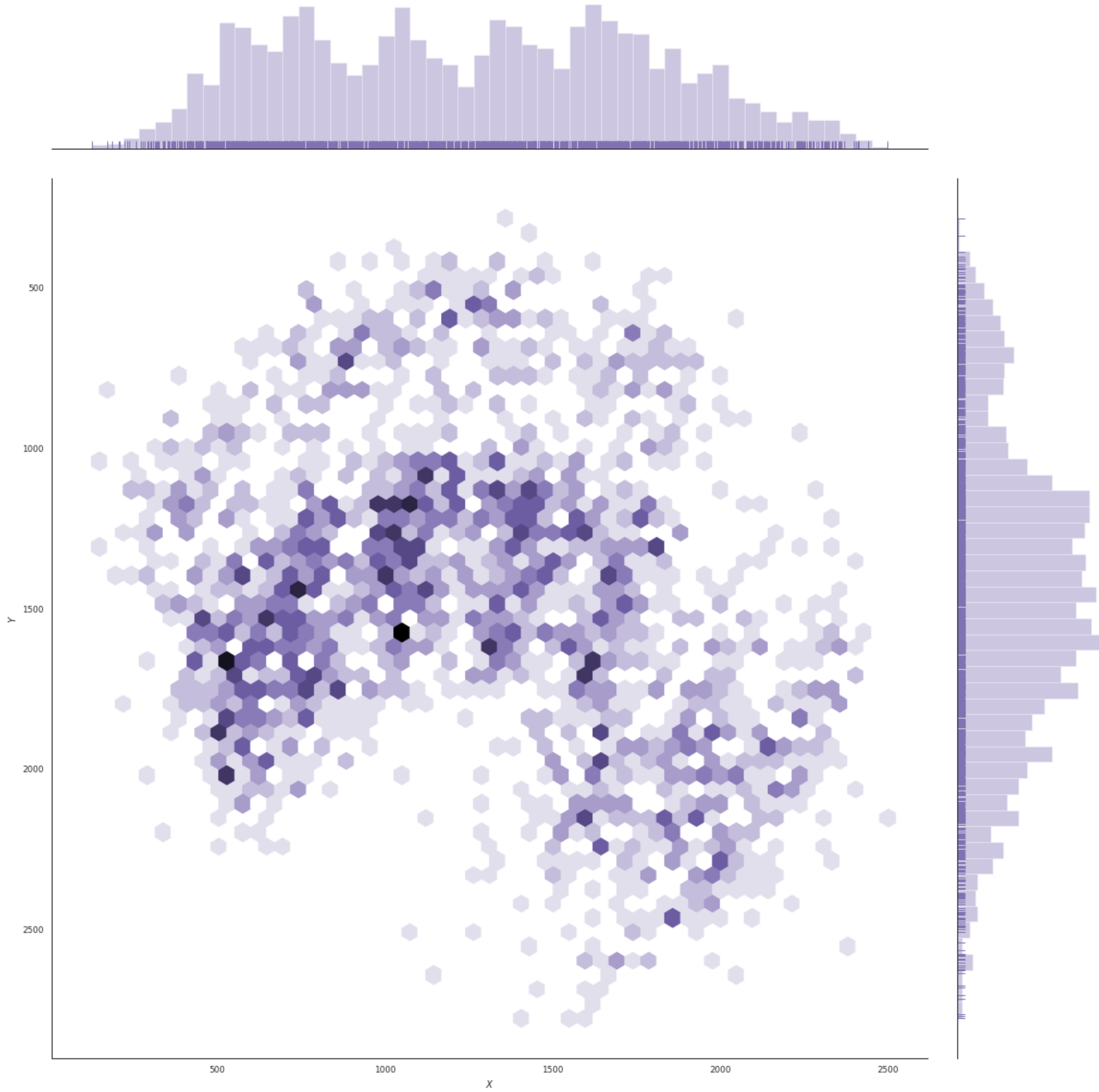
In [707]:
```
#Convert labels to integer codes
df_train['label'], mapping_index = pd.Series(df_train['label']).factorize()
```

