Titanic_Jupyter

March 6, 2021

[11]:
```python
# -*- Project -*-
"""
Created on Sun Dec 20 12:09:18 2020

@author: Georg Z.
"""
#Read Data
import pandas as pd
titanic = pd.read_csv("train.csv", sep = ',')   #train data
test = pd.read_csv("test.csv", sep = ',')       #data to be predicted
```

[12]:
```python
#######################################################################
#Model Meta-Parmeters
cv=2
polynom_degree = 2
```

[13]:
```python
#Describe Data
titanic.head()
titanic.keys()
titanic.Survived
titanic.shape
titanic.dtypes
```

[13]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

[14]:

```
#show non-numeric object columns
obj_titanic = titanic.select_dtypes(include=['object']).copy()
obj_titanic.head()

sum_stat = round(titanic.describe(),1)
print(sum_stat.to_latex())
print(titanic)

print(titanic.dtypes.to_latex())
```

```
\begin{tabular}{lrrrrrrr}
\toprule
{} &  PassengerId &  Survived &  Pclass &    Age &  SibSp &  Parch &   Fare \\
\midrule
count &        891.0 &     891.0 &   891.0 &  714.0 &  891.0 &  891.0 &  891.0
\\
mean  &        446.0 &       0.4 &     2.3 &   29.7 &    0.5 &    0.4 &   32.2
\\
std   &        257.4 &       0.5 &     0.8 &   14.5 &    1.1 &    0.8 &   49.7
\\
min   &          1.0 &       0.0 &     1.0 &    0.4 &    0.0 &    0.0 &    0.0
\\
25\%   &        223.5 &       0.0 &     2.0 &   20.1 &    0.0 &    0.0 &    7.9
\\
50\%   &        446.0 &       0.0 &     3.0 &   28.0 &    0.0 &    0.0 &   14.5
\\
75\%   &        668.5 &       1.0 &     3.0 &   38.0 &    1.0 &    0.0 &   31.0
\\
max   &        891.0 &       1.0 &     3.0 &   80.0 &    8.0 &    6.0 &  512.3
\\
\bottomrule
\end{tabular}

     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …         …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
```

```
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 …     …    …      …
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     …               …       …    …      …
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
\begin{tabular}{ll}
\toprule
{} &        0 \\
\midrule
PassengerId &    int64 \\
Survived    &    int64 \\
Pclass      &    int64 \\
Name        &   object \\
Sex         &   object \\
Age         &  float64 \\
SibSp       &    int64 \\
Parch       &    int64 \\
Ticket      &   object \\
Fare        &  float64 \\
Cabin       &   object \\
Embarked    &   object \\
\bottomrule
\end{tabular}
```

[15]:
```
#mean of survived is more towards 0, which means that sample is imbalanced
#more people did not survive, than those who did. Eventually have to use
#stratified CV

titanic["Survived"].sum()

#Women/Men
titanic.loc[titanic.Sex == "female"]
titanic.loc[titanic.Sex == "male"]

#Women/Men survival rate
titanic.loc[titanic.Sex == "female"][titanic.Survived == 1]
titanic.loc[titanic.Sex == "male"][titanic.Survived == 1]

#round(sum(men)/len(men)*100,1)
#round(sum(women)/len(women)*100,1)
#20 of men survived, 75 of women survived
```

```
<ipython-input-15-167ecdf373d8>:12: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  titanic.loc[titanic.Sex == "female"][titanic.Survived == 1]
<ipython-input-15-167ecdf373d8>:13: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  titanic.loc[titanic.Sex == "male"][titanic.Survived == 1]
```

[15]:
```
     PassengerId  Survived  Pclass                           Name   Sex  \
17            18         1       2     Williams, Mr. Charles Eugene  male
21            22         1       2           Beesley, Mr. Lawrence  male
23            24         1       1     Sloper, Mr. William Thompson  male
36            37         1       3               Mamee, Mr. Hanna  male
55            56         1       1               Woolner, Mr. Hugh  male
..           …         …       …                              …    …
838          839         1       3                Chip, Mr. Chang  male
839          840         1       1            Marechal, Mr. Pierre  male
857          858         1       1           Daly, Mr. Peter Denis  male
869          870         1       3  Johnson, Master. Harold Theodor  male
889          890         1       1           Behr, Mr. Karl Howell  male

      Age  SibSp  Parch  Ticket     Fare Cabin Embarked
17    NaN      0      0  244373  13.0000   NaN        S
21   34.0      0      0  248698  13.0000   D56        S
23   28.0      0      0  113788  35.5000    A6        S
36    NaN      0      0    2677   7.2292   NaN        C
55    NaN      0      0   19947  35.5000   C52        S
..    …     …     …     …      …     …      …
838  32.0      0      0    1601  56.4958   NaN        S
839   NaN      0      0   11774  29.7000   C47        C
```

