Part 2 - Final Modelling

After our exploration and model spot-checking in Part 1, we will progress to applying our best models in a pipeline after some grid searches for optimal parameters and see if we can improve the result with a voting ensemble.

Show hidden markdown

Show hidden cell

Import Data

Show hidden code

```
Train data:
 (2056, 13)
   Id  fixed acidity  volatile acidity  citric acid  residual sugar
\
0   0            8.0              0.50         0.39             2.2
1   1            9.3              0.30         0.73             2.3
2   2            7.1              0.51         0.03             2.1
3   3            8.1              0.87         0.22             2.6
4   4            8.5              0.36         0.30             2.3

   chlorides  free sulfur dioxide  total sulfur dioxide  density
pH  \
0      0.073                 30.0                  39.0  0.99572
3.33
1      0.092                 30.0                  67.0  0.99854
3.32
2      0.059                  3.0                  12.0  0.99660
3.52
3      0.084                 11.0                  65.0  0.99730
3.20
4      0.079                 10.0                  45.0  0.99444
3.20

   sulphates  alcohol  quality
0       0.77     12.1        6
1       0.67     12.8        6
2       0.73     11.3        7
3       0.53      9.8        5
4       1.36      9.5        6
```

Prepare Data

Split data into Train and Validation Sets

In [3]:

```
# Evaluate using a train and a test set
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#numerical predictor (NP) and response locations
NP_first_loc = 'fixed acidity' #ignore col 0 which is the ID
NP_last_loc = 'alcohol'
outcome_loc = 'quality'

NP_first_iloc = train.columns.get_loc(NP_first_loc) #ignore col 0 which is the ID
#print("NP_first_iloc:",NP_first_iloc, "\n" )
NP_last_iloc = train.columns.get_loc(NP_last_loc)
#print("NP_last_iloc:",NP_last_iloc, "\n" )
outcome_iloc = train.columns.get_loc(outcome_loc)

#Original Data
X = train.loc[:,NP_first_loc:NP_last_loc]
#print("\nX shape: ", X.shape)
#print("First two rows of X:\n", X.iloc[0:2, :], "\n")
Y = train.loc[:,outcome_loc]
#print("\n\nFist 10 rows of Y: \n", Y.iloc[0:10])

#Split Data
test_size = 0.25
seed = 100 #re-evaluate with different seeds to estimate variance?
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=test_size,rand
om_state=seed)
```

Lazy Predict

Just because we found out about this package, we'll slip in a quick look at the lazy predict results to compare to what we determined in Part 1.

In [4]:

```
pip install lazypredict
```

Show hidden output

In [5]:

```
from lazypredict.Supervised import LazyClassifier

clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X_train, X_val, Y_train, Y_val)
models
```

```
100%|██████████| 29/29 [00:08<00:00,  3.34it/s]
```

Out[5]:

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
Model					
AdaBoostClassifier	0.26	0.30	None	0.29	0.20
LinearDiscriminantAnalysis	0.59	0.30	None	0.57	0.04
QuadraticDiscriminantAnalysis	0.57	0.28	None	0.55	0.01
LogisticRegression	0.60	0.28	None	0.57	0.09
BernoulliNB	0.57	0.27	None	0.55	0.02
PassiveAggressiveClassifier	0.50	0.27	None	0.49	0.02
RandomForestClassifier	0.59	0.27	None	0.56	0.48
NearestCentroid	0.25	0.27	None	0.31	0.01
SGDClassifier	0.57	0.26	None	0.54	0.08
CalibratedClassifierCV	0.59	0.26	None	0.54	1.93
LGBMClassifier	0.55	0.26	None	0.53	3.84
ExtraTreesClassifier	0.57	0.26	None	0.54	0.38
SVC	0.58	0.26	None	0.54	0.24
GaussianNB	0.54	0.26	None	0.53	0.02
Perceptron	0.51	0.25	None	0.48	0.02
BaggingClassifier	0.54	0.25	None	0.52	0.12
LinearSVC	0.58	0.25	None	0.52	0.55
RidgeClassifier	0.58	0.24	None	0.51	0.02
RidgeClassifierCV	0.58	0.24	None	0.51	0.02
ExtraTreeClassifier	0.46	0.24	None	0.45	0.02
DecisionTreeClassifier	0.46	0.24	None	0.46	0.03
LabelSpreading	0.45	0.24	None	0.45	0.24
LabelPropagation	0.45	0.24	None	0.45	0.19
KNeighborsClassifier	0.51	0.23	None	0.47	0.04
DummyClassifier	0.43	0.17	None	0.26	0.01

The lazy prediction confirms that an LDA model is a good option.

Tune LDA model

Grid Search for solver

In [6]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

model = LinearDiscriminantAnalysis()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# define grid
grid = dict()
grid['solver'] = ['svd', 'lsqr', 'eigen']

# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)

# perform the search
results = search.fit(X_train, Y_train)

# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
Mean Accuracy: 0.563
Config: {'solver': 'lsqr'}
```

Grid Search for Shrinkage parameter

In [7]:

```
from numpy import arange
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# define model
model = LinearDiscriminantAnalysis(solver='lsqr')

# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# define grid
grid = dict()
grid['shrinkage'] = arange(0, 1, 0.01)

# define search
search = GridSearchCV(model, grid, scoring='accuracy', cv=cv, n_jobs=-1)

# perform the search
results = search.fit(X_train, Y_train)

# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)
```

```
Mean Accuracy: 0.563
Config: {'shrinkage': 0.0}
```

Featue Extraction and Modeling Pipeline

In [8]:

```
# Create a pipeline that extracts features from the data then creates a model
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# create feature union
features = []
features.append(('pca', PCA(n_components=8)))
#features.append(('select_best', SelectKBest(k=9)))
feature_union = FeatureUnion(features)

#define test harness
kfold = KFold(n_splits=10, random_state=100, shuffle=True)

# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
#estimators.append(('standardize', StandardScaler()))
#estimators.append(('lda', LinearDiscriminantAnalysis(solver='lsqr', shrinkage =
0.0)))
estimators.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(estimators)

# evaluate pipeline

results = cross_val_score(model, X_train, Y_train, cv=kfold)
print("Model Cross Validation result: %f (%f)" % (results.mean(), results.std
()))
```

```
Model Cross Validation result: 0.571944 (0.037023)
```

Results proved slightly better with default settings.

Voting Ensemble

In [9]:

```
# Create a pipeline that extracts features from the data then creates a model
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier

#define test harness
kfold = KFold(n_splits=10, random_state=100, shuffle=True)

# create pipeline
estimators_C = []
estimators_C.append(('lda', LinearDiscriminantAnalysis()))
estimators_C.append(('LGBM_C', LGBMClassifier()))
estimators_C.append(('RF_C', RandomForestClassifier()))

# create the ensemble model
ensemble_C = VotingClassifier(estimators_C, voting = 'hard')
results_C = cross_val_score(ensemble_C, X_train, Y_train, cv=kfold)
print("Accuracy: %.3f (%.3f)" % (results_C.mean(), results_C.std()))
```

```
Accuracy: 0.573 (0.046)
```

The best results is attained by the single LDA model