Linear Regression

Predicting time since ketamine administration

In [1]:
```
# packages
import os
import pandas as pd
import math
from scipy import io
import numpy as np
from numpy import squeeze
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
```

Load in data and perform checks

In [2]:
```
allData = pd.read_csv('postKetamineTable.csv')
```

In [3]:
```
allData.keys()
```

Out[3]:
```
Index(['animalName', 'sessionDate', 'trialNum', 'totalCellNum', 'gender',
       'genotype', 'weight_g', 'ketamine_day', 'correlationScore',
       'lickAccuracy', 'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth',
       'timeSinceKetamine', 'ketamineAdministered'],
      dtype='object')
```

In [4]:
```
# Check size information
print("num_cols =",len(allData.keys()))
print("num_rows =",len(allData))

# Check for duplicate rows
print("num_dup =",np.sum(pd.DataFrame.duplicated(allData)))
```

```
num_cols = 19
num_rows = 5000
num_dup = 0
```

In [5]:
```
# Check for NaNs and see where they are coming from
np.sum(pd.isna(allData))
```

Out[5]:
```
animalName               0
sessionDate              0
trialNum                 0
totalCellNum             0
gender                   0
genotype                 0
weight_g                 0
ketamine_day             0
correlationScore         0
lickAccuracy             0
lickNumber               0
avgFR                    0
avgSingleCellVariance    0
varianceFR               5
avgTrialSpeed            0
varianceSpeed            0
medianCellDepth          0
timeSinceKetamine        0
ketamineAdministered     0
dtype: int64
```

In [6]:
```
# Remove any rows with nans
allDataNN = pd.DataFrame.dropna(allData,'index')
print("After Drop NaN")
print("num_rows =",len(allDataNN))
```

```
After Drop NaN
num_rows = 4995
```

In [7]:
```
ketBool = allDataNN['ketamineAdministered']
timeSinceKetamine = allDataNN['timeSinceKetamine']
sessionDate = allDataNN['sessionDate']
trialNum = allDataNN['trialNum']
neuralData = allDataNN[['animalName', 'totalCellNum',
       'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy',
       'lickNumber', 'avgFR', 'avgSingleCellVariance',
       'varianceFR', 'avgTrialSpeed', 'varianceSpeed',
       'medianCellDepth']]
```

In [8]:
```
# Convert categorical columns
le = LabelEncoder()
neuralData_LE = neuralData.copy()
neuralData_LE['animalName'] = le.fit_transform(neuralData_LE['animalName'])
neuralData_LE['gender'] = le.fit_transform(neuralData_LE['gender'])
neuralData_LE['genotype'] = le.fit_transform(neuralData_LE['genotype'])
features = list(neuralData_LE.keys())
```

In [9]:
```
# Standardize data
stdNeuralData = StandardScaler().fit_transform(neuralData_LE)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype in
t64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [10]:
```
# Split off test set for later
X, X_ho, y, y_ho = train_test_split(stdNeuralData,timeSinceKetamine.values.ravel(), test_size=0.2, random_state = 2019)
```

In [11]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [12]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
```

In [13]:
```
print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)
```

```
Intercept:  1881.3040918494605
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [  89.
37637717  -48.31423738 -388.77287316  233.85776474  580.47450308
 -334.13598052  261.54986522  304.54847766   24.61109336 -597.46522638
  262.84684555  244.33634822 -395.38514629  314.91671046 -387.04573926]
RMSE:  1879.9189130710613
R2: 0.2643830933222706
```

In [14]:
```
scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
0.14169716149281847
```