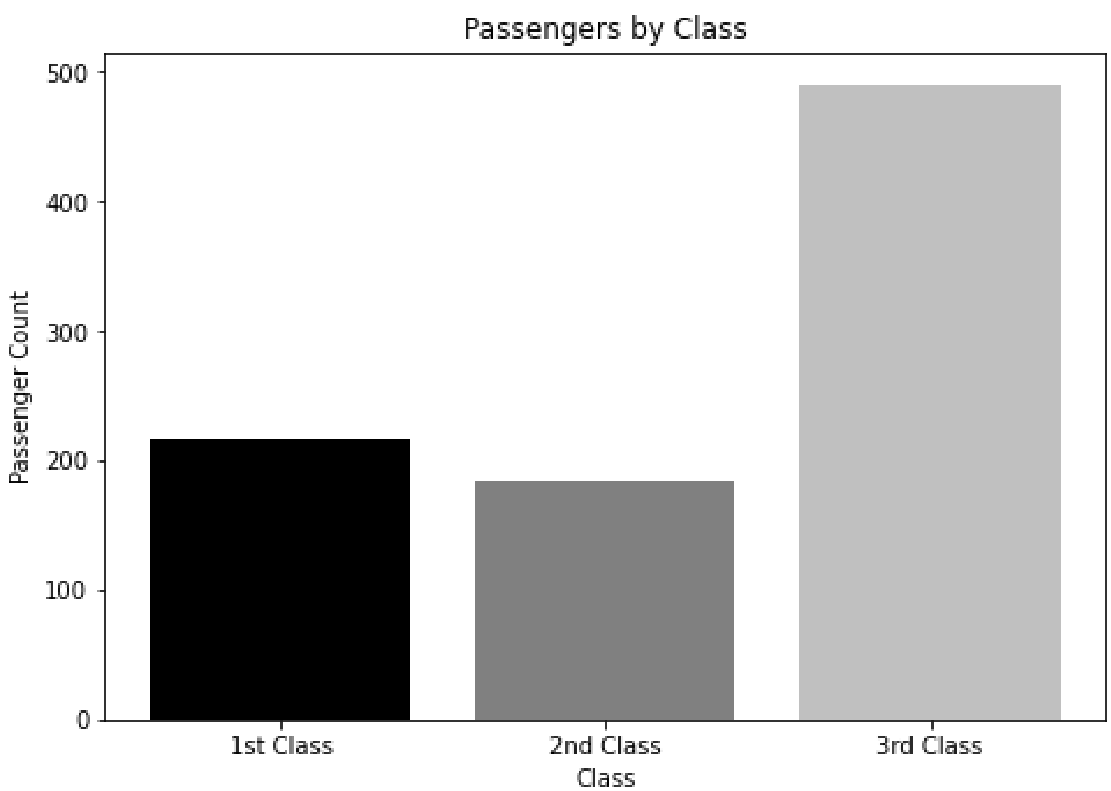
```
857  51.0      0      0  113055  26.5500   E17        S
869   4.0      1      1  347742  11.1333   NaN        S
889  26.0      0      0  111369  30.0000  C148        C

[109 rows x 12 columns]
```

[16]:
```
#ticket class and survival rate
passengers = [len(titanic.loc[titanic.Pclass == 1]), len(titanic.loc[titanic.
 ↪Pclass == 2]),
len(titanic.loc[titanic.Pclass == 3])]
```

[17]:
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
classes = ['1st Class', '2nd Class', '3rd Class']
ax.bar(classes,passengers, color=['black', 'grey', 'silver'])
plt.title('Passengers by Class')
plt.xlabel('Class')
plt.ylabel('Passenger Count')
plt.show()
```



[18]:

