ML Project Notebook report

1. Understanding and plotting the data

In the first part of this exercise we will use `pandas` DataFrames to store and manipulate the data and use `seaborn` to produce nice visualizations of the data.

The point of this part is to handle the data to get to know the dataset better.

The Data ProQDock.csv is taken from the following piece of literature: https://academic.oup.com/bioinformatics/article/32/12/i262/2288786 (https://academic.oup.com/bioinformatics/article/32/12/i262/2288786)

The data attributes are used to find correct protein-portein models and are listed as follows within the above mentionned paper in the Training features section:

- rGb: Residue Given burial. Relative solvant accessibility of the protein amino acids. Values range around 0.059 (+- 0.022)
- nBSA: Normalized buried surface area. It measures the fraction of exposed surface area buried upon association
- Fintres: Fraction of residues buried at the interface
- Sc: Shape Complementarity at the interface
- Ec: Electrostatic Complementarity at the interface
- ProQ: Protein quality predictor score
- Isc: Rosetta energy at the interface
- rTs: Roseta total energy
- Erep: Rosetta repulsive term
- Etmr: Rosetta total ernergy minus repulsive
- CPM: Joint Conditional Probability of Sc, EC given nBSA. CPM is the joint conditional probability of finding its interface within a certain range of Sc and EC given its size (nBSA)
- Ld: Link Density at the interface
- CPscore: Contact Preference score

As presented in the paper, the target function or benchmark is also part of our dataset and lists the following properties:

- DockQ: Score of quality for a protein-protein docking model
- DockQ-binary: Applied threshold on the DockQ score reflecting no similarity or perfect similarity scores
- ProQDock: Predicted DockQ protein docking quality score
- zrank and zrank2: All atom energy terms. Non bonded energy terms based (Coulomb, Van der Waals, desolvation)
- ProQDockZ: External energy term. Hybrid method combining ProQDock and Zrank.

Finally here, 'cv' represents the cross validation batched used initally in our dataset.

In [1]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

train=pd.read_csv('ProQDock.csv')
train
```

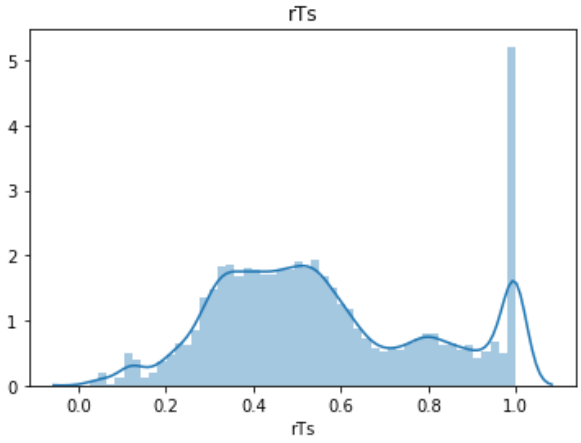
Out[1]:

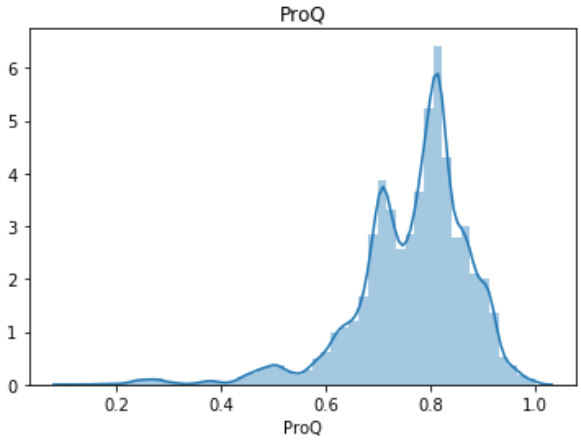
	Model	rGb	nBSA	Fintres	Sc	EC	ProQ	zrank	zrank2	Isc	...	Erep	Etmr	CPM	I
0	T50-1	0.035	0.034	0.106	0.571	0.072	0.682	0.611	0.657	1.000	...	0.998	0.400	0.723	0.1
1	T50-2	0.033	0.036	0.124	0.579	-0.128	0.703	0.633	0.671	1.000	...	0.998	0.487	0.695	0.0
2	T50-3	0.042	0.027	0.088	0.776	0.434	0.698	0.536	0.452	0.464	...	0.611	0.345	0.857	0.1
3	T50-4	0.032	0.032	0.118	0.514	0.458	0.640	0.579	0.534	0.490	...	0.406	0.911	0.735	0.1
4	T50-5	0.040	0.029	0.102	0.336	0.172	0.708	0.589	0.839	1.000	...	1.000	0.419	0.451	0.0
5	T50-6	0.046	0.030	0.104	0.375	-0.084	0.710	0.612	0.711	1.000	...	1.000	0.417	0.372	0.1
6	T50-7	0.043	0.027	0.090	0.602	0.429	0.746	0.633	0.584	0.784	...	0.647	0.381	0.799	0.1
7	T50-8	0.041	0.014	0.058	0.575	0.051	0.715	0.622	0.682	1.000	...	1.000	0.421	0.723	0.1
8	T50-9	0.041	0.037	0.106	0.377	-0.287	0.683	0.687	0.997	1.000	...	1.000	0.491	0.289	0.0
9	T50-10	0.045	0.020	0.082	0.552	0.065	0.714	0.639	0.655	0.527	...	0.982	0.414	0.723	0.1
10	T50-11	0.035	0.040	0.116	0.401	0.026	0.704	0.739	1.000	1.000	...	1.000	0.390	0.473	0.1

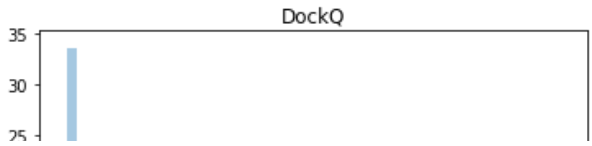
Now, we can plot some of the columns containing continuous values to visualize the distributions.

In [2]:
```
plt.clf()
sns.distplot(train['rTs'])
import numpy as np
estimated_rows=(train['rTs']>0.6) & (train['rTs']<=1.0)
print("rTs mean: " + str(np.mean(train['rTs'])))
print("tail rTs mean: " + str(np.mean(train[~estimated_rows]['rTs'])))
print("DockQ mean: " + str(np.mean(train['DockQ'])))
print("DockQ std: " + str(np.std(train['DockQ'])))
print("ProQDock mean: " + str(np.mean(train['ProQDock'])))
print("ProQDock std: " + str(np.std(train['ProQDock'])))
plt.title('rTs')
plt.show()

