Machine Learning

April 3, 2017

Notebook for training and assessing model

- various samples for Liner Regression and SVC
- confusion matrix
- Network graphs

```
In [ ]: import axelrod as axl
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import networkx as nx
        from MachineLearning import *
        import plotly
        import plotly.plotly as py
        from plotly.graph_objs import *
```

```
In [ ]: axelrod_strategies = axl.strategies
        training_df = pd.read_csv('large_training_data.csv', index_col=0)
        compute_sample_scores(3, 3, training_df, axelrod_strategies)
```

```
In [ ]: # this takes a long time
        # sample_scores = [compute_sample_scores(i, 50, training_df, axelrod_strate
```

```
In [ ]: zipped_scores = [list(zip(*k)) for k in sample_scores[2:]]
        LR_sample_scores = [[0], [0]] + [i[0] for i in zipped_scores]
        SVC_sample_scores = [[0], [0]] + [i[1] for i in zipped_scores]
```

```
In [ ]: plt.figure(figsize=(30, 10))
        lr_ps = [i for i in range(50)]
        sns.set_style("darkgrid", {'axes.grid' : True})
        lr_plt = plt.violinplot(LR_sample_scores, positions=lr_ps, widths=0.9)
        plt.tick_params(axis='both', which='major', labelsize=20)
        # plt.title('Violin Plot of Model Score against Number of Strategies used i
        plt.xlabel('Number of Strategies used in Sample', fontsize=20)
        plt.ylabel('Model Score', fontsize=20)
        plt.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/ML/score_
        plt.show()
        plt.clf()
```

```
In [ ]: plt.figure(figsize=(30, 10))
        svc_ps = [i for i in range(50)]
        sns.set_style("darkgrid", {'axes.grid' : True})
        svc_plt = plt.violinplot(SVC_sample_scores, positions=svc_ps, widths=0.9)
        plt.tick_params(axis='both', which='major', labelsize=20)
        # plt.title('Violin Plot of Model Score against Number of Strategies used i
        plt.xlabel('Number of Strategies used in Sample', fontsize=20)
        plt.ylabel('Model Score', fontsize=20)
        plt.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/ML/score_
        plt.show()
        plt.clf()
```

```
In [ ]: # create dataframe with no dubplicates, only one row for each strategy pai
        unique_scoring_df = training_df.groupby(['Name_A', 'Name_B']).first()
        unique_scoring_df.head(10)
```

```
In [ ]: # create models using specified strategies to see how the confusion matrice
        results_df = pd.read_csv('std_summary.csv')
        ordered_strats = results_df.Name.values
        model_strats = ordered_strats[::5] # every 5th strategy when order by rank
        model_train_df, model_score_df = split_dataframe(model_strats, training_df)
        lr_model, svc_model = create_models_for_sample(model_train_df)
```

```
In [ ]: scoring_equivalent = unique_scoring_df['Equivalent']
        scoring_data = unique_scoring_df.copy()
        scoring_data.drop('Equivalent', axis=1, inplace=True)
        svc_predictions = svc_model.predict(scoring_data)
        svc_c_matrix = confusion_matrix(scoring_equivalent, svc_predictions)
        np.set_printoptions(precision=2)
        class_names = ['Different', 'Same']
        sns.set_style("whitegrid", {'axes.grid' : False})
        plt.figure()
        plot_confusion_matrix(svc_c_matrix, classes=class_names, title='')
        plt.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/ML/confus
        plt.show()
        plt.clf()
        print(svc_model.score(X=scoring_data, y=scoring_equivalent))
```

```
In [ ]: from sklearn.feature_selection import chi2
        scores, pvalues =chi2(X=scoring_data, y=scoring_equivalent)
        p_vals = list(zip(scoring_data.columns, pvalues))
        pval_df = pd.DataFrame(data=p_vals, columns=['Variable', 'p Value'])
        pval_df
        p = pval_df.to_latex(index=False)
        with open("/Users/James/Projects/FinalYearReport-Manuscript/img/p-values.te
            text_file.write(p)
```

```
In [ ]: # collect rows where the model thinks the strategies are the same
        actual_model_predictions = svc_predictions
        equivalent_df = unique_scoring_df.copy().reset_index()
        equivalent_df['Predictions'] = actual_model_predictions
        equivalent_df = equivalent_df[equivalent_df['Predictions'] == 1]
        equivalent_names = equivalent_df[['Name_A', 'Name_B']]
        equivalent_names.head()

In [ ]: # create adjacency matrix for strategies that are equivalent
        adj_df = pd.crosstab(equivalent_names.Name_A, equivalent_names.Name_B)
        idx = adj_df.columns.union(adj_df.index)