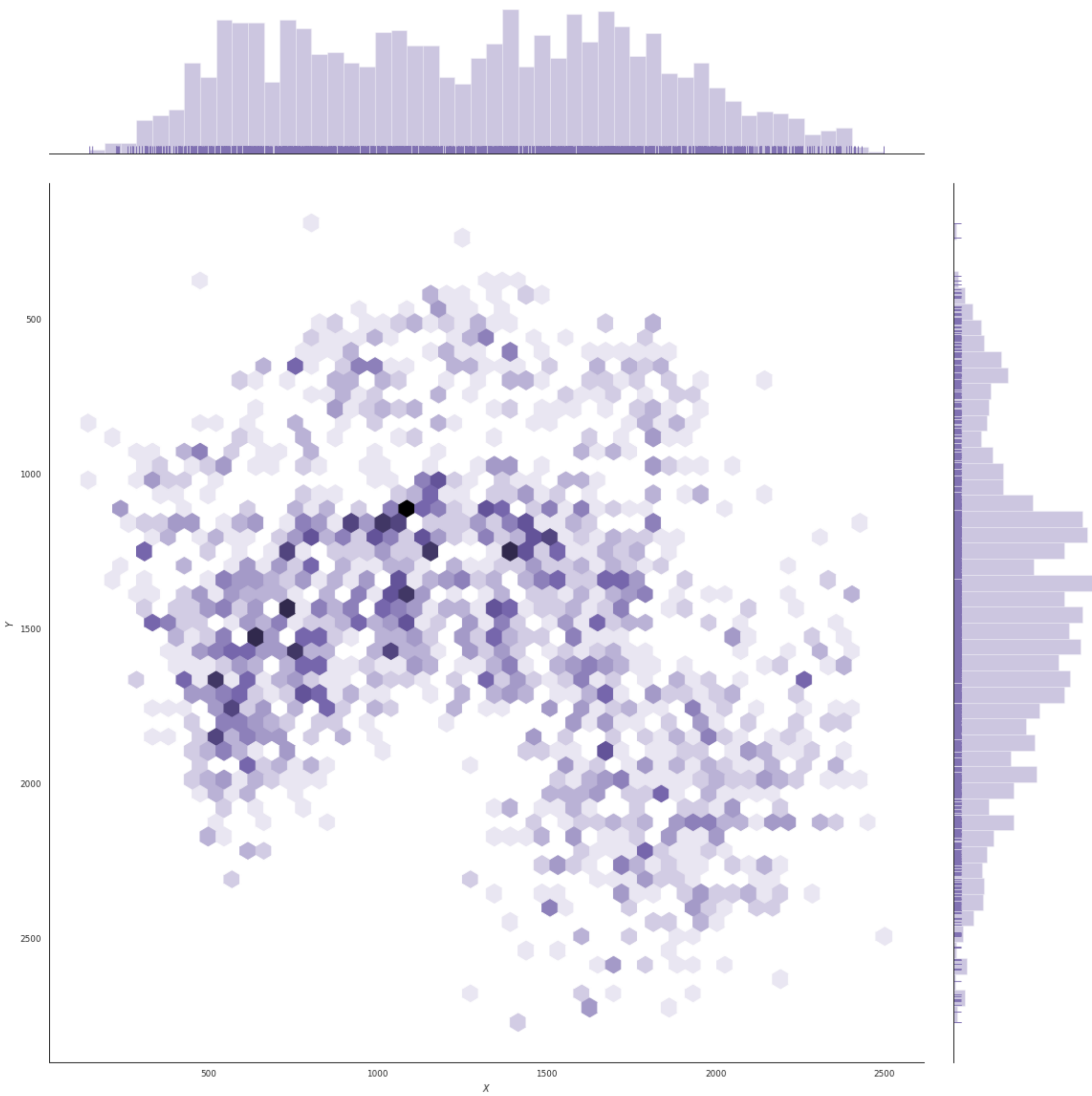
3 3. Data Exploration and Cleaning

- what kind of data do we have? => Numerical only
- Is there any missing Data? => No.
- Are there outliers, should I care> => No.

In [843]:
```
#Plot highlevel view of data to confirm it is what we expect
sns.set_style("white")
def plotHexBins(df):
    x = df['01x'].values
    y = df['01y'].values

    for p in range(2,NPOINTS+1):
        x = np.append(x,df[getxLabel(p)].values)
        y = np.append(y,df[getyLabel(p)].values)

    g = sns.jointplot(x=x, y=y, kind="hex", color="m", height=16, marginal_kws=dict(
                      gridsize=50)
    # g.plot_joint(plt.scatter, c="w", s=30, linewidth=1, marker="+")
    # g.ax_joint.collections[0].set_alpha(0)
    g.set_axis_labels("$X$", "$Y$")
    g.fig.axes[0].invert_yaxis()
    # g.fig.axes[0].invert_xaxis()
    plt.show()

sns.despine()
plotHexBins(df_train)
plotHexBins(df_test)
```

```
<Figure size 1152x864 with 0 Axes>
```


500
1000
1500
2000
2500
Y
500
1000
1500
2000
2500
X



4 4. Feature Engineering

- Distance between each point?
- triangulation of all pairs? area of triangle?
- Removing highly correlated?

```
In [856]: %%time
          from sklearn.metrics.pairwise import paired_distances

          def getListofPoints(df, p):
              return df[[getxLabel(p),getyLabel(p)]].values.tolist()

          def getEdColname(p1,p2):
```

```
    return "{:02d}_e_{:02d}".format(p1,p2)

def getMdColname(p1,p2):
    return "{:02d}_m_{:02d}".format(p1,p2)

def getDistances(df):
    '''
    Calculates euclidean and manhattan distances between all possible pairs
    and stores the distances in the data frame.
    RETURNS:
        - DataFrame
    '''

    for p1 in range(1,NPOINTS+1):
        p1s = getListofPoints(df, p1)
        for p2 in range(p1+1,NPOINTS+1):
            if p1==p2:
                continue
            p2s = getListofPoints(df, p2)
            colname = getEdColname(p1,p2)
            df[colname] = paired_distances(p1s,p2s,'euclidean')
            colname = getMdColname(p1,p2)
            df[colname] = paired_distances(p1s,p2s,'manhattan')
    return df


df_trainf = getDistances(df_train.copy())
df_testf = getDistances(df_test.copy())
```

```
CPU times: user 850 ms, sys: 0 ns, total: 850 ms
Wall time: 858 ms
```

In [857]:
```
%%time
import itertools

def getPermiterColname(comb):
    return "P_{:02d}_{:02d}_{:02d}".format(comb[0],comb[1],comb[2])

def getTAreaColname(comb):
    return "TA_{:02d}_{:02d}_{:02d}".format(comb[0],comb[1],comb[2])

def getSideLength(df,p1,p2):
    try:
        sl = df[getEdColname(p1,p2)]
    except:
        sl = df[getEdColname(p2,p1)]
    return sl
```

```
def getTriangleFeatures(df):
    '''
    Calculates Triangle perimeter and area between all possible triplets of points
    and stores the values in the data frame.
    RETURNS:
        - DataFrame
    ''''
    for comb in itertools.combinations(range(1,NPOINTS+1), 3):
        a = getSideLength(df,comb[0],comb[1])
        b = getSideLength(df,comb[0],comb[2])
        c = getSideLength(df,comb[1],comb[2])
        df[getPermiterColname(comb)] = (a + b + c)

        s = df[getPermiterColname(comb)] / 2
        df[getTAreaColname(comb)] = np.sqrt((s*(s-a)*(s-b)*(s-c)))

    return df

df_trainf = getTriangleFeatures(df_trainf)
df_testf = getTriangleFeatures(df_testf)
```

```
CPU times: user 6.51 s, sys: 0 ns, total: 6.51 s
Wall time: 6.51 s
```