```
passengers_surv_rate = [len(titanic.loc[titanic.Pclass == 1][titanic.Survived␣
↪== 1])/len(titanic.loc[titanic.Pclass == 1])*100,
                   len(titanic.loc[titanic.Pclass == 2][titanic.Survived == 1])/
↪len(titanic.loc[titanic.Pclass == 2])*100,
                   len(titanic.loc[titanic.Pclass == 3][titanic.Survived == 1])/
↪len(titanic.loc[titanic.Pclass == 3])*100]

fig2 = plt.figure()
ax = fig2.add_axes([0,0,1,1])
classes = ['1st Class', '2nd Class', '3rd Class']
ax.bar(classes,passengers_surv_rate, color=['black', 'grey', 'silver'])
plt.title('% Survived by Class')
plt.xlabel('Class')
plt.ylabel('% survived')
plt.show()
```

```
<ipython-input-18-f83440d09da8>:1: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  passengers_surv_rate = [len(titanic.loc[titanic.Pclass == 1][titanic.Survived
== 1])/len(titanic.loc[titanic.Pclass == 1])*100,
<ipython-input-18-f83440d09da8>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  len(titanic.loc[titanic.Pclass == 2][titanic.Survived ==
1])/len(titanic.loc[titanic.Pclass == 2])*100,
<ipython-input-18-f83440d09da8>:3: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  len(titanic.loc[titanic.Pclass == 3][titanic.Survived ==
1])/len(titanic.loc[titanic.Pclass == 3])*100]
```

[19]:
```
#age and survival
titanic["Age"].max()
titanic["Age"].min()
titanic["Age"].describe()


#print('Train Data Info')
#print(titanic.info())
```

[19]:
```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

[20]:
```
##############################################################
#save passenger ids to assign them to predictions later
```

```python
def save_id(data):
    id = data['PassengerId']
    return id

#Observe Missing Values
ids_test = save_id(test)
test.isna().sum()
test.isna().sum()
```

[20]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

[21]:
```python
#Pre-Process Data
#observe missing values
pd.isnull(titanic).sum()
pd.isnull(test).sum()

#create bins for fare
# ============================================================================
# def create_bins(x):
#     range_value = [0,0,0,0]
#     if 0 < x < 50:
#         range_value[0] = 1
#     elif 50 < x < 100:
#         range_value[1] = 1
#     elif 100 < x < 250:
#         range_value[2] = 1
#     else:
#         range_value[3] = 1
#     return range_value
# ============================================================================

def pre_process (data):
    #Drop Variables that don't contribute to outcome
    data = data.drop(columns=["PassengerId", "Name","Ticket","Cabin"])
    #only include rows in which embarked is not missing
    #titanic loses two observations, test loses none
```

```
    data = data.loc[data.Embarked.notna()]
    data = data.interpolate()
# =============================================================================
#     #bin age
#     import numpy as np
#     bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]
#     labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young␣
↪Adult', 'Adult', 'Senior']
#     data['AgeGroup'] = pd.cut(data["Age"], bins, labels = labels)
#     #bin fare
#     data["Fare_lt_50"], data["Fare_50_100"], data["Fare_100_250"],␣
↪data["Fare_gt_250"] = zip(*data["Fare"].map(create_bins))
#     data = data.drop(columns=["Fare"])
# =============================================================================
    return data

titanic = pre_process(titanic)
test = pre_process(test)
pd.isnull(titanic).sum()
pd.isnull(test).sum()
```

[21]:
```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

[22]:
```
######################################################################
#One Hot Encoding
def one_hot (data):
    from sklearn.preprocessing import OneHotEncoder
    enc = OneHotEncoder()

    #One hot encode Pclass
    Pclass = enc.fit_transform(data[["Pclass"]]).toarray()
    Pclass = pd.DataFrame(Pclass)
    Pclass.rename(columns={0: "1st Class", 1: "2nd Class", 2: "3rd Class"},␣
↪inplace=True)

    #One hot encode Sex
    sex = enc.fit_transform(data[["Sex"]]).toarray()
    sex = pd.DataFrame(sex)
    sex.rename(columns={0: "female", 1: "male"}, inplace=True)
```

```
    #One hot encode Embarked
    embarked = enc.fit_transform(data[["Embarked"]]).toarray()
    embarked = pd.DataFrame(embarked)
    embarked.rename(columns={0: "C", 1: "Q", 2: "S"}, inplace=True)

# ==============================================================================
#     #One hote encode Age
#     age = enc.fit_transform(data[["AgeGroup"]]).toarray()
#     age = pd.DataFrame(age)
#     age.rename(columns={0: "Baby", 1: "Child", 2: "Teenager", 3: "Student", 4:
 ↪ "Young Adult", 5: "Adult", 6: "Senior"}, inplace=True)
#
# ==============================================================================
    #reset index because pd.concat mismatches indexes and creates nans
    data.reset_index(drop=True, inplace=True)
    sex.reset_index(drop=True, inplace=True)