sns.distplot(train['ProQ'].dropna())
plt.title('ProQ')
plt.show()

sns.distplot(train['DockQ'].dropna())
plt.title('DockQ')
plt.show()

sns.distplot(train['ProQDock'].dropna())
plt.title('ProQDock')
plt.show()
```

```
rTs mean: 0.5602546178969766
tail rTs mean: 0.4060517957694497
DockQ mean: 0.05342665370177128
DockQ std: 0.13084898615761434
ProQDock mean: 0.1529110643090434
ProQDock std: 0.16285711061293873
```







It is then possible to sort the data to find the proteins with the best relative solvent accessibility residues using rGb attribute, or the best predicted ProQDock score using ProQDock atrtibute. Finally we can select the categorical DockQ-Binary score and sort in ascending mode to get the minimun value for the binary threshold.

In [3]:
```
train.sort_values('rGb',ascending=False)[:10]
train.sort_values('ProQDock',ascending=False)[:10]
train_log[(train['DockQ_Binary'] == 1)].sort_values('DockQ',ascending=True)[:10]
```

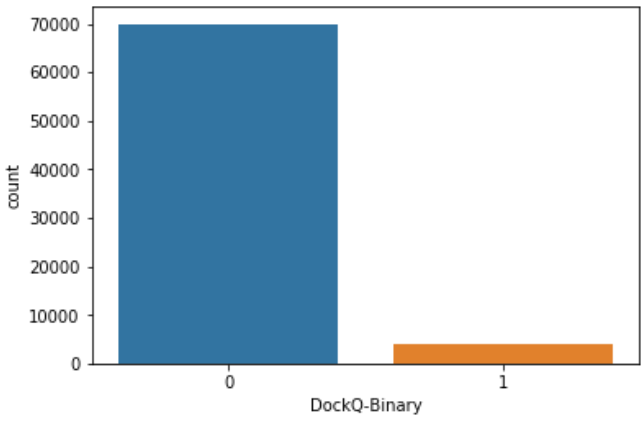
Out[3]:

	Model	rGb	nBSA	Fintres	Sc	EC	ProQ	zrank	zrank2	Isc	...	Erep	Etmr	CPM	
30928	D2OOB-a88d	0.048	0.062	0.366	0.778	-0.139	0.580	0.362	0.283	0.879	...	0.445	0.784	0.703	0.
30546	D2OOB-a103a	0.056	0.060	0.380	0.539	0.005	0.604	0.406	0.369	0.873	...	0.456	0.640	0.434	0.
65988	D1I9R-a91a	0.022	0.030	0.151	0.642	0.158	0.820	0.596	0.635	0.260	...	0.525	0.606	0.739	0.
48273	D1MQ8-a22a	0.003	0.041	0.227	0.682	0.453	0.904	0.469	0.559	0.300	...	0.624	0.431	0.891	0.
70821	D2FD6-a124a	0.002	0.031	0.147	0.569	0.354	0.789	0.572	0.518	0.448	...	0.480	0.806	0.830	0.
1550	T29-105	0.067	0.025	0.066	0.480	0.249	0.729	0.518	0.972	1.000	...	0.982	0.803	0.623	0.
2766	T29-1379	0.066	0.031	0.095	0.368	0.046	0.743	0.599	0.999	1.000	...	1.000	0.810	0.342	0.
3453	T29-2099	0.065	0.037	0.102	0.366	-0.090	0.747	0.613	1.000	1.000	...	1.000	0.793	0.372	0.
14031	T41-550	0.083	0.066	0.205	0.554	0.294	0.701	0.279	0.199	1.000	...	0.997	0.786	0.679	0.
10130	T35-418	0.031	0.038	0.112	0.242	-0.122	0.752	0.743	1.000	1.000	...	1.000	0.834	0.280	0.

10 rows × 21 columns

As an additional visualization and after the continuous values, we will focus on the categorical value in our dataset.

In [4]:
```
plt.clf()
sns.countplot(x="DockQ-Binary", data=train);
plt.show()
```



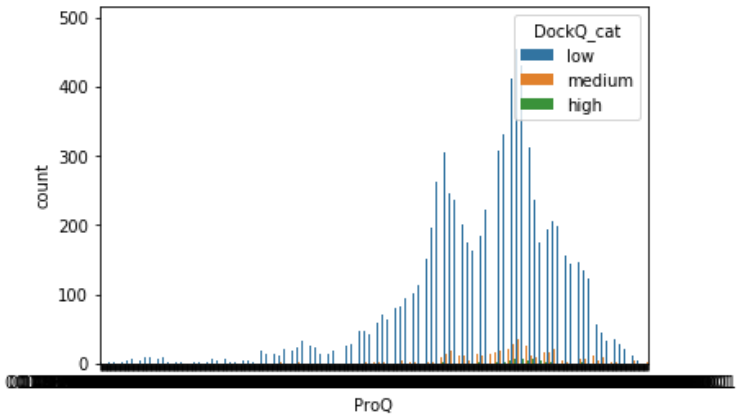
It is also possible to add a categorical column based on the binary score of dockQ for the protein-protein interaction.

In [5]:
```
import numpy as np
