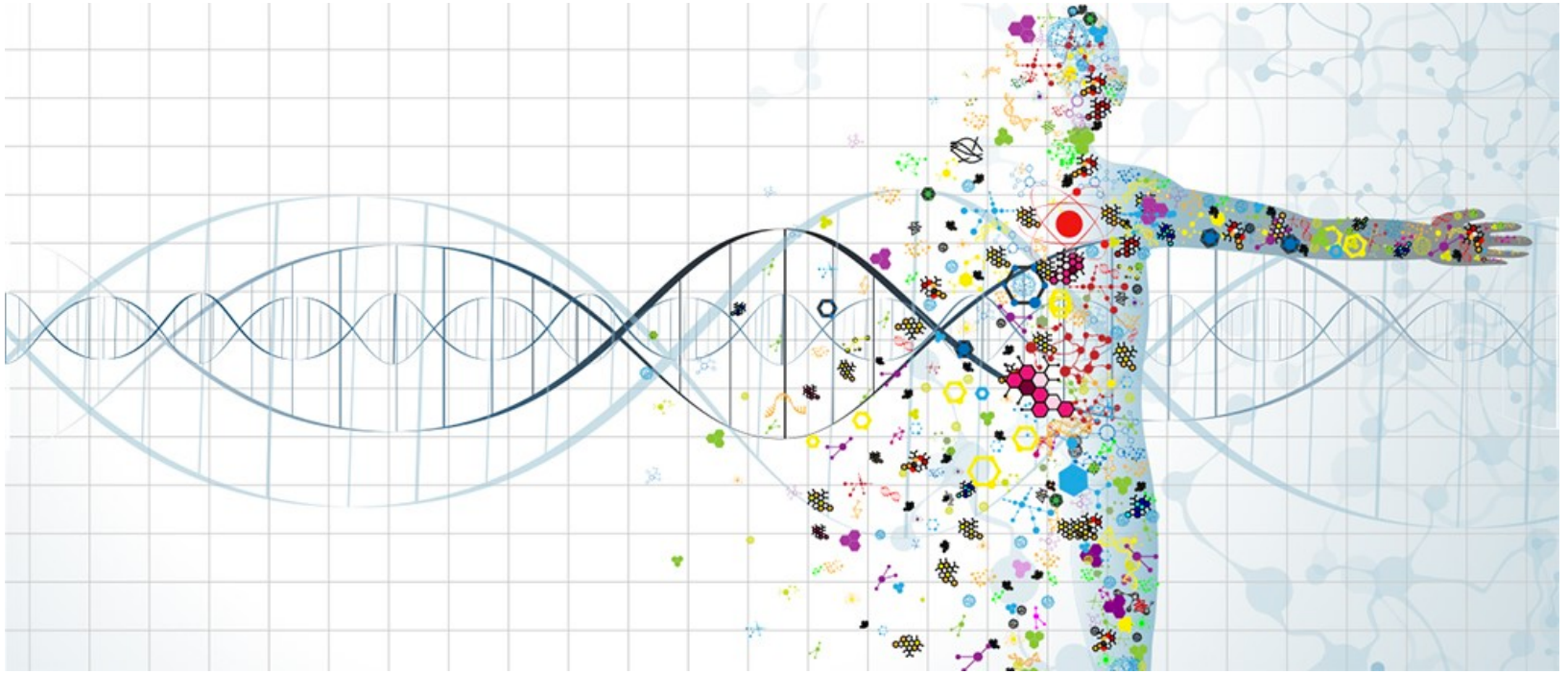
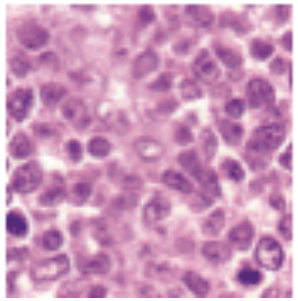
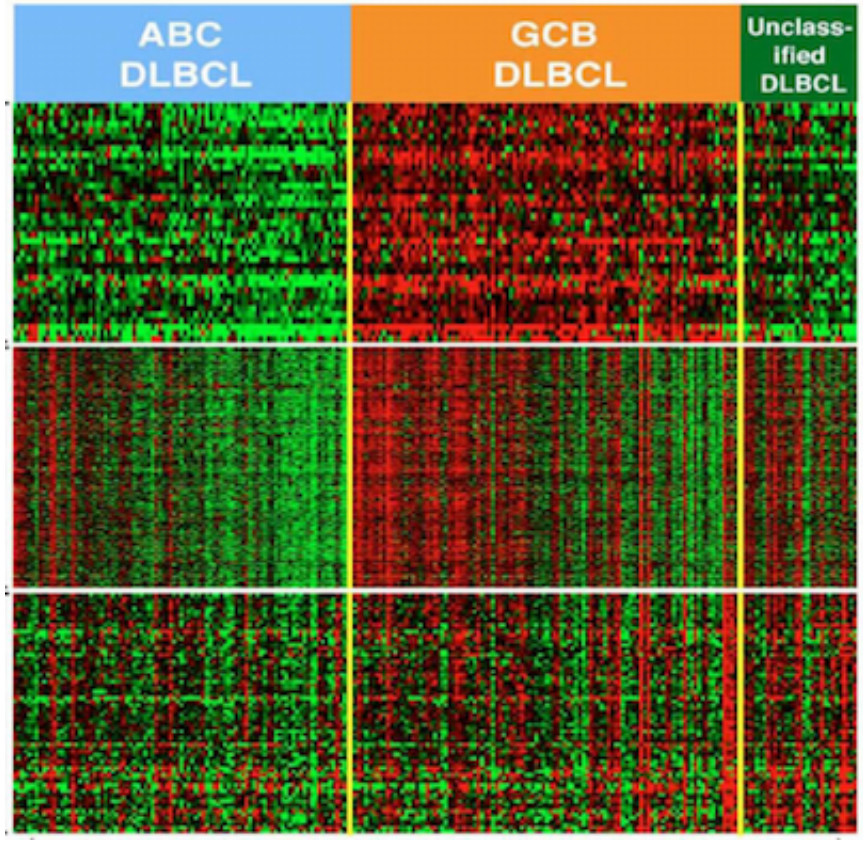
Predicting Lymphoma Subtype from Patient Chromosome Copy Number Data