    #Add recoded columns to data frame
    data = pd.concat([data, pd.DataFrame(sex)], axis=1)
    data = pd.concat([data, pd.DataFrame(embarked)], axis=1)
    #data = pd.concat([data, pd.DataFrame(age)], axis=1)
    data = pd.concat([data, pd.DataFrame(Pclass)], axis=1)

    #Drop old columns
    #data = data.drop(columns=["Sex","Embarked","AgeGroup","Age","Pclass"])
    data = data.drop(columns=["Sex","Embarked","Pclass"])
    return data

titanic = one_hot(titanic)
test = one_hot(test)
```

[23]:
```
#last check for missing values: no missing values
pd.isnull(titanic).sum()
pd.isnull(test).sum()
titanic.shape
test.shape
```

[23]:
```
(418, 12)
```

[24]:
```
################################################################################
#min max scaler chosen because all other are one hot encoded between 0,1
def scale_min_max (data):
    #Scale some features
    from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()
    scaler.fit(data[["SibSp"]])
    data[["SibSp"]] = scaler.transform(data[["SibSp"]])
```

```
    #Parch (Parents)
    scaler.fit(data[["Parch"]])
    data[["Parch"]] = scaler.transform(data[["Parch"]])
    #Scale Age
    scaler.fit(data[["Age"]])
    data[["Age"]] = scaler.transform(data[["Age"]])
    #Scale Fare
    scaler.fit(data[["Fare"]])
    data[["Fare"]] = scaler.transform(data[["Fare"]])
    return data

titanic = scale_min_max(titanic)
test = scale_min_max(test)

#look at summary statistics of scaled data
titanic.max(axis=0)
titanic.min(axis=0)

#explain .scale command(which scaler is it: Standard Scaler)
#not worried about scaling problems or outliers
#possibly outliers by Fare


sum_stat = round(titanic.describe(),1)
print(sum_stat)
print(sum_stat.to_latex())

print(titanic.dtypes.to_latex())


#import sys
#sys.exit("Stop here")
############################################################
```

```
       Survived    Age  SibSp  Parch   Fare  female   male      C      Q  \
count     889.0  889.0  889.0  889.0  889.0   889.0  889.0  889.0  889.0
mean        0.4    0.4    0.1    0.1    0.1     0.4    0.6    0.2    0.1
std         0.5    0.2    0.1    0.1    0.1     0.5    0.5    0.4    0.3
min         0.0    0.0    0.0    0.0    0.0     0.0    0.0    0.0    0.0
25%         0.0    0.3    0.0    0.0    0.0     0.0    0.0    0.0    0.0
50%         0.0    0.4    0.0    0.0    0.0     0.0    1.0    0.0    0.0
75%         1.0    0.5    0.1    0.0    0.1     1.0    1.0    0.0    0.0
max         1.0    1.0    1.0    1.0    1.0     1.0    1.0    1.0    1.0

           S  1st Class  2nd Class  3rd Class
count  889.0      889.0      889.0      889.0
```

```
mean     0.7        0.2        0.2        0.6
std      0.4        0.4        0.4        0.5
min      0.0        0.0        0.0        0.0
25%      0.0        0.0        0.0        0.0
50%      1.0        0.0        0.0        1.0
75%      1.0        0.0        0.0        1.0
max      1.0        1.0        1.0        1.0
\begin{tabular}{lrrrrrrrrrrrrr}
\toprule
{} &  Survived &    Age &  SibSp &  Parch &   Fare &  female &   male &      C &
Q &      S &  1st Class &  2nd Class &  3rd Class \\
\midrule
count &     889.0 &  889.0 &  889.0 &  889.0 &  889.0 &   889.0 &  889.0 &
889.0 &  889.0 &  889.0 &      889.0 &      889.0 &      889.0 \\
mean  &       0.4 &    0.4 &    0.1 &    0.1 &    0.1 &     0.4 &    0.6 &
0.2 &    0.1 &    0.7 &        0.2 &        0.2 &        0.6 \\
std   &       0.5 &    0.2 &    0.1 &    0.1 &    0.1 &     0.5 &    0.5 &
0.4 &    0.3 &    0.4 &        0.4 &        0.4 &        0.5 \\
min   &       0.0 &    0.0 &    0.0 &    0.0 &    0.0 &     0.0 &    0.0 &
0.0 &    0.0 &    0.0 &        0.0 &        0.0 &        0.0 \\
25\%   &       0.0 &    0.3 &    0.0 &    0.0 &    0.0 &     0.0 &    0.0 &
0.0 &    0.0 &    0.0 &        0.0 &        0.0 &        0.0 \\
50\%   &       0.0 &    0.4 &    0.0 &    0.0 &    0.0 &     0.0 &    1.0 &
0.0 &    0.0 &    1.0 &        0.0 &        0.0 &        1.0 \\
75\%   &       1.0 &    0.5 &    0.1 &    0.0 &    0.1 &     1.0 &    1.0 &
0.0 &    0.0 &    1.0 &        0.0 &        0.0 &        1.0 \\
max   &       1.0 &    1.0 &    1.0 &    1.0 &    1.0 &     1.0 &    1.0 &
1.0 &    1.0 &    1.0 &        1.0 &        1.0 &        1.0 \\
\bottomrule
\end{tabular}