In [858]:
```
%%time
def getRidOfPointVars(df):
    '''
    Drops the raw location points (they are not valuable features, we do not need th
    RETURNS:
        - DataFrame
    ''''
    to_drop = []
    for p in range(1,NPOINTS+1):
        to_drop.append(getxLabel(p))
        to_drop.append(getyLabel(p))
    return df.drop(columns=to_drop)

df_trainf = getRidOfPointVars(df_trainf)
df_testf = getRidOfPointVars(df_testf)

print('TOTAL FEATURE COLUMNS:', len(df_trainf.columns))

#drop highly correlated features

# Compute the correlation matrix
corr = df_trainf.drop(columns=['label']).corr()
```

```
# Select upper triangle of correlation matrix
upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.90 and drop them fro
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]

df_trainf = df_trainf.drop(columns=to_drop)
df_testf = df_testf.drop(columns=to_drop)


feature_columns = df_trainf.drop(columns=['label']).columns
print('FEAT COLUMNS REMAINING:',len(feature_columns))

#Convert data to Numpy Arrays
y = df_trainf.label.values
X = df_trainf.drop(columns=['label']).values

y_test = df_testf.label.values
X_test = df_testf.drop(columns=['label']).values
```

```
TOTAL FEATURE COLUMNS: 4049
FEAT COLUMNS REMAINING: 460
CPU times: user 4.02 s, sys: 160 ms, total: 4.18 s
Wall time: 4.06 s
```

5 5. Classification (Modelling)

- Feature Scaling
- Feature selection / reduction
- Stratified Cross Validation
- Testing Classifiers / hyper parameter tuning

In [859]:
```
#Copy training data into other variables
X_train, y_train = X, y
```

In [860]:
```
%%time

#Import all classification and pre processing libraries we need
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline, Pipeline

from sklearn.svm import SVC, LinearSVC
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
```

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.gaussian_process import GaussianProcessClassifier

from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score, precision_score, recall_score

from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.gaussian_process.kernels import RBF

def get_name(estimator):
    name = estimator.__class__.__name__
    if name == 'Pipeline':
        name = [get_name(est[1]) for est in estimator.steps]
        name = ' + '.join(name)
    return name
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 66.8 ţs
```

In [864]:
```
%%time

#Define pipeline consisting of scaling, reduction, and classification
pipe = Pipeline([
    ('scaler', preprocessing.MinMaxScaler()),
    ('reduce_dim', PCA()),
    ('classify', LinearSVC())
])

#Create arrays of possible options for pipeline
scalers = [preprocessing.MinMaxScaler(),preprocessing.StandardScaler()]
classifiers = [LinearSVC(),SVC(gamma='scale'),RandomForestClassifier(), GaussianProce
MAX_ITER_OPT = range(100,1000,100)
N_FEATURES_OPTIONS = [15,20,25,30]
C_OPTIONS = np.logspace(-2, 7, 10)
n_estimators_opt = range(25,75,25)
max_depth_opt = range(2,4,1)
RANDOM_STATES = [0,5,42]

#Define the parameter grid for experimentation
#Joins together all appropriate combination of scaling, reduction/selection, and cla
param_grid = [
    #SVC Classifier options
    {
```

```
        'scaler': scalers,
        'reduce_dim': [PCA(iterated_power=7)],
        'reduce_dim__n_components': N_FEATURES_OPTIONS,
        'classify': classifiers[:2],
        'classify__C': C_OPTIONS
    },
    {
        'scaler': scalers,
        'reduce_dim': [SelectKBest(f_classif)],
        'reduce_dim__k': N_FEATURES_OPTIONS,
        'classify': classifiers[:2],
        'classify__C': C_OPTIONS
    },
    #Gaussian Process Classifier Option
    {
        'scaler': scalers,
        'reduce_dim': [PCA(iterated_power=7)],
        'reduce_dim__n_components': N_FEATURES_OPTIONS,
        'classify': classifiers[3:],
        'classify__random_state': RANDOM_STATES
    },
    {
        'scaler': scalers,
        'reduce_dim': [SelectKBest(f_classif)],
        'reduce_dim__k': N_FEATURES_OPTIONS,
        'classify': classifiers[3:],
        'classify__random_state': RANDOM_STATES
    },
    #Random Forest Classifier Options
    {
        'scaler': scalers,
        'reduce_dim': [PCA(iterated_power=7)],
        'reduce_dim__n_components': N_FEATURES_OPTIONS,
        'classify': classifiers[2:3],
        'classify__n_estimators': n_estimators_opt,
        'classify__max_depth': max_depth_opt,
    },
    {
        'scaler': scalers,
        'reduce_dim': [SelectKBest(f_classif)],
        'reduce_dim__k': N_FEATURES_OPTIONS,
        'classify': classifiers[2:3],
        'classify__n_estimators': n_estimators_opt,
        'classify__max_depth': max_depth_opt,
    },
]