Diffuse Large B-Cell Lymphoma (DLBCL)

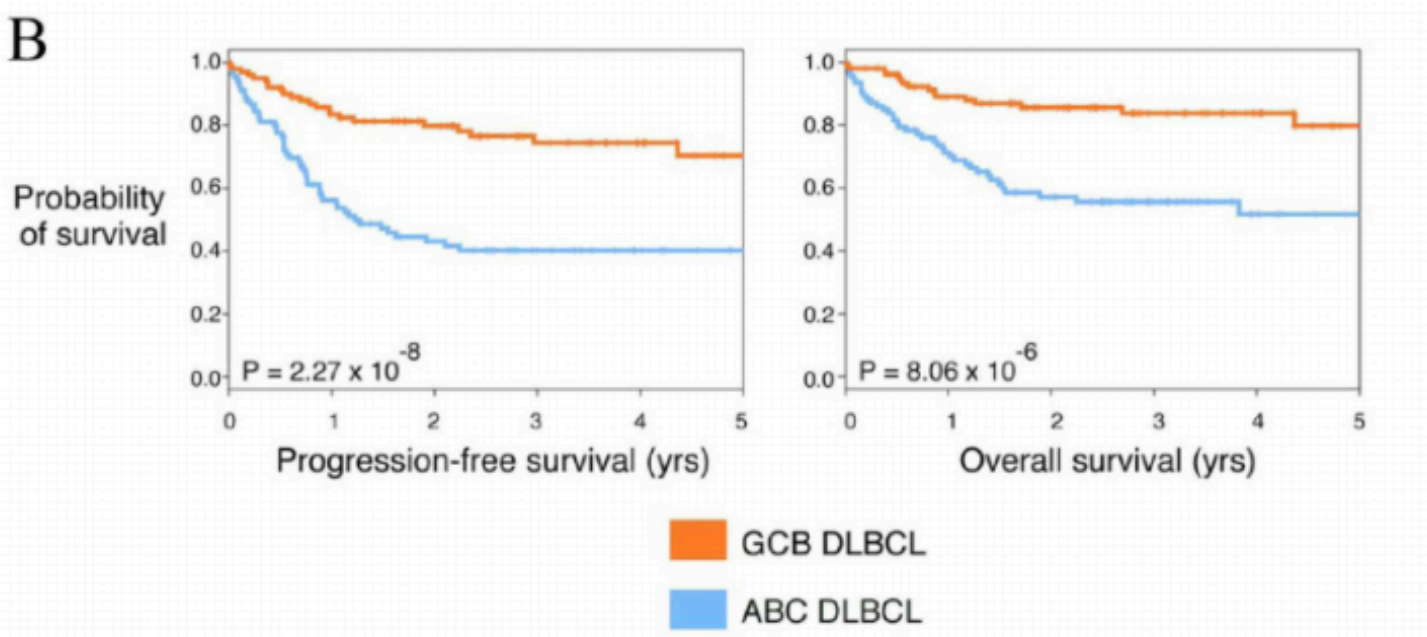
DLBCL





We currently divide DLBCL into three major subtypes:
ABC, GCB, and Unclassified.