\begin{tabular}{ll}
\toprule
{} &        0 \\
\midrule
Survived  &    int64 \\
Age       &  float64 \\
SibSp     &  float64 \\
Parch     &  float64 \\
Fare      &  float64 \\
female    &  float64 \\
male      &  float64 \\
C         &  float64 \\
Q         &  float64 \\
S         &  float64 \\
1st Class &  float64 \\
2nd Class &  float64 \\
3rd Class &  float64 \\
```

```
\bottomrule
\end{tabular}
```

[25]:
```
#Higher-Order Polynomials

def feat_engin (data):
    from sklearn.preprocessing import PolynomialFeatures
    poly_transformer = PolynomialFeatures(degree=polynom_degree,␣
 ↪interaction_only=True, include_bias=True)
    data_poly = poly_transformer.fit_transform(data.loc[:, data.columns !=␣
 ↪'Survived'])

    #Add "Survived" column
    data_poly = pd.DataFrame(data_poly)
    if len(data) > len(test):     #only the train set receives a target column
        data_poly = pd.concat([data_poly, pd.DataFrame(data["Survived"])],␣
 ↪axis=1)
    else:
        data_poly = data_poly #the test set does not have a target column␣
 ↪"Survived"
    return data_poly

titanic_poly = feat_engin(titanic)
test_poly = feat_engin(test)

######################################################
```

[26]:
```
#Grid Search
###########################
#Import Model APIs
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
#from sklearn.calibration import CalibratedClassifierCV
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier

#Define Models and Parameters
#kNN
param_knn = {'n_neighbors': [3,5,11,19],
             'weights': ['uniform','distance'],
             'metric': ['eucldean', 'manhattan']
```

```
              }
kNN = KNeighborsClassifier()

#SVC
param_svc = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
             'gamma': [0.001, 0.01, 0.1, 1, 10, 100],
              'random_state' : [1],
              'probability' : [True]
              }
svc = SVC()

#LogReg
import numpy as np
param_logreg = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 5),
    'max_iter' : [1000],
    'random_state' : [1]
    }
]
logreg = LogisticRegression()

#Forest
#left to default parameters because it takes too long
param_forest = [
    {'random_state' : [1],
 'min_samples_leaf': [1, 2],
 'min_samples_split': [2, 5]
    }
]
#empty because defaults work very good
forest = RandomForestClassifier()

#Grad Boosting
#left to default parameters because it takes too long
param_grbt = [
    {'random_state' : [1], 'learning_rate':[0.1,0.01,0.001],
     'max_depth':[2,3,4,5]
    }
]
#empty because defaults work very good
grbt = GradientBoostingClassifier()

#MLP
param_mlp = [
    {
    'activation': ['relu'],
```

```
    'solver': ['adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
    'max_iter': [1000]
    }
]
mlp = MLPClassifier()

#Voting Classifier
estimators=[("knn", kNN), ("log_reg", logreg),
            ("rf", forest), ("gbrt", grbt), ("svc", svc), ("mlp", mlp)]
#create our voting classifier, inputting all models
ensemble = VotingClassifier(estimators, voting="hard")
param_ensemble = [
    {
    }
]
```

[27]:

```
#Define Tuning Function
def tune(model, param_grid):
    grid_search = GridSearchCV(model, param_grid, cv=cv, verbose=False)
    return grid_search
```

[28]:

```
#Define Scoring Function
def score(data, model_to_tune, parameters):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        #assess train and test fit of test set provided by Kaggle
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(columns=["Survived"]),
                                                            data['Survived'],␣
 ↪stratify=data['Survived'], random_state=1)
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import StratifiedKFold
        Stkfold = StratifiedKFold(n_splits=3, shuffle=True)
        model = tune(model_to_tune, parameters)
        train = round(cross_val_score(model, X_train, y_train, cv=Stkfold).
 ↪mean(),2)
        test = round(cross_val_score(model, X_test, y_test, cv=Stkfold).
 ↪mean(),2)
        train_txt = "{model_to_tune} Training accuracy:"
        test_txt = "{model_to_tune} Testing accuracy:"
        print(train_txt.format(model_to_tune = str(model_to_tune)[:10]), train)
        print(test_txt.format(model_to_tune = str(model_to_tune)[:10]), test)
    return
```

[29]:
```python
#Model Evaluation
##############################################################################

#Fit and Score Models using Accuracy
print(" ")
print(" ")
print("Simple Model: Parts")
print(" ")
score(titanic, svc, param_svc)
score(titanic, kNN, param_knn)
score(titanic, logreg, param_logreg)
score(titanic, forest, param_forest)
score(titanic, grbt, param_grbt)
score(titanic, mlp, param_mlp)
print(" ")
print("Simple Model Combined:")
score(titanic, ensemble, param_ensemble)
print(" ")
```

```
Simple Model: Parts

SVC() Training accuracy: 0.79
SVC() Testing accuracy: 0.82
KNeighbors Training accuracy: 0.76
KNeighbors Testing accuracy: 0.81
LogisticRe Training accuracy: 0.78
LogisticRe Testing accuracy: 0.78
RandomFore Training accuracy: 0.8
RandomFore Testing accuracy: 0.78
GradientBo Training accuracy: 0.79
GradientBo Testing accuracy: 0.78
MLPClassif Training accuracy: 0.78
MLPClassif Testing accuracy: 0.82

Simple Model Combined:
VotingClas Training accuracy: 0.79
VotingClas Testing accuracy: 0.82
```

[30]:
```python
############################################################################
#Fit on train set predict on test set and evaluate predictions with CMs

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(titanic.
 ↪drop(columns=["Survived"]), titanic['Survived'],␣
 ↪stratify=titanic['Survived'], random_state=1)


def eval(model, parameters):
    #predict test set, resulting from split
    #from sklearn.model_selection import train_test_split
    #X_train, X_test, y_train, y_test = train_test_split(data.
 ↪drop(columns=["Survived"]), data['Survived'], stratify=data['Survived'],␣
 ↪random_state=1)
    #tuned_model = tune(model, parameters)

    fitted_model = model.fit(X_train, y_train)
    predictions = fitted_model.predict(X_test)

    #Confusion Matrix
    from sklearn.metrics import confusion_matrix
    print('Confusion matrix:')
    mat1 = confusion_matrix(y_test, predictions)
    print(mat1)

    #Sensitivity
    #from sklearn.metrics import recall_score
    #print(recall_score(y_test, predictions))

    sens = mat1[1,1]/(mat1[1,0]+mat1[1,1])
    print('Sensitivity:', round(sens,2))
    spec = mat1[0,0]/(mat1[0,1]+mat1[0,0])
    print('Specificity:', round(spec,2))
    return predictions
```

[31]:

```
print(" ")
print("Ensemble")
pred_ensemble = eval(ensemble, param_ensemble)
print(" ")
print("Random Forest")
pred_forest = eval(forest, param_forest)
print(" ")
print("Log Reg")
pred_logreg = eval(logreg, param_logreg)
print(" ")
print("KNN")
pred_kNN = eval(kNN, param_knn)
print(" ")
print("Grbt")
pred_grbt = eval(grbt, param_grbt)
```

```python
print(" ")
print("MLP")
pred_mlp = eval(mlp, param_mlp)
print(" ")
print("SVC")
pred_svc = eval(svc, param_svc)
```

```
Ensemble
```

```
C:\Users\zhele\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
Confusion matrix:
[[129   9]
 [ 27  58]]
Sensitivity: 0.68
Specificity: 0.93

Random Forest
Confusion matrix:
[[120  18]
 [ 21  64]]
Sensitivity: 0.75
Specificity: 0.87

Log Reg
Confusion matrix:
[[119  19]
 [ 24  61]]
Sensitivity: 0.72
Specificity: 0.86