v = train["DockQ"].values
cats = np.array(['low', 'medium', 'high'])
min_cat = np.min(train.loc[(train['DockQ-Binary'] == 1)]['DockQ']) + np.std(train
max_cat = np.max(train.loc[(train['DockQ-Binary'] == 1)]['DockQ']) - np.std(train
code = np.searchsorted([min_cat, max_cat], v.ravel()).reshape(v.shape)
train['DockQ_cat'] = cats[code]
train.loc[(train['DockQ_cat'] == 'high')]
```

Out[5]:

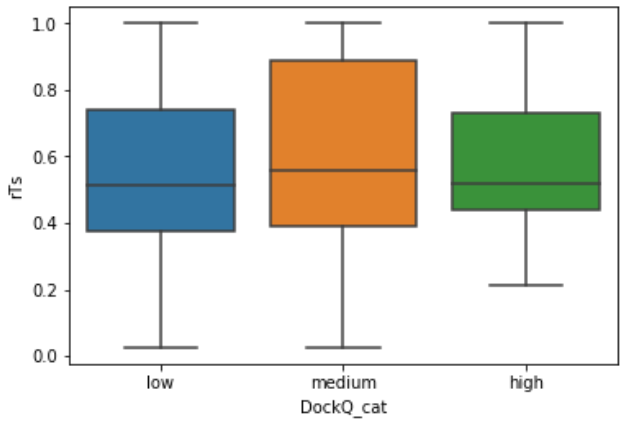
	Model	rGb	nBSA	Fintres	Sc	EC	ProQ	zrank	zrank2	Isc	...	Etmr	CPM	Ld	CPs
8552	T37-305	0.057	0.049	0.148	0.736	0.606	0.821	0.370	0.189	0.114	...	0.279	0.808	0.120	
9042	T37-853	0.051	0.050	0.148	0.664	0.634	0.828	0.332	0.139	1.000	...	0.294	0.871	0.118	
10297	T40-23	0.008	0.059	0.300	0.714	0.532	0.838	0.410	0.181	0.975	...	0.587	0.936	0.098	
10322	T40-48	0.012	0.056	0.296	0.632	0.364	0.833	0.467	0.523	1.000	...	0.397	0.808	0.100	
10324	T40-50	-0.001	0.043	0.251	0.750	0.757	0.806	0.422	0.349	0.241	...	0.566	0.625	0.107	
10330	T40-56	0.012	0.056	0.278	0.697	0.527	0.848	0.428	0.212	0.037	...	0.610	0.883	0.104	
10332	T40-58	-0.000	0.049	0.274	0.455	0.630	0.808	0.483	0.679	1.000	...	0.432	0.590	0.116	
10334	T40-60	0.006	0.059	0.296	0.686	0.546	0.847	0.427	0.200	0.182	...	0.553	0.883	0.090	
10337	T40-63	-0.003	0.043	0.256	0.764	0.449	0.823	0.417	0.323	0.365	...	0.550	0.857	0.117	
10358	T40-84	0.005	0.055	0.296	0.754	0.509	0.852	0.427	0.224	0.246	...	0.453	0.928	0.101	
10377	T40-103	0.001	0.044	0.238	0.609	0.621	0.810	0.446	0.649	1.000	...	0.398	0.730	0.128	

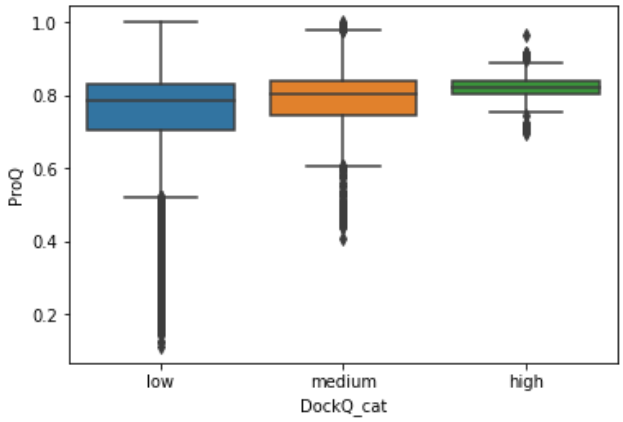
In [6]:
```
plt.clf()
sns.countplot(x="ProQ", hue='DockQ_cat', data=train);
plt.show()
```



It is possible as well to visualize the link between certain attributes and the DockQ score classes (either binary or the added categorical column). It will give us some insight regarding the similarity between the attributes and their capacity to describe the target score. A more advance study on the similarity on the attributes or the rows will be conducted in the next part.

In [7]:
```
plt.clf()
sns.boxplot(x="DockQ_cat", y="rTs", data=train)
plt.show()
sns.boxplot(x="DockQ_cat", y="ProQ", data=train)
plt.show()
```





2 Clustering

As we performed it during the exercise session, we will cluster the data based on some of the attributes to analyse the similarity of the rows. Just as in the exercises, we have to scale the attributes using sklearn python module.

In [8]:
```
from sklearn import preprocessing