In [15]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X_train[i],y_
train[i])
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1883.6861818752764
Average Scaled RMSE across Folds: 0.1419811159189515
```

Now let's try with some second order interaction terms

In [16]:
```
AugData = neuralData_LE.copy()
AugData.keys()
```

Out[16]:
```
Index(['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
       'ketamine_day', 'correlationScore', 'lickAccuracy', 'lickNumber',
       'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
       'varianceSpeed', 'medianCellDepth'],
      dtype='object')
```

In [17]:
```
primaryF = ['correlationScore', 'lickAccuracy', 'lickNumber',
       'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed',
       'varianceSpeed']
secondaryF = ['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g',
       'ketamine_day','medianCellDepth']
```

In [18]:
```
AugData['animalNamexCorrelationScore'] = AugData['animalName']*AugData['correlationScore']
AugData['animalNamexLickAccuracy'] = AugData['animalName']*AugData['lickAccuracy']
AugData['animalNamexLickNumber'] = AugData['animalName']*AugData['lickNumber']
AugData['animalNamexAvgFR'] = AugData['animalName']*AugData['avgFR']
AugData['animalNamexAvgSingleCellVariance'] = AugData['animalName']*AugData['avgSingleCellVariance']
AugData['animalNamexVarianceFR'] = AugData['animalName']*AugData['varianceFR']
AugData['animalNamexAvgTrialSpeed'] = AugData['animalName']*AugData['avgTrialSpeed']
AugData['animalNamexVarianceSpeed'] = AugData['animalName']*AugData['varianceSpeed']
```

In [19]:
```
AugData['totalCellNumxCorrelationScore'] = AugData['totalCellNum']*AugData['correlationScore']
AugData['totalCellNumxLickAccuracy'] = AugData['totalCellNum']*AugData['lickAccuracy']
AugData['totalCellNumxLickNumber'] = AugData['totalCellNum']*AugData['lickNumber']
AugData['totalCellNumxAvgFR'] = AugData['totalCellNum']*AugData['avgFR']
AugData['totalCellNumxAvgSingleCellVariance'] = AugData['totalCellNum']*AugData['avgSingleCellVariance']
AugData['totalCellNumxVarianceFR'] = AugData['totalCellNum']*AugData['varianceFR']
AugData['totalCellNumxAvgTrialSpeed'] = AugData['totalCellNum']*AugData['avgTrialSpeed']
AugData['totalCellNumxVarianceSpeed'] = AugData['totalCellNum']*AugData['varianceSpeed']
```

In [20]:
```
AugData['genderxCorrelationScore'] = AugData['gender']*AugData['correlationScore']
AugData['genderxLickAccuracy'] = AugData['gender']*AugData['lickAccuracy']
AugData['genderxLickNumber'] = AugData['gender']*AugData['lickNumber']
AugData['genderxAvgFR'] = AugData['gender']*AugData['avgFR']
AugData['genderxAvgSingleCellVariance'] = AugData['gender']*AugData['avgSingleCellVariance']
AugData['genderxVarianceFR'] = AugData['gender']*AugData['varianceFR']
AugData['genderxAvgTrialSpeed'] = AugData['gender']*AugData['avgTrialSpeed']
AugData['genderxVarianceSpeed'] = AugData['gender']*AugData['varianceSpeed']
```

In [21]:
```
AugData['genotypexCorrelationScore'] = AugData['genotype']*AugData['correlationScore']
AugData['genotypexLickAccuracy'] = AugData['genotype']*AugData['lickAccuracy']
AugData['genotypexLickNumber'] = AugData['genotype']*AugData['lickNumber']
AugData['genotypexAvgFR'] = AugData['genotype']*AugData['avgFR']
AugData['genotypexAvgSingleCellVariance'] = AugData['genotype']*AugData['avgSingleCellVariance']
AugData['genotypexVarianceFR'] = AugData['genotype']*AugData['varianceFR']
AugData['genotypexAvgTrialSpeed'] = AugData['genotype']*AugData['avgTrialSpeed']
AugData['genotypexVarianceSpeed'] = AugData['genotype']*AugData['varianceSpeed']
```

In [22]:
```
AugData['weight_gxCorrelationScore'] = AugData['weight_g']*AugData['correlationScore']
AugData['weight_gxLickAccuracy'] = AugData['weight_g']*AugData['lickAccuracy']
AugData['weight_gxLickNumber'] = AugData['weight_g']*AugData['lickNumber']
AugData['weight_gxAvgFR'] = AugData['weight_g']*AugData['avgFR']
AugData['weight_gxAvgSingleCellVariance'] = AugData['weight_g']*AugData['avgSingleCellVariance']
AugData['weight_gxVarianceFR'] = AugData['weight_g']*AugData['varianceFR']
AugData['weight_gxAvgTrialSpeed'] = AugData['weight_g']*AugData['avgTrialSpeed']
AugData['weight_gxVarianceSpeed'] = AugData['weight_g']*AugData['varianceSpeed']
```

In [23]:
```
AugData['ketamine_dayxCorrelationScore'] = AugData['ketamine_day']*AugData['correlationScore']
AugData['ketamine_dayxLickAccuracy'] = AugData['ketamine_day']*AugData['lickAccuracy']
AugData['ketamine_dayxLickNumber'] = AugData['ketamine_day']*AugData['lickNumber']
AugData['ketamine_dayxAvgFR'] = AugData['ketamine_day']*AugData['avgFR']
AugData['ketamine_dayxAvgSingleCellVariance'] = AugData['ketamine_day']*AugData['avgSingleCellVariance']
AugData['ketamine_dayxVarianceFR'] = AugData['ketamine_day']*AugData['varianceFR']
AugData['ketamine_dayxAvgTrialSpeed'] = AugData['ketamine_day']*AugData['avgTrialSpeed']
AugData['ketamine_dayxVarianceSpeed'] = AugData['ketamine_day']*AugData['varianceSpeed']
```

In [24]:
```
AugData['medianCellDepthxCorrelationScore'] = AugData['medianCellDepth']*AugData['correlationScore']
AugData['medianCellDepthxLickAccuracy'] = AugData['medianCellDepth']*AugData['lickAccuracy']
AugData['medianCellDepthxLickNumber'] = AugData['medianCellDepth']*AugData['lickNumber']
AugData['medianCellDepthxAvgFR'] = AugData['medianCellDepth']*AugData['avgFR']
AugData['medianCellDepthxAvgSingleCellVariance'] = AugData['medianCellDepth']*AugData['avgSingleCellVariance']
AugData['medianCellDepthxVarianceFR'] = AugData['medianCellDepth']*AugData['varianceFR']
AugData['medianCellDepthxAvgTrialSpeed'] = AugData['medianCellDepth']*AugData['avgTrialSpeed']
AugData['medianCellDepthxVarianceSpeed'] = AugData['medianCellDepth']*AugData['varianceSpeed']
```

In [25]:
```
# Standardize data
stdNeuralDataAug = StandardScaler().fit_transform(AugData)
```