#Define scoring metric
```

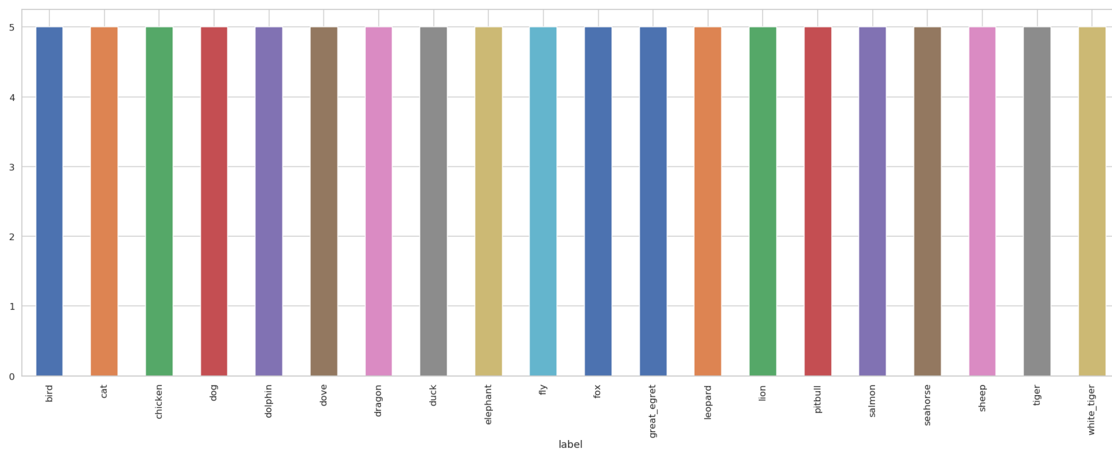
```
scoring = {'Accuracy': make_scorer(accuracy_score)}

#Create 5-fold classification cross validation grid
grid = GridSearchCV(pipe, cv=5, param_grid=param_grid,
                   scoring=scoring, refit='Accuracy',
                    return_train_score=True)

with ignore_warnings(category=ConvergenceWarning):
    grid.fit(X_train, y_train)

print("BEST CV SCORE:", grid.best_score_)
print("BEST CONFIGURATION:")
print(grid.best_params_)

#Save results into a dataframe locally
results_df = pd.DataFrame(grid.cv_results_)
results_df.to_pickle('results_df.pkl')
```

```
BEST CV SCORE: 0.98
BEST CONFIGURATION:
{'classify': SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False), 'classify__C': 10.0, 'reduce_dim': SelectKBest(k=15, score_func=<
CPU times: user 2min 40s, sys: 3min 19s, total: 5min 59s
Wall time: 1min 13s
```

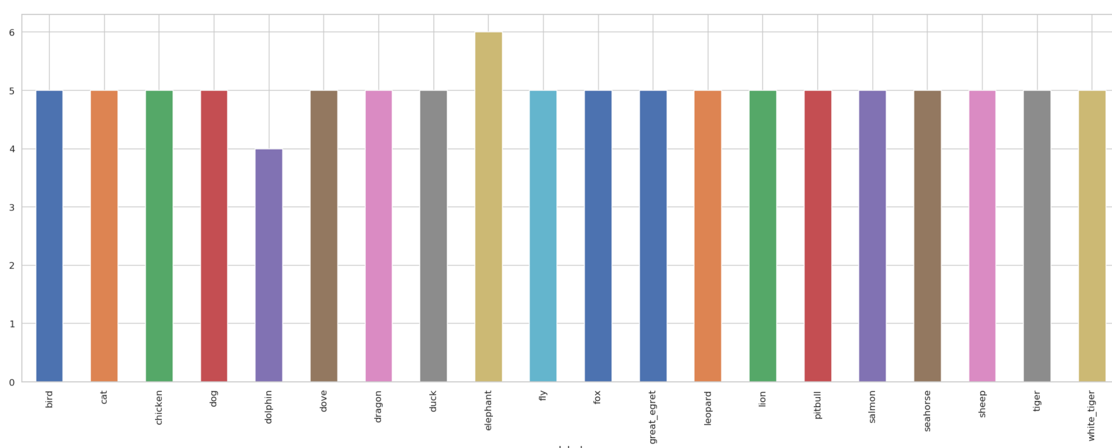
In [1040]:
```
#Outputting final results for train and test

print("FINAL TRAIN SCORE:", grid.best_estimator_.score(X,y))
y_pred = grid.best_estimator_.predict(X)
df_train['label'] = y_pred

final_train_results = df_train.copy()
final_train_results['label'] = final_train_results['label'].map(lambda x: mapping_i
final_train_results.to_csv('trainfile_output.txt',sep=' ',index=False,header=False)
final_train_results.groupby('label').size().plot(kind='bar')
plt.show()


print("FINAL TRAIN CLASSIFICATION:")
y_test = grid.best_estimator_.predict(X_test)
df_test['label'] = y_test
df_testf['label'] = y_test

final_test_results = df_test.copy()
final_test_results['label'] = final_test_results['label'].map(lambda x: mapping_inde
```

```
final_test_results.to_csv('testfile_output.txt',sep=' ',index=False,header=False)
final_test_results.groupby('label').size().plot(kind='bar')
plt.show()
```

FINAL TRAIN SCORE: 1.0



FINAL TRAIN CLASSIFICATION:



6 6. Communicating Results

- Discriminative Feature Correlation Plot
- 2D Representation of Classification Results for training and test
- plots to show model metrics across various classifiers

```
In [866]: results_df = pd.read_pickle('results_df.pkl')