train['DockQ_cat'] = train['DockQ_cat'].astype('category')
cat_columns = train.select_dtypes(['category']).columns
train['DockQ_cat-int'] = train[cat_columns].apply(lambda x: x.cat.codes)

trainable_cols=["rGb","nBSA","Fintres","Sc","EC","ProQ","Isc","rTs","Erep","Etmr"
trainable_cols_target=trainable_cols + ["DockQ-Binary"] + ["DockQ_cat-int"]
train_target=train[trainable_cols_target].dropna()
df=train_target[trainable_cols]

pd.options.mode.chained_assignment = None  # default='warn'

scaling=True
if scaling:
    min_max_scaler = preprocessing.MinMaxScaler()
    columns_to_scale=trainable_cols
```

Once the data are normalized and using the module sklearn, we will apply PrincipalComponent Analysis clustering on our dataset. Once computed, the components will be plotted.

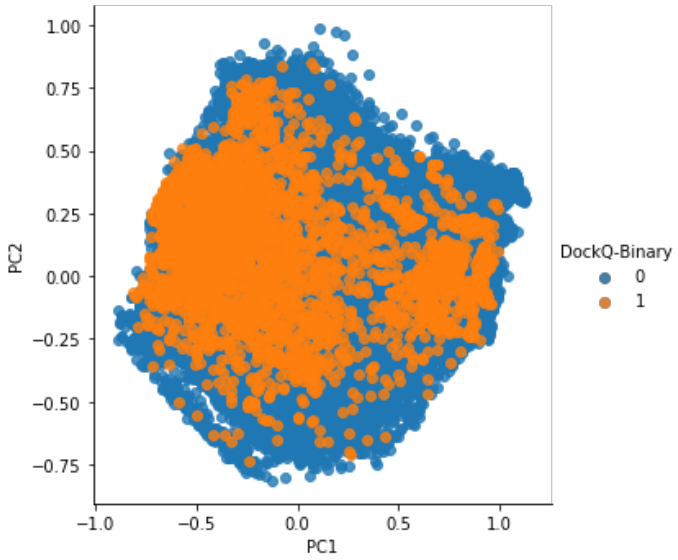
In [9]:
```
from sklearn.decomposition import PCA

pca=PCA(n_components=5)
X = pca.fit(df.values).transform(df.values)
print('explained variance ratio: %s'
      % str(pca.explained_variance_ratio_))
print(np.sum(pca.explained_variance_ratio_[0:3]))
print(df.shape)
train_target['PC1']=X[:,0]
train_target['PC2']=X[:,1]
train_target['PC3']=X[:,2]
train_target['PC4']=X[:,3]
train_target['PC5']=X[:,4]
print (pca.components_)
df.columns
```

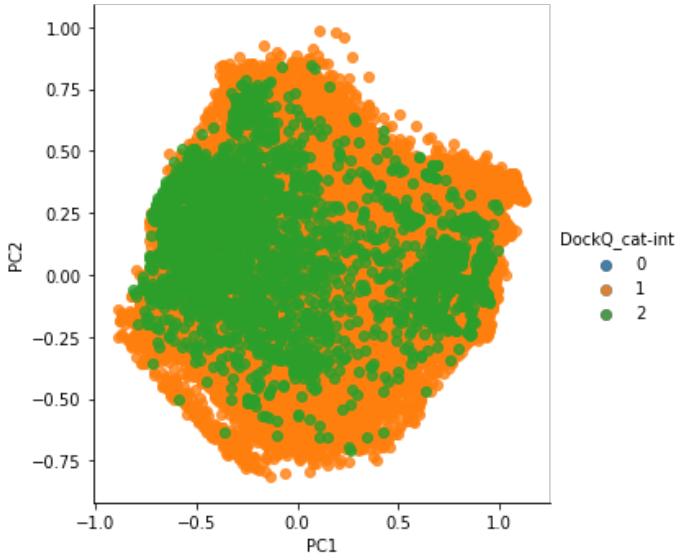
```
explained variance ratio: [0.41745529 0.20978048 0.12164017 0.07557945 0.0623216
8]
0.7488759383296759
(73789, 13)
[[-0.00374763 -0.02858294 -0.05922698 -0.18538819 -0.06827822 -0.09869298
   0.67089074  0.47226253  0.3884008   0.15725463 -0.31310572  0.01101119
  -0.00481878]
 [-0.01915935  0.05414057  0.06885     0.02618219  0.07510112 -0.1322844
  -0.5587904   0.56693507  0.09489171  0.56777336  0.05199549 -0.0165276
   0.01125764]
 [-0.00912357  0.14445139  0.17199819 -0.25242131 -0.22214998  0.00217774
  -0.42510317 -0.03104567  0.25205552 -0.36215537 -0.67681765 -0.04404686
  -0.00282973]
 [-0.0295906   0.21537207  0.39196438  0.08642008 -0.42491739 -0.18265192
   0.19359791 -0.16219594 -0.5246276   0.40494197 -0.2724514  -0.01827164
   0.01244329]
 [-0.03635431  0.43310545  0.67044487  0.10702555  0.43294955 -0.17864764
   0.11965255  0.02991242  0.20193073 -0.19228325  0.18595182 -0.05680988
   0.00459985]]
```

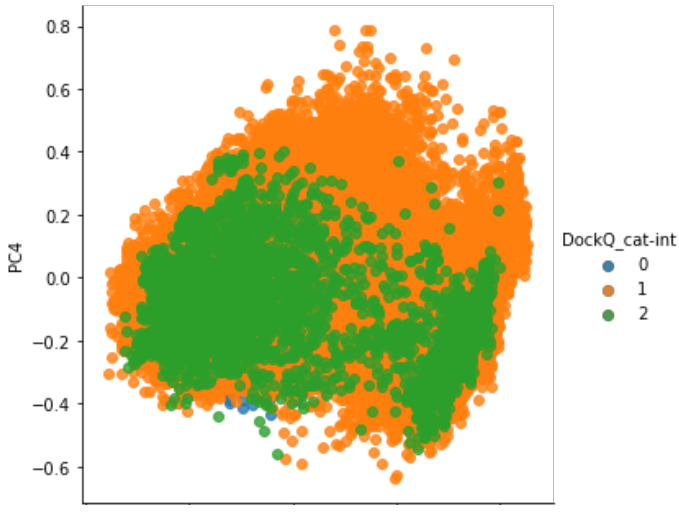
Out[9]:
```
Index(['rGb', 'nBSA', 'Fintres', 'Sc', 'EC', 'ProQ', 'Isc', 'rTs', 'Erep',
       'Etmr', 'CPM', 'Ld', 'CPscore'],
      dtype='object')
```

In [10]:
```
plt.clf()
sns.lmplot(x='PC1',y='PC2',hue='DockQ-Binary',data=train_target,fit_reg=False)
plt.show()
sns.lmplot(x='PC1',y='PC2',hue='DockQ_cat-int',data=train_target,fit_reg=False)
plt.show()
sns.lmplot(x='PC1',y='PC4',hue='DockQ_cat-int',data=train_target,fit_reg=False)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```







3 Supervised learning

After visualizing and understanding the data, we will train different machine learning methods to predict the DockQ binary score.