```
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/browne/anaconda3/lib/python3.7/site-packages/sklearn/base.py:462: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [26]:
```
# Split off test set for later
X, X_ho, y, y_ho = train_test_split(stdNeuralDataAug,timeSinceKetamine.values.ravel(), test_size=0.2, random_state=2019
)
```

In [27]:
```
# Split for cross validation, use 10 folds
num_folds = 10
XA = np.array(X)
yA = np.array(y)
X_train = []
X_test = []
y_train = []
y_test = []
kf = KFold(n_splits=num_folds)
for train_index, test_index in kf.split(XA, yA):
    X_train.append(XA[train_index])
    X_test.append(XA[test_index])
    y_train.append(yA[train_index])
    y_test.append(yA[test_index])
```

In [28]:
```
# Run basic linreg model on full train set, check performance against train
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)

print("Intercept: ",model.intercept_)
print(features,model.coef_)
#print(model.coef_)
y_pred = model.predict(X)
rmse = np.sqrt(mean_squared_error(y,y_pred))
r2 = r2_score(y,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
Intercept:  1882.209448651285
['animalName', 'totalCellNum', 'gender', 'genotype', 'weight_g', 'ketamine_day', 'correlationScore', 'lickAccuracy', 'l
ickNumber', 'avgFR', 'avgSingleCellVariance', 'varianceFR', 'avgTrialSpeed', 'varianceSpeed', 'medianCellDepth'] [ 1.32
202873e+03 -8.91679131e+02 -2.06782434e+03  6.59351804e+02
  7.64377410e+02 -2.08632125e+03 -5.32902729e+02  1.08121105e+03
 -1.41105840e+01 -3.37318052e+03  1.03249755e+03  1.13663558e+03
 -1.89808003e+03  7.79734372e+01 -1.59342695e+03  1.83861714e+02
  1.52829870e+02 -1.45411245e+02 -7.72091371e+02 -1.79495421e+02
  2.49282418e+02 -9.68636013e+02 -4.68015392e+02  3.69227271e+02
 -3.75314462e+02 -3.27108339e+02  3.96505917e+02  1.92277904e+02
  4.20441858e+01  3.53520147e+02  7.28198961e+01 -8.83274314e+02
  3.05970931e+02  3.55601020e+02 -1.86602325e+03  3.77074704e+03
 -6.18993573e+02  1.38501951e+03 -3.65147648e+02 -9.43761107e+01
  1.54665444e+02  4.59137890e+01 -1.69314230e+03  2.34941560e+03
 -7.38502618e+02  1.29824145e+02 -3.17202563e+02  8.41440984e+02
 -6.30467051e+02  3.52670997e+00  1.74618314e+03 -5.32068004e+03
  1.12276741e+03  7.83081844e+02  9.69140010e+02  3.00707198e+02
 -1.99411562e+02  6.39908126e+01  2.68099617e+03 -5.74977686e+02
 -5.57100624e+02  8.75358993e+02 -5.86782274e+02 -4.18856382e+01
 -2.18915645e+02  2.33213773e+02  1.21191515e+03  2.21624222e+03
 -6.93099690e+02  1.13823488e+02  4.27475283e+02]
RMSE:  1513.3468605647722
R2: 0.5232942086644587
0.11406712970709422
```

In [29]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1511.7573978732548
Average Scaled RMSE across Folds: 0.11394732541653638
```

In [31]:
```
rmse_cv = []
for i in range(0,num_folds):
    model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
    y_pred = model.predict(X_test[i])
    rmse_cv.append(np.sqrt(mean_squared_error(y_test[i],y_pred)))
print("Average RMSE across Folds:",np.mean(rmse_cv))
print("Average Scaled RMSE across Folds:",np.mean(rmse_cv)/(max(y)-min(y)))
```

```
Average RMSE across Folds: 1511.7573978732548
Average Scaled RMSE across Folds: 0.11394732541653638
```

In [33]:
```
# Check performance on test set
model = linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=None).fit(X,y)
y_pred = model.predict(X_ho)
rmse = np.sqrt(mean_squared_error(y_ho,y_pred))
r2 = r2_score(y_ho,y_pred)
print("RMSE: ",rmse)
print("R2:",r2)

scaled_RMSE = rmse/(max(y)-min(y))
print(scaled_RMSE)
```

```
RMSE:  1544.0262848099287
R2: 0.5359175043695535
0.11637956313257161
```