In [867]: #Getting names of various components of the pipeline
          results_df['param_classify_name'] = results_df['param_classify'].map(lambda x: get_na
          results_df['param_reduce_dim_name'] = results_df['param_reduce_dim'].map(lambda x: ge
          results_df['param_scaler_name'] = results_df['param_scaler'].map(lambda x: get_name(
          results_df['param_reduce_dim__n_components'] = results_df['param_reduce_dim__n_compo
                  .fillna(results_df['param_reduce_dim__k'])
          results_df['param_preprocess_name'] = results_df[['param_scaler_name','param_reduce_
                          .apply(lambda x: '+'.join(x), axis=1)

In [971]: #Printing top results for each combination of scaler, reducer, and classifier
          results_df.groupby(['param_scaler_name',
                  'param_reduce_dim_name',
                  'param_classify_name'])['mean_test_Accuracy'].max().reset_index().sort_values
```

Out[971]:

	param_scaler_name	param_reduce_dim_name	param_classify_name
7	MinMaxScaler	SelectKBest	SVC
15	StandardScaler	SelectKBest	SVC
4	MinMaxScaler	SelectKBest	GaussianProcessClassifier
5	MinMaxScaler	SelectKBest	LinearSVC
12	StandardScaler	SelectKBest	GaussianProcessClassifier
13	StandardScaler	SelectKBest	LinearSVC
1	MinMaxScaler	PCA	LinearSVC
3	MinMaxScaler	PCA	SVC
9	StandardScaler	PCA	LinearSVC
0	MinMaxScaler	PCA	GaussianProcessClassifier
11	StandardScaler	PCA	SVC
6	MinMaxScaler	SelectKBest	RandomForestClassifier
14	StandardScaler	SelectKBest	RandomForestClassifier
10	StandardScaler	PCA	RandomForestClassifier
2	MinMaxScaler	PCA	RandomForestClassifier
8	StandardScaler	PCA	GaussianProcessClassifier

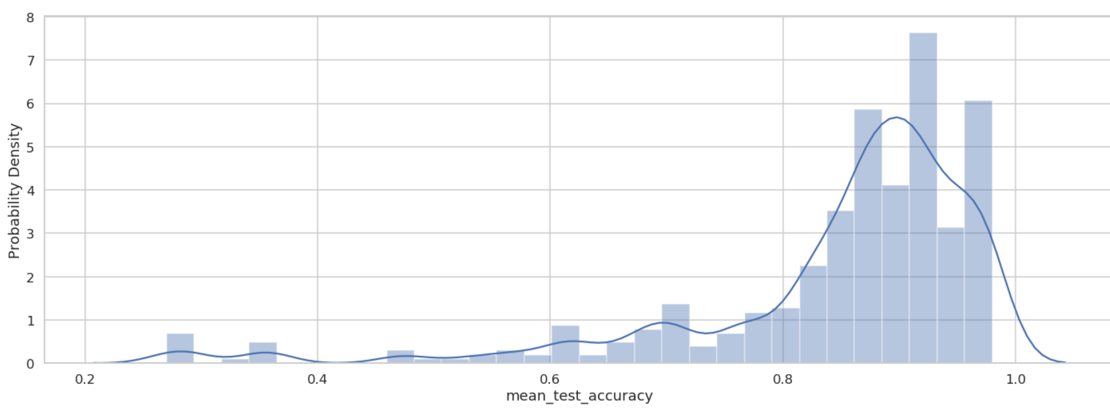
\

	mean_test_Accuracy
7	0.98
15	0.98
4	0.97
5	0.96
12	0.96
13	0.95
1	0.92
3	0.92
9	0.92
0	0.91
11	0.89
6	0.88
14	0.84

```
10                  0.82
2                   0.76
8                   0.47
```

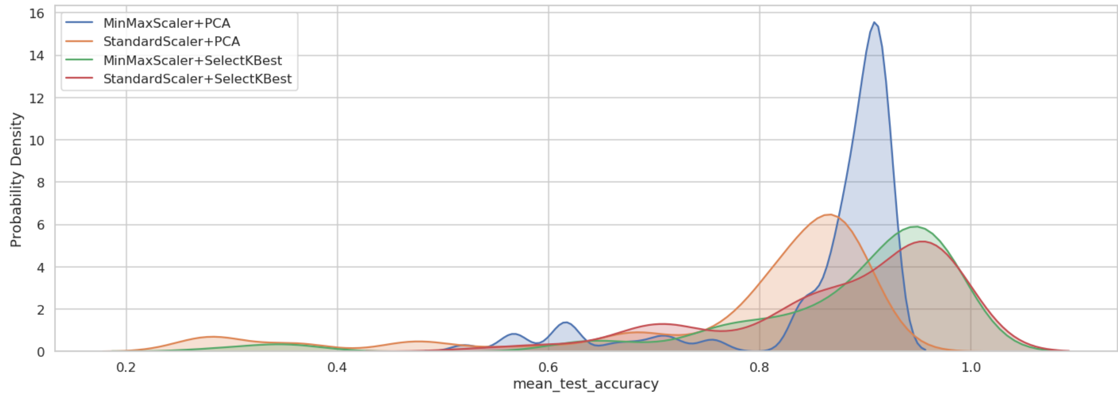
```
In [884]: #Accuracy Distribution
          sns.set_style('whitegrid')
          sns.despine(left=True)
          sns.set_context("talk")

          fig, ax = plt.subplots((1), figsize=(24,8))
          for metric in ['mean_test_Accuracy']:
              sns.distplot(results_df[metric],label=metric, ax=ax)

          ax.set_xlabel('mean_test_accuracy')
          ax.set_ylabel('Probability Density')