        adj_df = adj_df.reindex(index = idx, columns=idx, fill_value=0)
        adj_df.head()

In [ ]: plt.figure(figsize=(50, 50))
        sns.heatmap(adj_df, cbar=False)
        plt.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/ML/simila
        plt.show()

In [ ]: G=nx.from_pandas_dataframe(equivalent_names, 'Name_A', 'Name_B', True)
        UG = G.to_undirected()
        sub_graphs = nx.connected_component_subgraphs(UG)

In [ ]: plotly.offline.init_notebook_mode()
        py.sign_in('theref', '5zi1ecSf80pI5u7I5rP0')

In [ ]: def scatter_nodes(pos, labels=None, color=None, size=20, opacity=0.5):
            # pos is the dict of node positions
            # labels is a list  of labels of len(pos), to be displayed when hoverin
            # color is the color for nodes. When it is set as None the Plotly defau
            # size is the size of the dots representing the nodes
            #opacity is a value between [0,1] defining the node color opacity
            trace = Scatter(x=[], y=[],  mode='markers', marker=Marker(size=[]))
            for k, v in pos.items():
                trace['x'].append(pos[k][0])
                trace['y'].append(pos[k][1])
            attrib=dict(name='', text=labels , hoverinfo='text', opacity=opacity) #
            trace=dict(trace, **attrib)# concatenate the dict trace and attrib
            trace['marker']['size']=size
            return trace


        def scatter_edges(G, pos, line_color=None, line_width=1):
            trace = Scatter(x=[], y=[], mode='lines')
            for edge in G.edges():
                trace['x'] += [pos[edge[0]][0],pos[edge[1]][0], None]
                trace['y'] += [pos[edge[0]][1],pos[edge[1]][1], None]
                trace['hoverinfo']='none'
                trace['line']['width']=line_width
                if line_color is not None: # when it is None a default Plotly colo
                    trace['line']['color']=line_color
```

```
    return trace

def make_annotations(pos, font_size=14, font_color='rgb(25,25,25)'):
    annotations = Annotations()
    for k, v in pos.items():
        annotations.append(
            Annotation(
                text=str(k),
                x=pos[k][0], y=pos[k][1],
                xref='x1', yref='y1',
                font=dict(color= font_color, size=font_size),
                showarrow=False)
        )
    return annotations

G=nx.from_pandas_dataframe(equivalent_names, 'Name_A', 'Name_B', True)
sixth = len(G.nodes())//6
inner, middle, outer = G.nodes()[:sixth], G.nodes()[sixth:3*sixth], G.nodes
pos=nx.shell_layout(G, nlist=[inner, middle, outer], scale=10)

labels=[str(k) for k in range(len(pos))] # labels are  set as being the nod
trace1=scatter_edges(G, pos)
trace2=scatter_nodes(pos, labels=labels)

width=2000
height=2500
axis=dict(showline=False, # hide axis line, grid, ticklabels and  title
          zeroline=False,
          showgrid=False,
          showticklabels=False,
          title=''
          )
layout=Layout(
    font= Font(),
    showlegend=False,
    autosize=False,
    width=width,
    height=height,
    xaxis=XAxis(axis),
    yaxis=YAxis(axis),
    margin=Margin(
        l=40,
        r=40,
        b=85,
        t=100,
        pad=0,

    ),
    hovermode='closest',
#     plot_bgcolor='#EFECEA', #set background color
    )


data=Data([trace1, trace2])

fig = Figure(data=data, layout=layout)
fig['layout'].update(annotations=make_annotations(pos))

py.image.ishow(fig)
# py.image.save_as(fig, filename='/Users/James/Projects/FinalYearReport-Mar
```

In [ ]:

```
UG = G.to_undirected()
sub_graphs = nx.connected_component_subgraphs(UG)

for index, sg in enumerate(sub_graphs):
    if nx.number_of_nodes(sg) < 2:
        continue # ignore this graph and move onto the next one
    pos=nx.spring_layout(sg)
    labels=[str(k) for k in range(len(pos))] # labels are  set as being the
    trace1=scatter_edges(sg, pos)
    trace2=scatter_nodes(pos, labels=labels)
    width=500
    height=500
    data=Data([trace1, trace2])
    layout=Layout(
        font= Font(),
        showlegend=False,
        autosize=False,
        width=width,
        height=height,
        xaxis=XAxis(axis),
        yaxis=YAxis(axis),
        margin=Margin(
            l=40,
            r=40,
            b=85,
            t=100,
            pad=0,

        ),
    hovermode='closest',
#     plot_bgcolor='#EFECEA', #set background color
    )
    fig = Figure(data=data, layout=layout)
    fig['layout'].update(annotations=make_annotations(pos))
```

```
py.image.ishow(fig)      py.image.save_as(fig, filename='/Users/James/F
```