The cells from these lymphomas look identical under the microscope, but have unique genetic signatures, respond differently to drugs, and the patients have different survival outcomes.



Goal

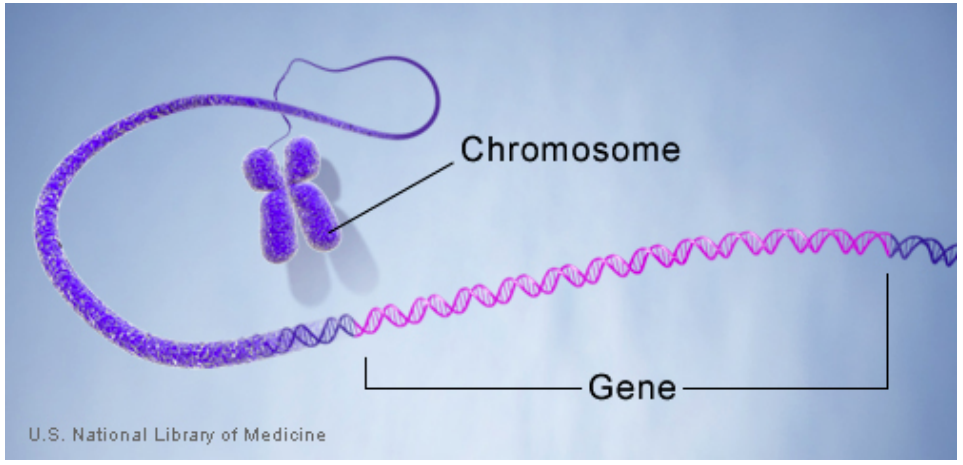
Use chromosome copy number data to predict DLBCL subtype

- Features = copy number data
- Target = DLBCL subtype

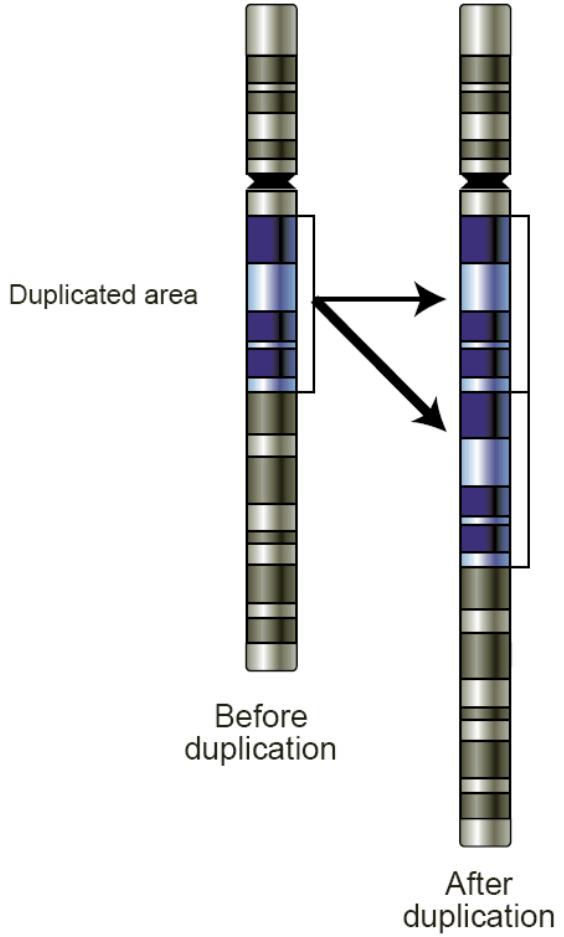
Example of data from one patient

539 Patients

Each patient has their own file



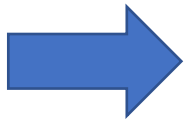
	Chromosome	Start	Stop	Copy Number
2	chr1	51723	149236241	0
3	chr1	149236241	149251876	2
4	chr1	149251876	249234376	0
5	chr2	2772	243099444	0
6	chr3	50333	197906005	0
7	chr4	2269	144920433	0
8	chr4	144920433	145044529	2
9	chr4	145044529	161880120	0
10	chr4	161880120	161884742	-1
11	chr4	161884742	183319181	0
12	chr4	183319181	183399086	-1
13	chr4	183399086	191030138	0
14	chr5	5520	158300100	1
15	chr5	158300100	158446952	-1
16	chr5	158446952	180729790	1
17	chr6	139649	23743214	0
18	chr6	23743214	23746575	-2
19	chr6	23746575	170992522	0
20	chr7	33259	109437087	0.5
21	chr7	109437087	109453934	-1.5
22	chr7	109453934	126046316	0.5
23	chr7	126046316	126050494	-2
24	chr7	126050494	159129708	0.5
25	chr8	36386	7236424	0
26	chr8	7240368	7787836	1
27	chr8	7787836	11009995	0
28	chr8	11009995	11011334	-2
29	chr8	11011334	12237105	0
30	chr8	12237105	12403096	2



Macroscopic look at the data

```
#Read in all .txt patient chromosome copy number files into
for filename in os.listdir('.'):
    if filename.endswith('.txt'):
        filename_df = pd.read_table(filename,\
        names = ['chr','start','stop','copynumber'], index_c
        filename_df.drop(['start','stop'], axis=1, inplace=T
        filename_df = filename_df.groupby('chr').sum()
        filename_df = filename_df.T
        filename_df['Sample'] = re.sub('\.txt$', '' , str(fi
        filename_df.set_index('Sample', inplace=True)
        df1 = df1.append(filename_df)
        df1 = df1.iloc[:,0:24]

#Merge another data frame to include the DLBCL Subtype in th
df2 = pd.read_excel('DLBCL_Subtype.xlsx')
df2.set_index('Sample', inplace = True)
df1 = df1.join(df2, how='left')
df1.reset_index(inplace=True)
del df1['Sample']
df1.dropna(inplace=True)
df1['DLBCL Subtype Map'] = df1['DLBCL Subtype']\
    .map({'ABC' : 1, 'GCB' : 0, 'Unclass' : 2})
```

+ 539 patient files



	chr1	chr10	chr11	chr12	chr13	chr14	chr15	chr16	chr17	chr18
0	2.0	0.0	0.0	0.0	0.0	1.0	-2.0	-2.0	2.0	0.0
1	-4.0	1.0	0.0	-2.0	-4.0	-2.0	-2.0	0.0	3.0	3.0
2	-1.0	-1.0	-3.0	2.0	-2.0	-5.0	-3.0	-2.0	-3.0	3.0
3	-6.0	-2.0	0.0	-2.0	-1.0	1.0	-2.0	-2.0	-2.0	6.0
4	-3.0	-1.0	-5.0	2.0	-2.0	-4.5	-6.0	-1.0	5.0	15.0

	...	chr4	chr5	chr6	chr7	chr8	chr9	chrX	chrY
0	...	0.0	7.0	-3.0	0.0	0.0	-2.0	4.0	0.0
1	...	-6.0	0.0	1.0	-1.0	3.0	-3.0	-1.0	0.0
2	...	-1.0	2.0	-10.0	1.0	-3.0	-2.0	5.0	1.0
3	...	-5.0	-1.0	-5.0	6.0	0.0	-2.0	1.0	-1.0
4	...	-2.0	2.0	-5.0	-6.0	-2.0	-4.0	6.0	-2.0

	DLBCL Subtype	DLBCL Subtype Map
0	ABC	1
1	ABC	1
2	ABC	1
3	ABC	1
4	ABC	1

[5 rows x 26 columns]
(539, 26)

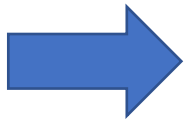
Final output:
539 rows, 26 columns

Microscopic look at the data

```
df2 = pd.DataFrame()

for filename in os.listdir('.'):
    if filename.endswith('.txt'):
        filename_df = pd.read_table(filename, names =
            ['chr','start','stop','copy_number'], hea

        filename_df['start'] = filename_df['start'].a
        filename_df['stop'] = filename_df['stop'].ast

        filename_df['copynumber_master'] = filename_d
            filename_df['start'] + '_' + filename_df[

        filename_df.drop(['start','stop', 'chr'], axi
        filename_df = filename_df.groupby('copynumber
        filename_df = filename_df.T
        filename_df['Sample'] = re.sub('\.txt$', '' ,
        filename_df.set_index('Sample', inplace = Tru
        df2 = df2.append(filename_df)
        df2.fillna(0, inplace = True)
```

```
df = pd.read_excel('DLBCL_Subtype.xlsx')
df.set_index('Sample', inplace=True)
df2 = df2.join(df, how = 'left')
df2.reset_index(inplace=True)
del df2['Sample']
df2.dropna(inplace=True)
```

+ 539 patient files



	chrY_9390879_28809923	chrY_9432191_9443799	chrY_9439059_9441690
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	chrY_9441690_22222983	chrY_9441690_28809923	chrY_9443799_9787623
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	chrY_9758611_13934470	chrY_9774900_22066345	chrY_9787623_28809923
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

Final output:
539 rows, 54,620 columns

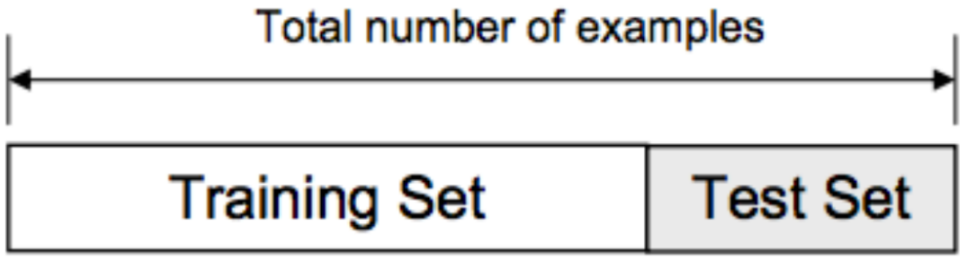
Models

```
ABC         278
GCB         154
Unclass     107
Name: DLBCL Subtype, dtype: int64

The overall null accuracy rate for the whole data set is: 52%
```

Use train, test split to narrow down the best model using accuracy score

- Logistic regression
- K-nearest neighbors
- Random forests

Double check with cross-validation



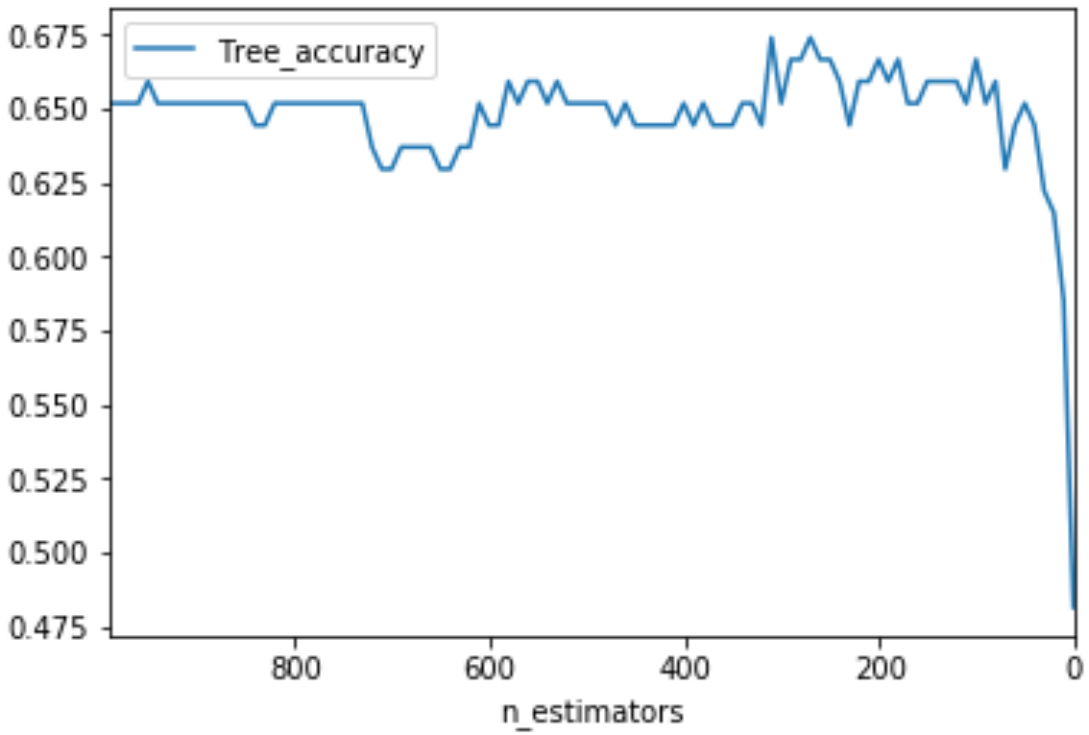
Apply bagging and boosting

Macroscopic view (24 features)

Random forest is best model

Randomized and grid searches for optimal number of trees and depth

```
from sklearn.ensemble import RandomForestClassifier

depth_range = range(1,1000,10)

tree_accuracy_list = []

def tree_score(depth_range):
    for i in depth_range:
        treeclass = RandomForestClassifier(n_estimators
        X_train, X_test, y_train, y_test = train_test_sp
        treeclass.fit(X_train, y_train)
        y_pred = treeclass.predict(X_test)
        tree_accuracy = accuracy_score(y_test, y_pred)
        tree_accuracy_list.append(tree_accuracy)

tree_score(depth_range)
tree_dict = {'n_estimators': depth_range, 'Tree_accuracy
df_tree = pd.DataFrame(tree_dict).set_index('n_estimator

df_tree.plot(y='Tree_accuracy')

df_tree.sort_values('Tree_accuracy', ascending = False).
```



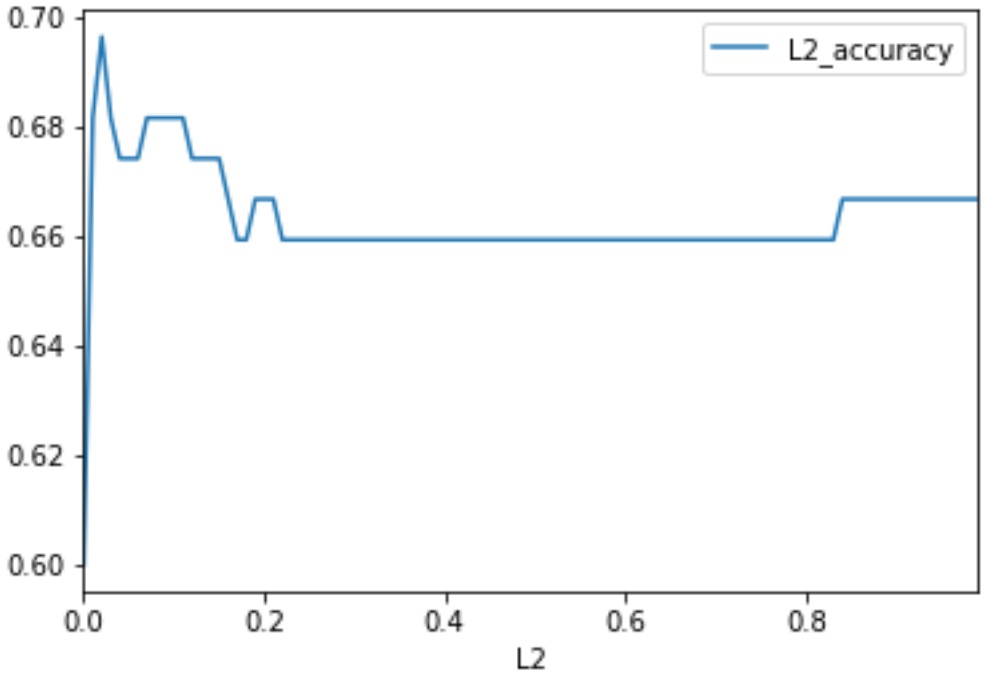
~65-67% for highest accuracy scores

Microscopic view (54,620 features)

Logistic regression is best model

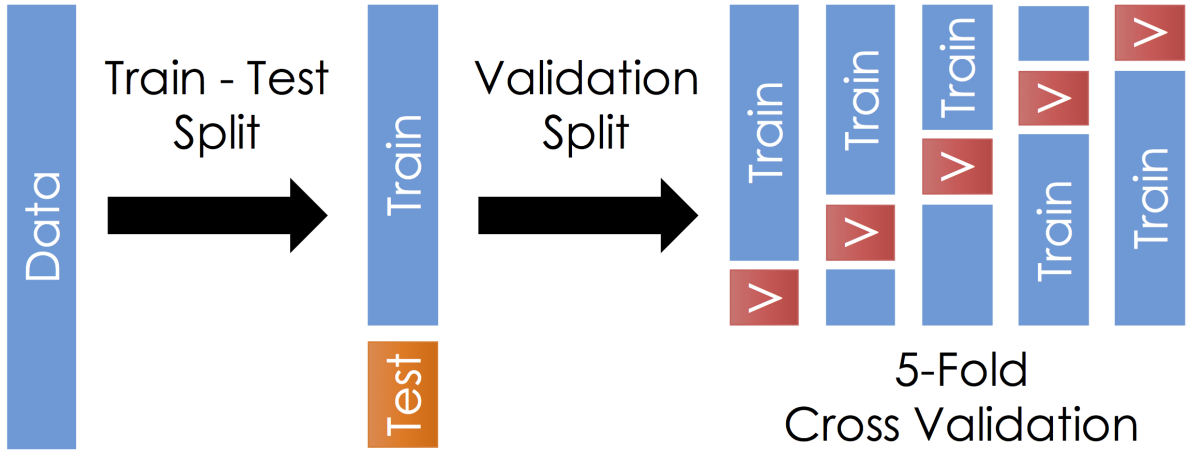
Randomized and grid searches for optimal number regularization method and “C” value

```
L2_range = np.arange(.0001, 1, 0.01)

L2_accuracy_list = []

def L2_logreg_accuracy(L2_range):
    for i in L2_range:
        logreg = LogisticRegression(C = i, random_state = 42)
        logreg.fit(X_train, y_train)
        y_pred = logreg.predict(X_test)
        L2_accuracy = accuracy_score(y_test, y_pred)
        L2_accuracy_list.append(L2_accuracy)

L2_logreg_accuracy(L2_range)
L2_dict = {'L2' : L2_range, 'L2_accuracy' : L2_accuracy_list}
L2_df = pd.DataFrame(L2_dict).set_index('L2')
L2_df.plot(y = 'L2_accuracy')
L2_df.sort_values('L2_accuracy', ascending = False).head()
```



~66-68% for highest accuracy scores

Cross-validation for both forms of data



Macroscopic view of data

```
from sklearn.model_selection import RandomizedSearch

clf = RandomForestClassifier()

depth_range = range(1, 100, 5)
n_range = range(1, 1000, 10)

param_dist = dict(n_estimators = n_range, max_depth

rand = RandomizedSearchCV(clf, param_dist, cv = 5, s
                          n_iter = 100, random_sta

rand.fit(X, y)

print(rand.best_score_)
rand.best_params_
```

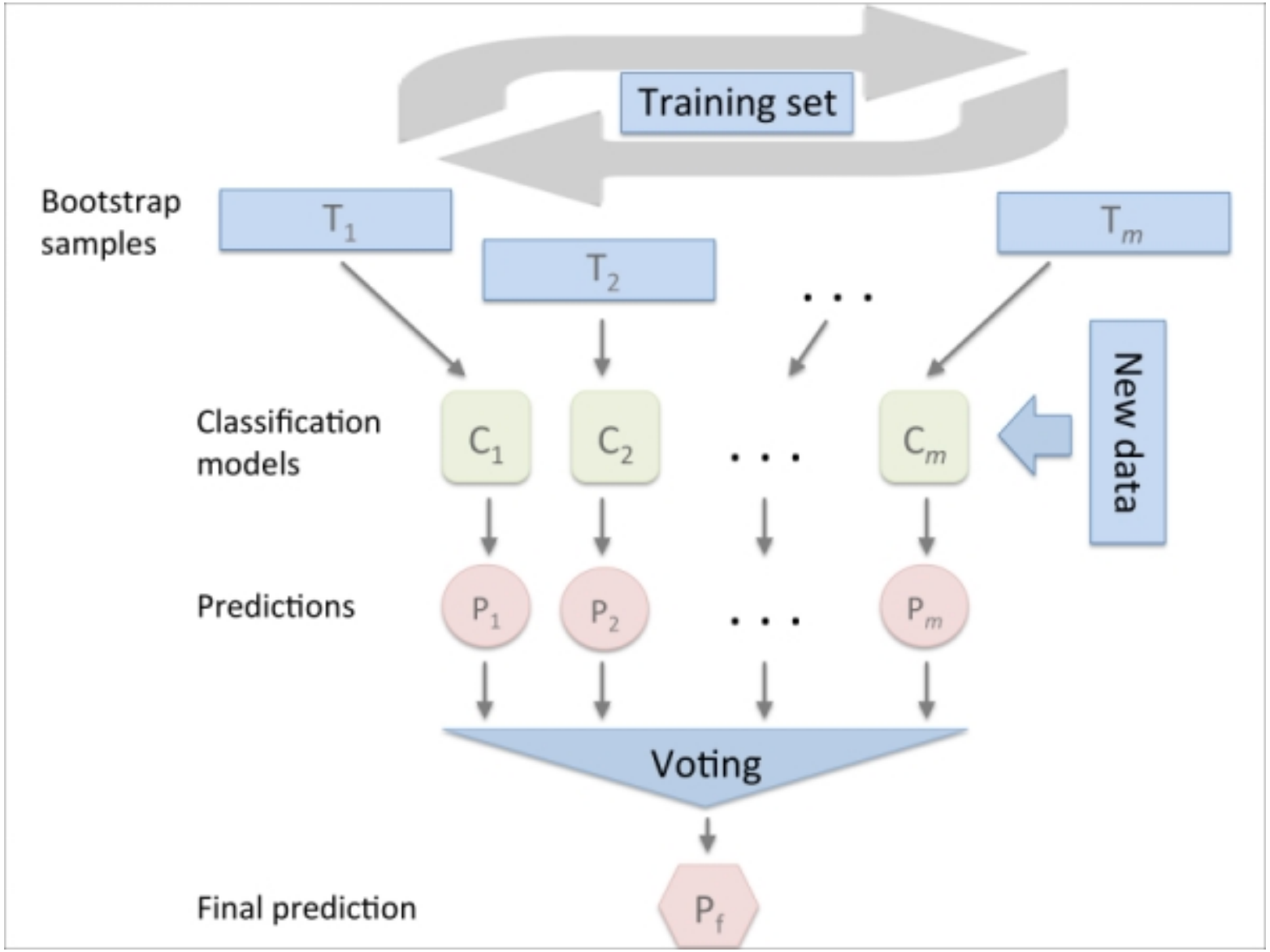
```
0.643784786642

{'max_depth': 66, 'n_estimators': 231}
```

Microscopic view of data

```
from sklearn.model_selection import RandomizedSearc

clf = RandomForestClassifier()

depth_range = range(1, 250, 5)
n_range = range(1, 1000, 10)

param_dist = dict(n_estimators = n_range, max_depth

rand = RandomizedSearchCV(clf, param_dist, cv = 5,
                          n_iter = 50, random_sta

rand.fit(X, y)

print(rand.best_score_)
rand.best_params_
```

```
0.641929499072

{'max_depth': 221, 'n_estimators': 261}
```

Bagging


Training set
Bootstrap samples
T_1
T_2
T_m
Classification models
C_1
C_2
C_m
New data
Predictions
P_1
P_2
P_m
Voting
Final prediction
P_f

Bagging

Macroscopic view of data
(random forest bagging)

```
from sklearn.ensemble import Ran
X_train, X_test, y_train, y_test
clf = RandomForestClassifier(max
bag = BaggingClassifier(base_est
bag.fit(X_train, y_train)
y_pred = bag.predict(X_test)
accuracy_score(y_pred, y_test)
```

```
.6592592592592921
```

Microscopic view of data
(logistic regression bagging)

```
X_train, X_test, y_train, y_test = train_t
logreg = LogisticRegression()
bag = BaggingClassifier(base_estimator = l
bag.fit(X_train, y_train)
y_pred = bag.predict(X_test)
accuracy_score(y_pred, y_test)
```

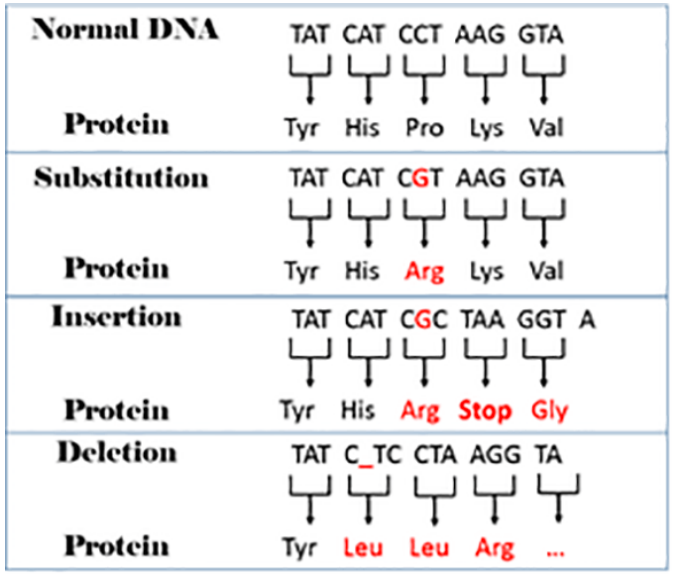
```
0.6888888888888888
```

Bagging does not really improve accuracy score
Gradient boosting and Adaboost also did not improve accuracy score

Conclusion

- Can not seem to beat ~65% accuracy with any models

- DLBCL subtype must be explained by more than just copy number or we need more samples/accurate data
  - DNA mutations
  - Epigenetics



Future Directions

- Look at more than accuracy score for model evaluation
- Include data on other types of DNA mutations
- Try other types of classification models