Cross validation

Here we will divide our dataset into subset used to test our supervised learning methods. Five-fold cross validation has been done for the ProQDock dataset. The division of subset has been conducted so that there is no homologous proteins between the substes.Each subset was built so that the number of models in each is similar.

In [11]:
```
print(train["cv"].value_counts())
```

```
5    15798
2    15365
1    14509
4    14492
3    13625
Name: cv, dtype: int64
```

Scaling the dataset and preparing the training

Before implementing the training of our machine learning methods we will scale the features of our dataset betwee 0 and 1. At first we will chose to scale all the 13 feature columns of our dataset.

In [12]:
```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()

train_columns = trainable_cols + ["DockQ-Binary", "cv"]

train_data=train[train_columns].dropna()

scaling=True
if scaling:
    columns_to_scale=trainable_cols
    # Fit the scaler on the training data
    min_max_scaler.fit(train_data[columns_to_scale].values)
    # Transform the scaling to the train_data
    train_data.loc[:,columns_to_scale]=min_max_scaler.transform(train_data[column

#train_data
```

To be used to train machine learning methods we will create a numpy matrix X for the training data and a vector Y for the target values. As done in the course exercise, we will use the predefined cross-validation splits form the sklearn module.

In [13]:
```
from sklearn.model_selection import PredefinedSplit
(size_x,size_y)=train_data.shape
target_index=size_y-2
cv_index=size_y-1

# Build the X matrix from the training dataset
X=train_data[trainable_cols].values

# Build the Y vector for the target
Y=train_data['DockQ-Binary'].values

cv = PredefinedSplit(train_data['cv'].values)
```

Machine Learning methods training

Just like in the course exercises, we will now test several machine learning methods on our prepared dataset.

Random Forest Classifier

The first method is the random forest classifier. We will firstly have a look into the choice of hyperparameters.

From sklearn module we get the definitions for the different hyperparameters:

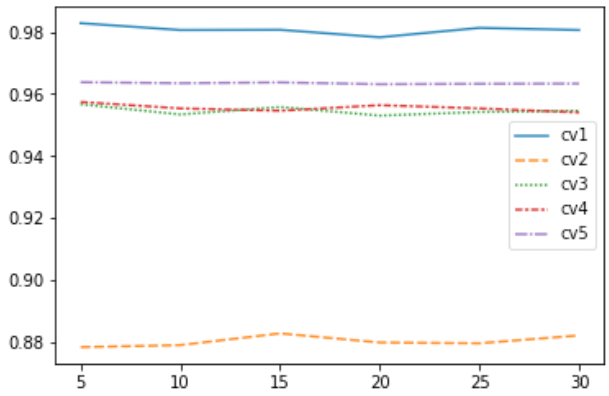
1. n_estimators: The number of trees in the forest of the model. The default value for this parameter is 10, which means that 10 different decision trees will be constructed in the random forest.
2. max_depth: The maximum depth of each tree. The default value for max_depth is None, which means that each tree will expand until every leaf is pure. A pure leaf is one where all of the data on the leaf comes from the same class.
3. min_samples_split: The minimum number of samples required to split an internal leaf node. The default value for this parameter is 2, which means that an internal node must have at least two samples before it can be split to have a more specific classification.
4. min_samples_leaf: The minimum number of samples required to be at a leaf node. The default value for this parameter is 1, which means that every leaf must have at least 1 sample that it classifies.

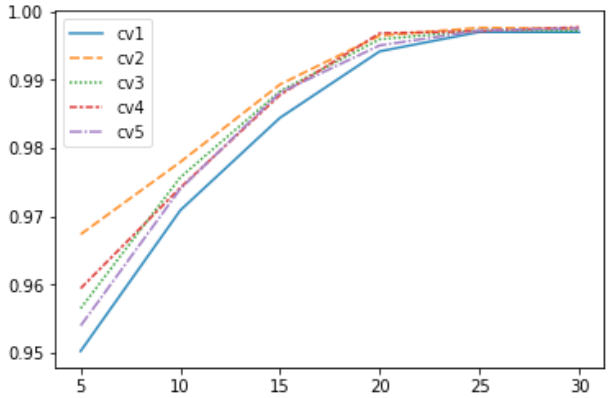
Here we run a benchmark on the different hyperparameters wiht the previous cross validation. Please note that, given the size of the dataset, the computation is probably too long to be ran here, and has been run on a Colab notebook (please see https://colab.research.google.com/drive/1Cm9yBDI1j9KQxuio3DC95c8wDI6iX_qC (https://colab.research.google.com/drive/1Cm9yBDI1j9KQxuio3DC95c8wDI6iX_qC))

In [14]:
```
from sklearn.model_selection import cross_val_score, validation_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


#train_scoreNum, test_scoreNum = validation_curve(
#                            RandomForestClassifier(),
#                            X = X, y = Y,
#                            param_name = 'n_estimators',
#                            param_range = [100, 300, 500, 700, 800, 1200], cv =
#print(train_scoreNum)
#print(test_scoreNum)
```

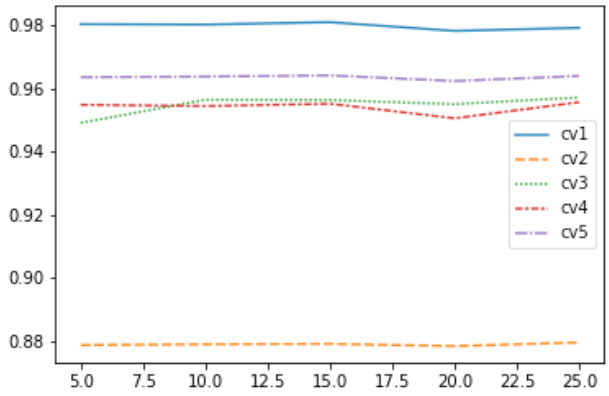
In [15]:
```
#df = pd.DataFrame(data=test_scoreNum,    # values
#              index=['100', '300', '500', '700', '800', '1200'],    # 1st column
#            columns=['cv1','cv2','cv3','cv4','cv5'])
#ax = sns.lineplot(data=df)
#plt.show()
#df2 = pd.DataFrame(data=train_scoreNum,    # values
#              index=['100', '300', '500', '700', '800', '1200'],    # 1st column
#              columns=['cv1','cv2','cv3','cv4','cv5'])
#ax2 = sns.lineplot(data=df2)
#plt.show()
```