KNN
Confusion matrix:
[[124  14]
 [ 28  57]]
Sensitivity: 0.67
Specificity: 0.9

Grbt
Confusion matrix:
[[119  19]
 [ 26  59]]
Sensitivity: 0.69
```

```
Specificity: 0.86

MLP
Confusion matrix:
[[126  12]
 [ 22  63]]
Sensitivity: 0.74
Specificity: 0.91

SVC
Confusion matrix:
[[132   6]
 [ 32  53]]
Sensitivity: 0.62
Specificity: 0.96
```

```
C:\Users\zhele\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

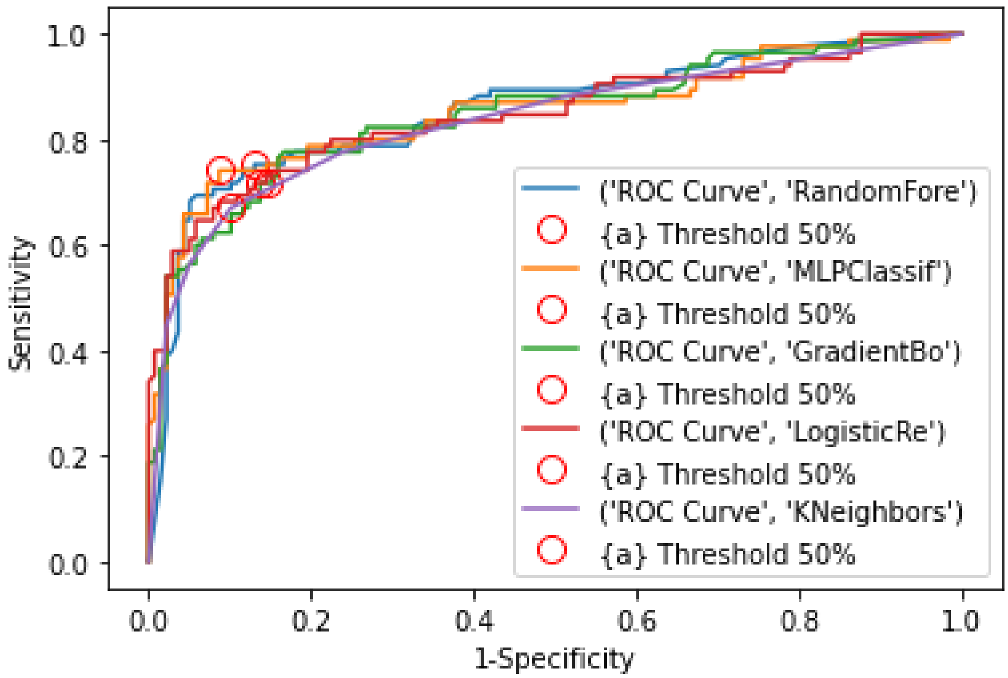
[32]:
```
#Optimize Discrimination Threshhold = optimize specificits/sensitivity trade-off

#ROC Curve
def roc(models):
    from sklearn.metrics import roc_curve
    for a in models:
        fpr, tpr, thresholds = roc_curve(y_test, a.predict_proba(X_test)[:, 1])
        label = ("ROC Curve", str(a)[:10])
        plt.plot(fpr, tpr, label=label)
        plt.xlabel("1-Specificity")
        plt.ylabel("Sensitivity")
        default = np.argmin(np.abs(thresholds - 0.5))
        plt.plot(fpr[default], tpr[default], 'o', markersize=10, label="{a}␣
↪Threshold 50%", fillstyle="none", c='r')

        plt.legend(loc=4)
    return

models1=[forest, mlp]
roc(models1)
models2=[grbt, logreg, kNN]
roc(models2)
print("SVC has no Proba Eval Possibility")
```

```
SVC has no Proba Eval Possibility
```



[33]:
```
#Summary: Models are in good shape: no need for adjustment
#the Threshholds of the two and the three curves are all on the 45 degree axis
#therefore there is no oportunity for improvement, because Threschold on the␣
↪flat side of the curve
#if try to improve will sacrifice specificity
#lower prediction accuracy is due to other reasons than to Threshold selection
#one can see that MLP and RF are the best models
#perfect model ROC is as high sensitivity and as high specificity as possible
```