          plt.show()
```

```
<Figure size 1152x864 with 0 Axes>
```



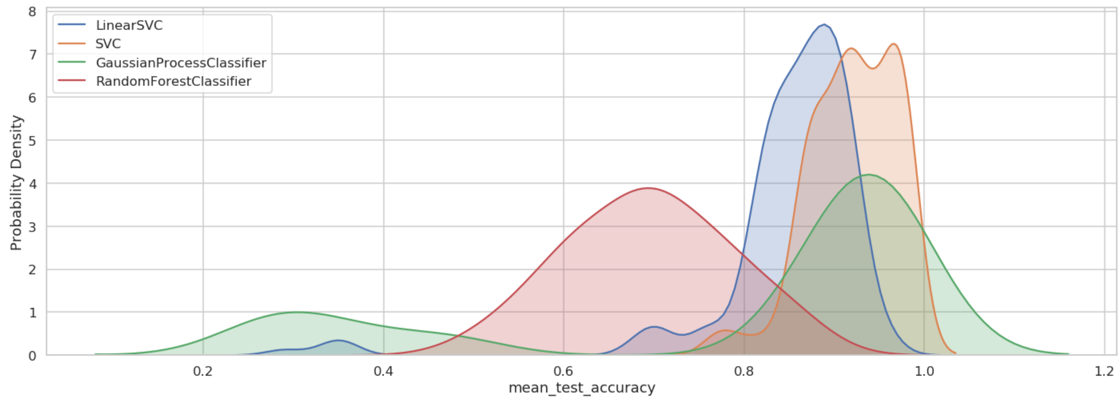

```
In [883]: #Accuracy Distribution for each PreProcesser
          fig, ax = plt.subplots((1), figsize=(24,8))
          for label in results_df['param_preprocess_name'].unique():
              print(label)
              mask = results_df['param_preprocess_name']==label
          #     sns.distplot(results_df[mask]['mean_test_Precision'], bins=5,
          #                  kde=True, rug=False, norm_hist=True,label=label, ax=ax)
              sns.kdeplot(results_df[mask]['mean_test_Accuracy'],shade=True,label=label, ax=ax

          ax.set_xlabel('mean_test_accuracy')
          ax.set_ylabel('Probability Density')
          plt.show()
```

```
MinMaxScaler+PCA
StandardScaler+PCA
MinMaxScaler+SelectKBest
StandardScaler+SelectKBest
```



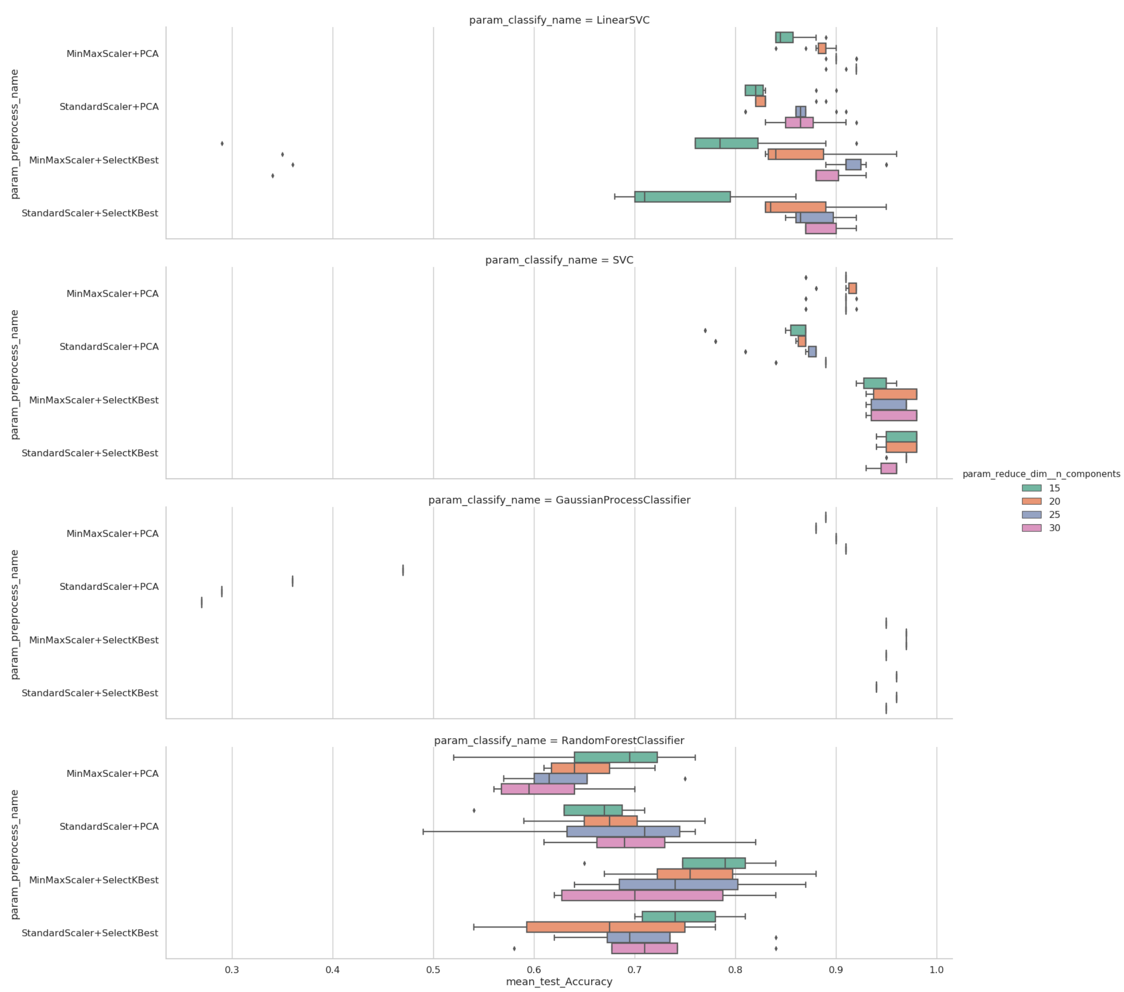