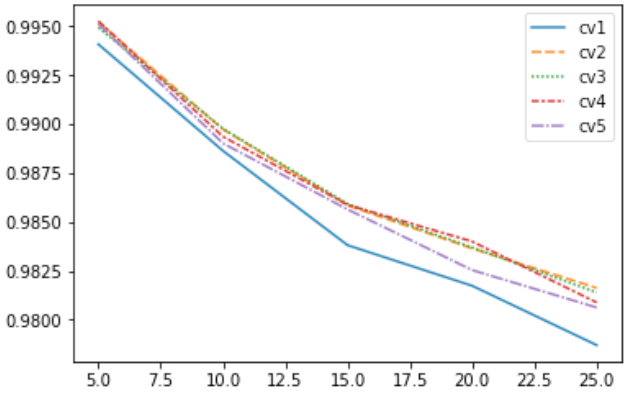
We ran all the mentionned hyperparameter (see Colab notebook) and obtaind the following curves: On generated test, followed by train dataset, for the max_depth hyperparameter, we can see the trend at 20 which stabilizes, we will use this hyperparamter.



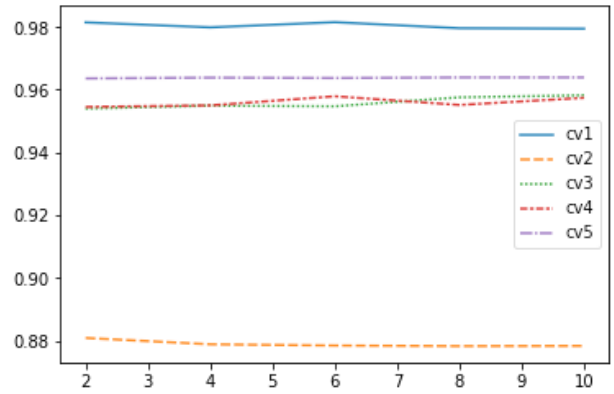


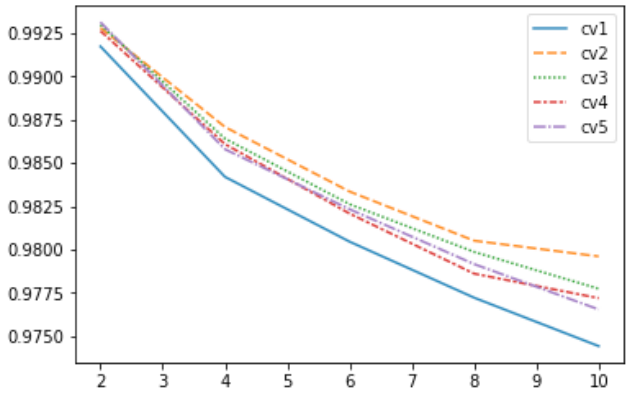
On generated test, followed by train dataset, for the min_samples_split hyperparameter, we will use 5 here for this hyperparamter.





On generated test, followed by train dataset, for the min_samples_leaf hyperparameter, we will use 2 here for this hyperparamter.





Running the Random Forest Classifier with the correct choice of hyperparameters

Based on the previous benchmark we will run the RFC with the chosen hyperparamters and using the example provided in the course exercises.

In [16]:
```
clf = RandomForestClassifier(n_estimators=100, max_depth=20, min_samples_split=5,
```

In [17]:
```
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn import svm
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

pred_save=[]
true_save=[]
pred_prob_save=[]

for i, (train_index, val_index) in enumerate(cv.split(),1):
    print("Set: ",i)
    print("Training on",len(train_index),"examples")
    print("Testing on",len(val_index),"examples")
    (X_train, X_val) = X[train_index,:], X[val_index,:]
    (Y_train, Y_val) = Y[train_index], Y[val_index]

    clf=clf.fit(X_train,Y_train)

    #Predict on the training data
    pred=clf.predict(X_train)
    #Calculate performance measures on the validation data
    acc_train=accuracy_score(pred,Y_train)
    mcc_train=matthews_corrcoef(pred,Y_train)
    f1_train=f1_score(pred,Y_train)

    #Predict on the validation data
    val_pred=clf.predict(X_val)
    #Predict the probability (to use the roc-plot later)
    val_pred_prob=val_pred

    #Save the values to have predictions for all folds.
    pred_save.append(val_pred)
    pred_prob_save.append(val_pred_prob)
    true_save.append(Y_val)
    #Calculate performance measures on the validation data
    acc=accuracy_score(val_pred,Y_val)
    mcc=matthews_corrcoef(val_pred,Y_val)
    f1=f1_score(val_pred,Y_val)

    print("Training performance","f1",f1_train,"acc",acc_train,"mcc",mcc_train)
    print("Validation performance","f1",f1,"acc",acc,"mcc",mcc)
    print("==============")



#Calculate overall validation performance
predictions=np.concatenate(pred_save)
correct=np.concatenate(true_save)
predicted_prob=np.concatenate(pred_prob_save)
acc=accuracy_score(predictions,correct)
mcc=matthews_corrcoef(predictions,correct)
f1=f1_score(predictions,correct)
print("==============")
print("Overall Validation Performance","f1",f1,"acc",acc,"mcc",mcc)
print("==============")

pred_save=np.concatenate(pred_save)
true_save=np.concatenate(true_save)
pred_prob_save=np.concatenate(pred_prob_save)
(fpr,tpr,thres_roc)=roc_curve(true_save,pred_prob_save)
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('fpr')
plt.ylabel('tpr')
```

Here we have a overall f1 score (precision and recall) of 0.03. It's very low value, meaning that our model have issues with false positive or false negative. The overall accuracy score is 0.94 and represents the fraction of correctly predicted samples. Finally, mcc score, representing a correlation score to measure the quality of binary classification, is 0.06 which is also a low value and show issues with our model. The ROC Curve shows that our model failed in predicting the DockQ-Binary score of the samples.