[34]:
```
#Because ROC curves cross it is not possible to rank the models
#therefore one can use a single evaluation measure to rankt the models
#AUC is the integram of ROC
#The higher it is, the better a model performs in terms of both sensitivity/
↪specificity
#ranges between 0.5 and 1
#random guessing would be an AUC of 0.5
#AUC is insensitive to class proportions, due class inbalance
#
#AUC
def auc(models):
    from sklearn.metrics import roc_auc_score
    for a in models:
        auc = roc_auc_score(y_test, a.predict_proba(X_test)[:, 1])
        print("AUC for", str(a)[:10], round(auc,2))
```

```
    return

models=[forest, grbt, logreg, kNN, mlp]
auc(models)
#random forest is the best model
```

```
AUC for RandomFore 0.85
AUC for GradientBo 0.85
AUC for LogisticRe 0.85
AUC for KNeighbors 0.84
AUC for MLPClassif 0.85
```

[35]:
```
#Precision
def prec(predictions, models):
    from sklearn.metrics import precision_score
    for a, b in zip(predictions, models):
        print('Precision:', b)
        print(round(precision_score(y_test, a),2))
    return

predictions = [pred_ensemble, pred_logreg, pred_svc, pred_mlp, pred_forest,␣
 ↪pred_grbt]
prec(predictions, models)
```

```
Precision: RandomForestClassifier()
0.87
Precision: GradientBoostingClassifier()
0.76
Precision: LogisticRegression()
0.9
Precision: KNeighborsClassifier()
0.84
Precision: MLPClassifier()
0.78
```

[36]:
```
#F Score
#sensitivity and precision combined
def f_score(predictions, models):
    from sklearn.metrics import f1_score
    for a, b in zip(predictions, models):
        print('Precision:', b)
        print(round(f1_score(y_test, a),2))
    return

f_score(predictions, models)
```

```
Precision: RandomForestClassifier()
0.76
```

```
Precision: GradientBoostingClassifier()
0.74
Precision: LogisticRegression()
0.74
Precision: KNeighborsClassifier()
0.79
Precision: MLPClassifier()
0.77
```

[37]:

```
#Fit Complex Model
#a complex model of squares power does not imporve performance
# ==============================================================================
# print("Complex Model: Parts")
# print(" ")
# score(titanic_poly, svc, param_svc)
# score(titanic_poly, kNN, param_knn)
# score(titanic_poly, logreg, param_logreg)
# score(titanic_poly, forest, param_forest)
# score(titanic_poly, grbt, param_grbt)
# score(titanic_poly, mlp, param_mlp)
# print(" ")
# print("Complex Model Combined:")
# score(titanic_poly, ensemble, param_ensemble)
# ==============================================================================

#import sys
```

[38]:

```
##########################################################
print(" ")
print("Predict for Kaggle Competition:")
#predict "Survived" for test set provided by Kaggle
print(" ")
print("submission file is saved in the workign directory in csv format")
print("a submission based on the simple and one based on the complex model")

def predict_kaggle_test_set(data_to_predict, data_to_fit_on, model, parameters):
    #full train set provided by kaggle is used
    #test set provided by kaggle is predicted
    #prediction is done using the voting classifier
    X_data = data_to_fit_on.drop(columns=["Survived"])
    y_data = data_to_fit_on['Survived']
    model_tuned = tune(model, parameters)
    fitted_model = model_tuned.fit(X_data, y_data)
    predictions = fitted_model.predict(data_to_predict)
    return predictions

#prediction based on simple model
```

```
predictions_simple = predict_kaggle_test_set(test, titanic, ensemble,␣
↪param_ensemble)
```

```
Predict for Kaggle Competition:

submission file is saved in the workign directory in csv format
a submission based on the simple and one based on the complex model
```

```
C:\Users\zhele\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\zhele\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\zhele\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[39]:

```
#prediction based on complex model
#predictions_complex = predict_kaggle_test_set(test_poly, titanic_poly,␣
↪ensemble)

def save_csv(predictions, name):
    #set the output as a dataframe and convert to csv file
    output = pd.DataFrame({ 'PassengerId' : ids_test, 'Survived': predictions })
    output.to_csv(f"{name}.csv", index=False)
    return

save_csv(predictions_simple, "submission_simple")
#save_csv(predictions_complex, "submission_complex")
```

[40]:

```
##########################################################################
#Comments on Titanic:
#Problem: train test accuracy 87%, validation set 78%. Model
#doesn't generalize good. Now generalizes better.
#see how the classes are represented in train test, and hold out data
#is titanic set class-imbalanced? (may need other metric as accuracy score)
#yes

#Submission after correction of expectations using cv.
```

[41]:

```
#####################################################################
#Graphs/Tables

#discriptive
#ticket class and survival rate
#age and survival
#########################################################################

#Inference
#use stats models to show coefficients of logistic and linear reg
```