```
In [882]: #Accuracy Distributions for each classifier
          fig, ax = plt.subplots((1), figsize=(24,8))
          for label in results_df['param_classify_name'].unique():
              print(label)
              mask = results_df['param_classify_name']==label

              sns.kdeplot(results_df[mask]['mean_test_Accuracy'],shade=True,label=label, ax=ax)

          ax.set_xlabel('mean_test_accuracy')
          ax.set_ylabel('Probability Density')
          plt.show()
```

```
LinearSVC
SVC
GaussianProcessClassifier
RandomForestClassifier
```



```
In [881]: #Detailed Box Chart
          plt.rcParams['figure.figsize'] = [16,12]
          sns.set_style('whitegrid')
          sns.despine(left=True)
          sns.set_context("talk")

          g = sns.catplot(x="mean_test_Accuracy", y="param_preprocess_name", row="param_classi
          #                 col='param_scaler_name',param_preprocess_name
                          hue="param_reduce_dim__n_components",
                          kind="box", orient="h", height=6.0, aspect=4,
                          data=results_df,
                         palette="Set2",dodge=True)

          plt.show()
```

```
<Figure size 1152x864 with 0 Axes>
```



In [986]:
```
#Finding top discriminative features by the best classifier
scaler = grid.best_estimator_.named_steps.scaler
reduce_dim = grid.best_estimator_.named_steps.reduce_dim
X_train, y_train = X, y
X_val, y_val = X, y

X_train = scaler.transform(X_train)
X_train = reduce_dim.transform(X_train)

X_test = scaler.transform(X_test)
X_test = reduce_dim.transform(X_test)

print(X_train.shape,X_val.shape,X_test.shape)

print("COLUMNS SELECTED:")
final_feature_list = df_trainf.drop(columns=['label']).loc[:,reduce_dim.get_support(
final_feature_list
```

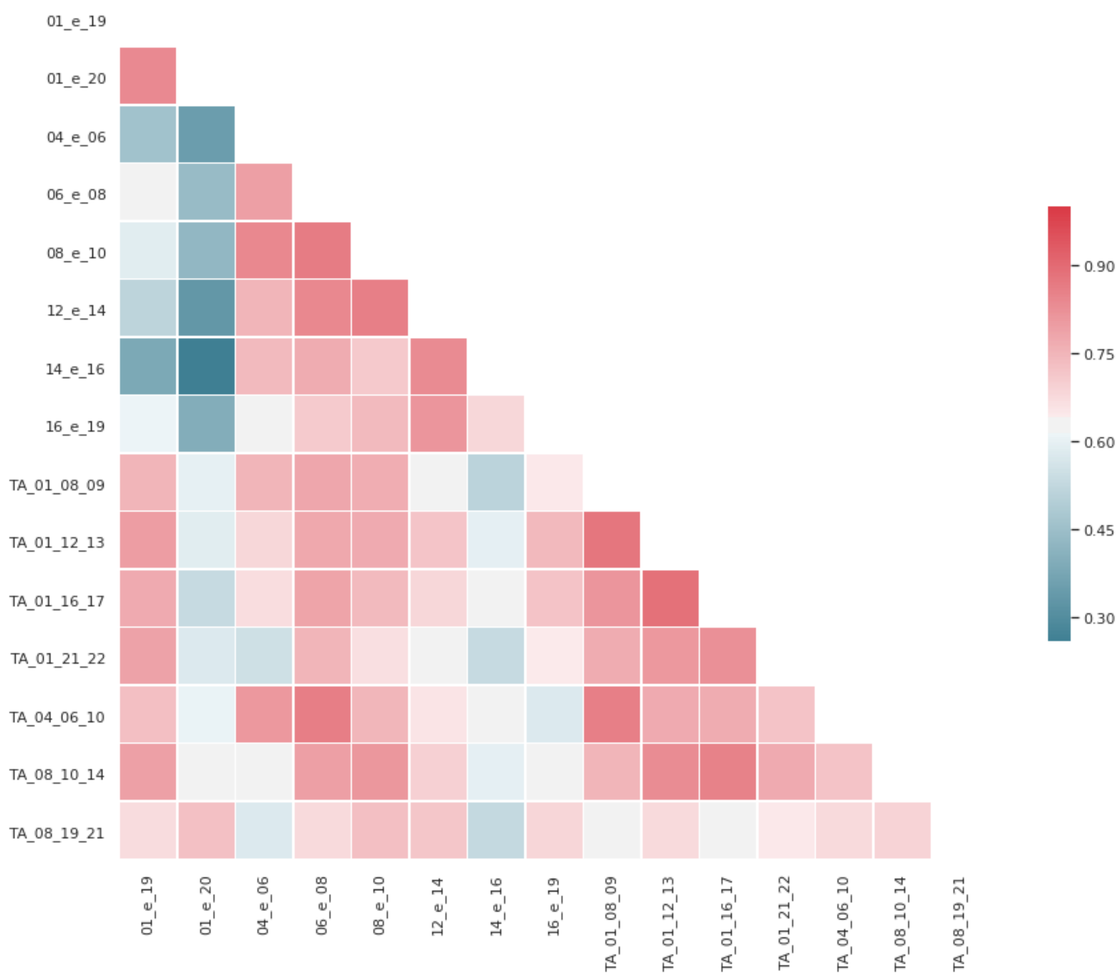
```
(100, 15) (100, 460) (100, 15)
COLUMNS SELECTED:
```

Out[986]:
```
['01_e_19',
 '01_e_20',
 '04_e_06',
 '06_e_08',
 '08_e_10',
 '12_e_14',
 '14_e_16',
 '16_e_19',
 'TA_01_08_09',
 'TA_01_12_13',
 'TA_01_16_17',
 'TA_01_21_22',
 'TA_04_06_10',
 'TA_08_10_14',
 'TA_08_19_21']
```

In [904]:
```
#Displaying correlation matrix of top features

sns.set(style="white")

# Compute the correlation matrix
# corr = df_trainf.drop(columns=['label']).loc[:,model.get_support()].corr()
corr = df_trainf[final_feature_list].corr()
```

```
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(16, 12))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1.0, vmin=corr.min().min(),
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
Out[904]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa5eb02e0f0>
```




```
In [825]: # bokeh basics library load
from bokeh.plotting import figure, output_notebook, show
```

```
from bokeh.models import ColumnDataSource, LabelSet, Label, LinearColorMapper, LogCol
from bokeh.models.tools import HoverTool
from bokeh.transform import dodge, factor_cmap, transform
from bokeh.palettes import PuBu,Spectral, Paired, Oranges, Greens, GnBu3, OrRd3, Purp
output_notebook()
```

In [1004]:
```
#Display clustered results

 from sklearn.manifold import Isomap

 def dt_scatter(source,title='Cluster Topic Plot',labelsFlag=False,csize=8):
     # Create a blank figure with labels
     p = figure(plot_width = 950, plot_height = 600,
                title = title,
                x_axis_label = 'X', y_axis_label = 'Y',
 #               y_axis_type="log", x_axis_type="log",
              tools=('pan, box_zoom, reset,save, wheel_zoom,hover')
               )

     colours = list((Category20c[source['label'].astype(int).max()+1]))
     source['color'] = source['label'].apply(lambda x: colours[int(x)])

     p.circle(x='x', y='y',
              source=source[source['type']=='TRAIN'],
              size=csize,
              color='color',
             legend='labelName',
             alpha=0.8)

     p.square(x='x', y='y',
              source=source[source['type']=='TEST'],
              size=csize,
              color='color',
 #             legend='labelName',
             alpha=0.8)

     if labelsFlag:
         labels = LabelSet(x='x', y='y', text='labelName',
                       x_offset=2, y_offset=2, source=ColumnDataSource(source),
                          text_font_size='7pt')


         p.add_layout(labels)


     p.hover.tooltips = [(c,'@'+c) for c in source.columns]
```

```
    # configure visual elements of the plot
    p.title.text_font_size = '10pt'
    p.xaxis.visible = False
    p.yaxis.visible = False
    p.legend.label_text_font_size = "5pt"
    p.legend.location = "top_right"
    p.legend.background_fill_alpha = 0.8

    p.legend.spacing = 0
    p.legend.padding = 0
    p.legend.margin = 0
#     p.grid.grid_line_color = None
#     p.outline_line_color = None

#     # Show the plot
    show(p)
    return

df_trainf['labelName'] = df_trainf['label'].map(lambda x: mapping_index[x])
df_testf['labelName'] = df_testf['label'].map(lambda x: mapping_index[x])
df_trainf['color'] = Paired[4][0]
df_testf['color'] = Paired[4][-1]

df_trainf['type'] = 'TRAIN'
df_testf['type'] = 'TEST'

displaydf = pd.concat([df_trainf, df_testf], ignore_index=True).reset_index()

displaydf.loc[:df_trainf.shape[0]-1,final_feature_list] = X_train
displaydf.loc[df_trainf.shape[0]:,final_feature_list] = X_test

#Reduce feature set into 2D
n_iter=1000
red_model = TSNE(n_components=2, n_iter=n_iter, learning_rate=20)

r_matrix = red_model.fit_transform(displaydf[final_feature_list].values)
displaydf['x'] = r_matrix[:,0]
displaydf['y'] = r_matrix[:,1]

#Plot scatter plot
dt_scatter(displaydf[['label','labelName','x','y','type','color'] + final_feature_li
           csize=15,
          labelsFlag=True)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  del sys.path[0]
```

```
In [1031]: # Display each variables profile across classes
           df_scaled_train = pd.DataFrame(X_train, columns=final_feature_list)

           df_scaled_train['labelName'] = df_trainf['labelName']

           plotdf = pd.melt(df_scaled_train, id_vars=['labelName'], value_vars=final_feature_li

           plt.rcParams['figure.figsize'] = [35,12]
           sns.set_style('whitegrid')
           sns.despine(left=True)
           sns.set_context("poster")

           clr_pal = sns.color_palette("muted", n_colors=10) + sns.color_palette("colorblind",

           g = sns.catplot(x="variable", y="value", hue="labelName",
           #                 col='param_scaler_name',param_preprocess_name
           #                 hue="variable",
                            row='labelName',
                            kind="point", orient="v", height=5.0, aspect=3,
                            data=plotdf,
                           palette=clr_pal,dodge=False,
                            ci='sd',join=True,errwidth=1.0,
           #                  cut=0
                           )

           plt.show()
```

```
<Figure size 2520x864 with 0 Axes>
```


variable = 01_e_19
variable = 01_e_20
variable = 04_e_06
variable = 06_e_08
variable = 08_e_10
variable = 12_e_14
variable = 14_e_16
variable = 16_e_19
variable = TA_01_08_09
variable = TA_01_12_13
variable = TA_01_16_17
variable = TA_01_21_22
variable = TA_04_06_10
variable = TA_08_10_14
variable = TA_08_19_21
value
bird cat chicken dog dolphin dove dragon duck elephant fly fox great_egret leopard lion pitbull salmon seahorse sheep tiger white_tiger
labelName