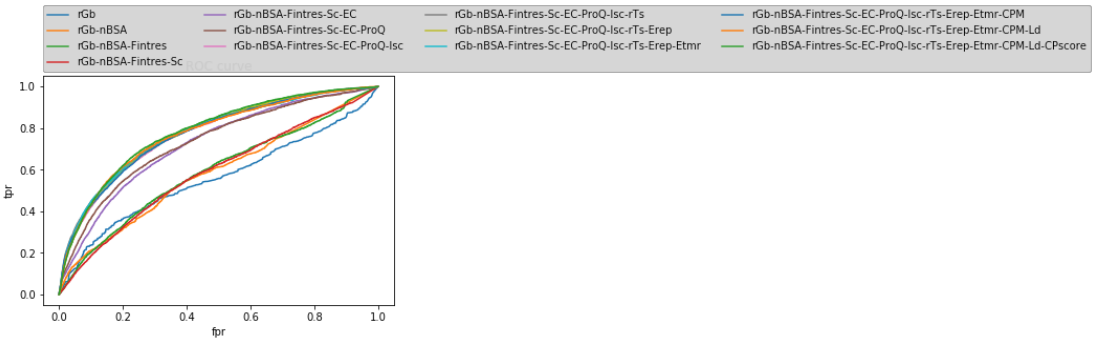
As done in the course exercise, we can try to cycle through the features to check if it can increase the performance.

In [18]:
```
#Some dictionaries to store cross-validated predictions
predictions={}
correct={}
predicted_prob={}

pred_save=[]
true_save=[]
pred_prob_save=[]
legend_text=[]

for feat_stop in range(0,X.shape[1]+1):
    name="-".join(trainable_cols[0:feat_stop+1])
    legend_text.append(name)
    pred_save=[]
    true_save=[]
    pred_prob_save=[]
    for i, (train_index, val_index) in enumerate(cv.split(),1):
        (X_train, X_val) = X[train_index,0:feat_stop+1], X[val_index,0:feat_stop+
        (Y_train, Y_val) = Y[train_index], Y[val_index]
        clf=clf.fit(X_train,Y_train)

    #Predict on the training data
        pred=clf.predict(X_train)
    #Calculate performance measures on the validation data
        acc_train=accuracy_score(pred,Y_train)
        mcc_train=matthews_corrcoef(pred,Y_train)
        f1_train=f1_score(pred,Y_train)

    #Predict on the validation data
        val_pred=clf.predict(X_val)
    #Predict the probability (to use the roc-plot later)
        val_pred_prob=clf.predict_proba(X_val)
    #Save the values to have predictions for all folds.
        pred_save.append(val_pred)
        pred_prob_save.append(val_pred_prob)
        true_save.append(Y_val)
    #Calculate performance measures on the validation data
        acc=accuracy_score(val_pred,Y_val)
        mcc=matthews_corrcoef(val_pred,Y_val)
        f1=f1_score(val_pred,Y_val)


#Calculate overall validation performance
    predictions[name]=np.concatenate(pred_save)
    correct[name]=np.concatenate(true_save)
    predicted_prob[name]=np.concatenate(pred_prob_save)
    acc=accuracy_score(predictions[name],correct[name])
    mcc=matthews_corrcoef(predictions[name],correct[name])
    f1=f1_score(predictions[name],correct[name])
    print("==============")
    print("Training on", name)
    print("Overall Validation Performance","f1",f1,"acc",acc,"mcc",mcc)
    print("==============")
```

```
/Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib/python3.6/site-packages
/sklearn/metrics/classification.py:538: RuntimeWarning: invalid value encountere
d in double_scalars
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
/Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib/python3.6/site-packages
/sklearn/metrics/classification.py:1137: UndefinedMetricWarning: F-score is ill-
defined and being set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib/python3.6/site-packages
/sklearn/metrics/classification.py:538: RuntimeWarning: invalid value encountere
d in double_scalars
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
/Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib/python3.6/site-packages
/sklearn/metrics/classification.py:1137: UndefinedMetricWarning: F-score is ill-
defined and being set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib/python3.6/site-packages
```

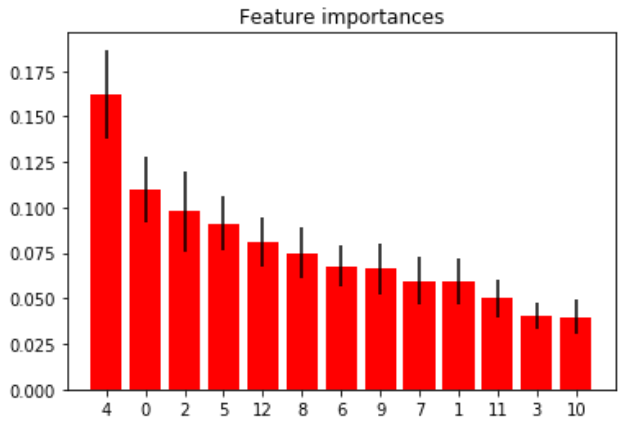
In [19]:
```
plt.clf()

legend_text=[]
pred_sorted=sorted(predictions.items(), key=lambda kv: (len(kv[1]), kv[0]))
print(pred_sorted)


for (name,value) in pred_sorted:
    #print key, value
    #continue
    legend_text.append(name)
    acc=accuracy_score(predictions[name],correct[name])
    mcc=matthews_corrcoef(predictions[name],correct[name])
    f1=f1_score(predictions[name],correct[name])
    #(prec,recall,thres)=precision_recall_curve(true_save,pred_prob_save[:,1])
    (fpr,tpr,thres_roc)=roc_curve(correct[name],predicted_prob[name][:,1])
    plt.plot(fpr,tpr)
    plt.title('ROC curve')
    plt.xlabel('fpr')
    plt.ylabel('tpr')

    plt.savefig('RF.png',dpi=300)
plt.legend(legend_text, bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left',
           ncol=4, borderaxespad=0.)#mode="expand"
plt.show()
```

```
[('rGb', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA', array([0, 0, 0, ..., 0,
0, 0])), ('rGb-nBSA-Fintres', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA-Fintre
s-Sc', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA-Fintres-Sc-EC', array([0, 0,
0, ..., 0, 0, 0])), ('rGb-nBSA-Fintres-Sc-EC-ProQ', array([0, 0, 0, ..., 0, 0,
0])), ('rGb-nBSA-Fintres-Sc-EC-ProQ-Isc', array([0, 0, 0, ..., 0, 0, 0])), ('rGb
-nBSA-Fintres-Sc-EC-ProQ-Isc-rTs', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA-F
intres-Sc-EC-ProQ-Isc-rTs-Erep', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA-Fin
tres-Sc-EC-ProQ-Isc-rTs-Erep-Etmr', array([0, 0, 0, ..., 0, 0, 0])), ('rGb-nBSA-
Fintres-Sc-EC-ProQ-Isc-rTs-Erep-Etmr-CPM', array([0, 0, 0, ..., 0, 0, 0])), ('rG
b-nBSA-Fintres-Sc-EC-ProQ-Isc-rTs-Erep-Etmr-CPM-Ld', array([0, 0, 0, ..., 0, 0,
0])), ('rGb-nBSA-Fintres-Sc-EC-ProQ-Isc-rTs-Erep-Etmr-CPM-Ld-CPscore', array([0,
0, 0, ..., 0, 0, 0]))]
```



In [20]:
```
print(clf.feature_importances_)

importances = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
[0.10946532 0.05915532 0.09770457 0.04059754 0.16242392 0.09099839
 0.06775387 0.05976718 0.07483966 0.06639083 0.039858   0.05007751
 0.08096788]
Feature ranking:
1. feature 4 (0.162424)
2. feature 0 (0.109465)
3. feature 2 (0.097705)
4. feature 5 (0.090998)
5. feature 12 (0.080968)
6. feature 8 (0.074840)
7. feature 6 (0.067754)
8. feature 9 (0.066391)
9. feature 7 (0.059767)
10. feature 1 (0.059155)
11. feature 11 (0.050078)
12. feature 3 (0.040598)
13. feature 10 (0.039858)
```



From the previous result it is possible to see that only 5 features would yield a better ROC curve with the training on rGb-nBSA-Fintres-Sc-EC. We can also conclude from the feature cycling that the feature EC is the most prevalent for a good fit.

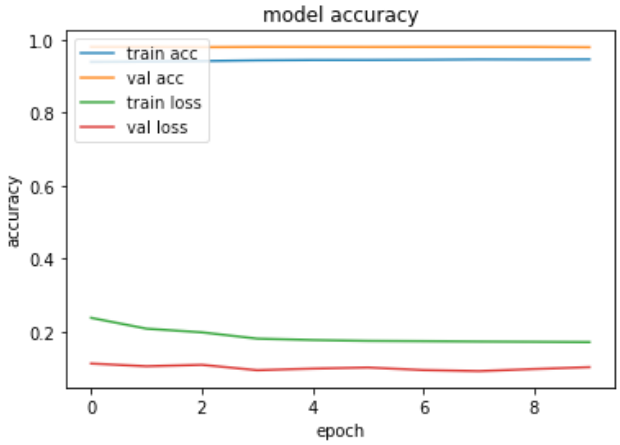
Deep learning

Im this last part we will use keras and tensorflow modules to train a neural network for deep learning on our ProQDock dataset. Here we will try to predict the DockQ-Binary score like we did with thee RFC. We build a four layers of 3 nodes for the architecture of our model.

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt

def plot_loss_acc(history):
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train acc', 'val acc', 'train loss', 'val loss'], loc='upper lef
    plt.show()

model = Sequential()

model.add(Dense(units=3, activation='tanh', input_dim=13))
model.add(Dense(units=3, activation='tanh', input_dim=13))
model.add(Dense(units=3, activation='tanh', input_dim=13))
model.add(Dense(units=3, activation='tanh', input_dim=13))
model.add(Dense(units=2, activation='softmax'))

model.compile(optimizer='rmsprop',                    #adaptive learning rate met
              loss='sparse_categorical_crossentropy', #loss function for classifi
              metrics=['accuracy'])                   #the metric doesn't influen

hist = model.fit(X, Y, epochs=10, batch_size=32, validation_split=0.2)

plot_loss_acc(hist)
```

```
Using TensorFlow backend.

WARNING:tensorflow:From /Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib
/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colo
cate_with (from tensorflow.python.framework.ops) is deprecated and will be remov
ed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /Users/pierrebedoucha/miniconda3/envs/medbioinfo-env/lib
/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future ve
rsion.
Instructions for updating:
Use tf.cast instead.
Train on 59031 samples, validate on 14758 samples
Epoch 1/10
```

There can be a problem with Tensorflow backend here. In that case, please see the previoulsy metionned Colab notebook (https://colab.research.google.com/drive/1Cm9yBDI1j9KQxuio3DC95c8wDI6iX_qC (https://colab.research.google.com/drive/1Cm9yBDI1j9KQxuio3DC95c8wDI6iX_qC)) where the upper cell has been run. We obtain the following loss and accuracy plot. The accuracy is above 0.8 and the loss decreases from the first epoch.


model accuracy
train acc
val acc
train loss
val loss
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

In [ ]: