Codecademy Capstone

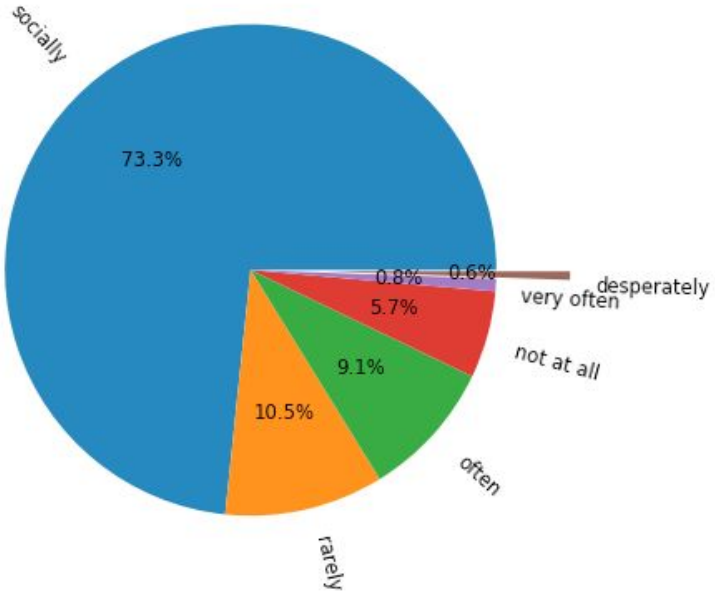
Machine Learning Fundamentals

Joe Bell

Nov 11, 2018

Table of Contents

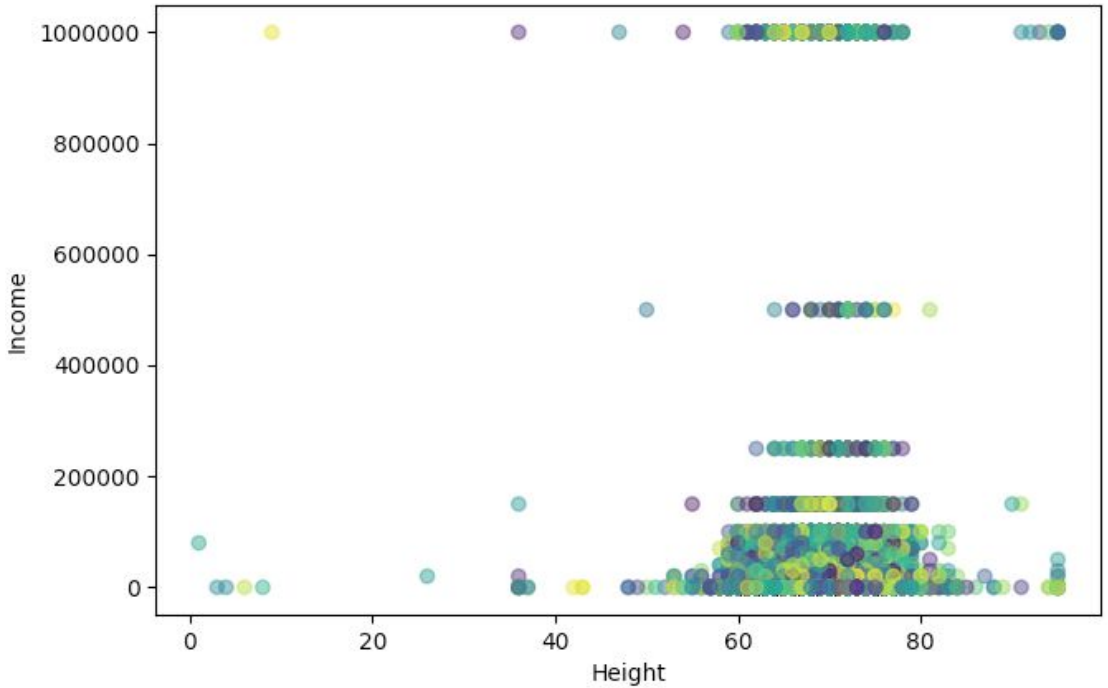
Frequency of alcoholic beverage consumption by OKCupid members



Frequency	Count
socially	41,780
rarely	5,957
often	5,164
not at all	5,164
very often	471
desperately	322

*exploded ‘desperately’ to prevent text overlap and to signal that this group probably requires help

Income based on height of OKCupid users



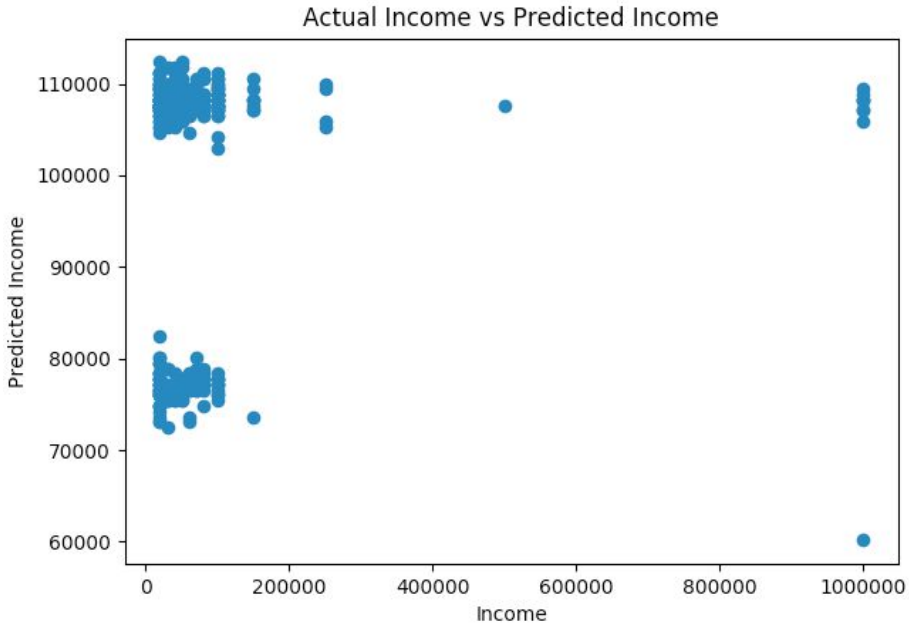
Median Income* = 50000
Median Height = 68

*assumes response of -1 in data is = “no response”, removed this from median calculation

Questions

- Predict income based on height and sex(regression)
- Can we predict the sex of someone based on religion and alcohol consumption? (classification)

Predicted income based on sex and height
Linear Regression



Train score: 0.005687629411298412
Test score: 0.006732083972623482

Outcome: Cannot reliably predict using MLR wit sex and height

Dropped rows where income was not provided

```
df = df.drop(df[df.income == -1].index)
```

Mapped sex to scalar

```
sex_map ={'m':0, 'f':1}
df['sex_code'] = df['sex'].map(sex_map)
```

Multiple Linear Regression performed

```
x = df[['height', 'sex_code']]
y = df[['income']]

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=6

mlr = LinearRegression()
model = mlr.fit(x_train, y_train)
y_predict = mlr.predict(x_test)

print("Train score:")
print(mlr.score(x_train, y_train))

print("Test score:")
print(mlr.score(x_test, y_test))

plt.xlabel("Income")
plt.ylabel("Predicted Income")
plt.title("Actual Income vs Predicted Income")

plt.scatter(y_test, y_predict)
plt.show()
```

Predicted income based on sex and height
K Nearest Neighbor Regression

```
#normalize
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
feature_data = pd.DataFrame(x_scaled, columns=feature_data.columns)

x = feature_data[['height', 'sex_code']]
y = feature_data[['income']]

#create the training and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=6)

#encode float so it can be used by classifier
lab_enc = preprocessing.LabelEncoder()
encoded_y_train = lab_enc.fit_transform(y_train.values)
encoded_y_test = lab_enc.fit_transform(y_test.values)


regressor = KNeighborsRegressor(n_neighbors = 5, weights='distance')
regressor.fit(x_train, encoded_y_train)

print(regressor.predict(x_test))

print("Train score:")
print(regressor.score(x_train, encoded_y_train))

print("Test score:")
print(regressor.score(x_test, encoded_y_test))
```

New array created for training values converting floating point to label encoded using sklearn label encoder

Train score: -0.09144135571729772
Test score: -0.08685180847967966

Regression Conclusions

Accuracy is low on both approaches. Accuracy was calculated using the .score method which returns the mean accuracy of the given test data and labels.

I am not sure I did the k nearest neighbors regression properly; I could not produce a meaningful plot.

I think each approach takes about the same about of time to implement and run. K nearest neighbors should produce a more accurate outcome because the results are normalized, and this is evidenced by the higher score.

We can conclude that sex and height are not good predictors of income.

Predicted sex based on religion and alcohol consumption
K-Nearest Neighbor

1) Create new mapped columns:

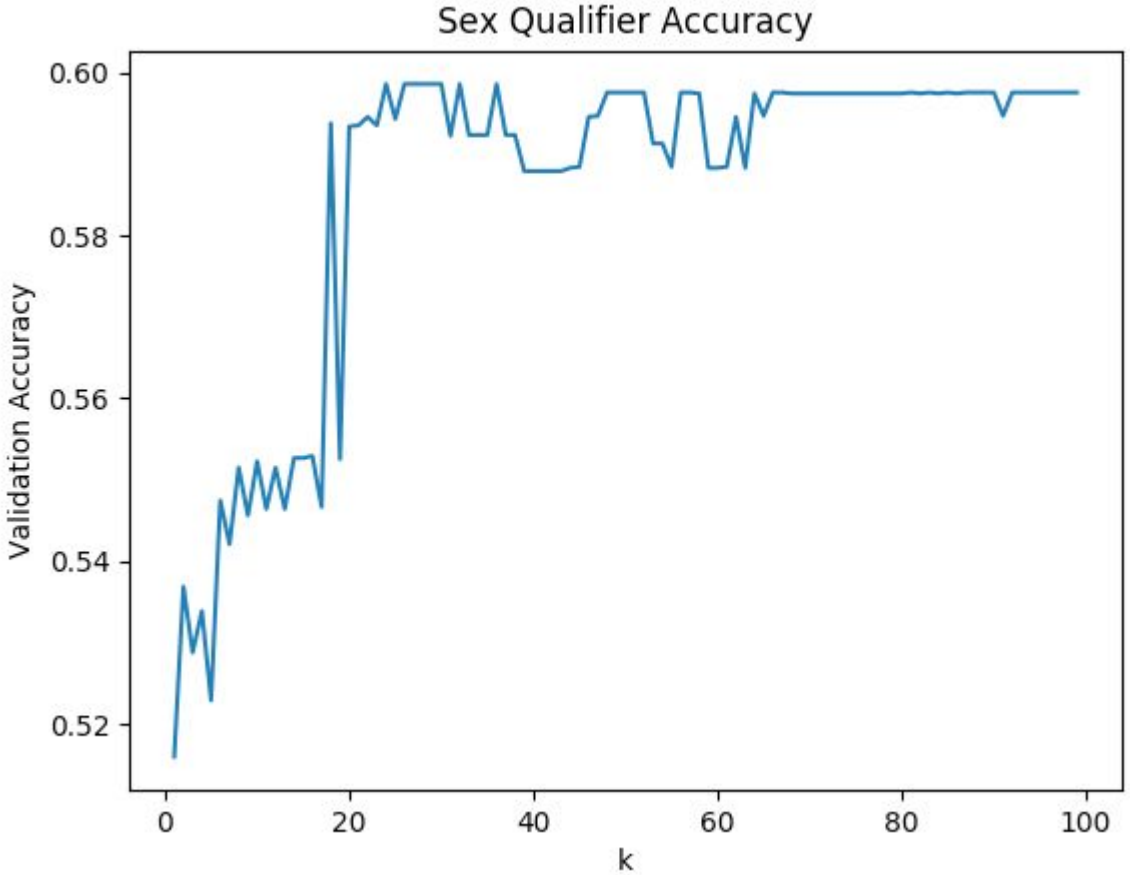
```
religion_map = {'agnosticism':0, 'other':1, 'atheism':2, 'christianity':3, 'judaism':4, 'catholicism':5, 'islam':6, 'buddhism':7}
df['religion_code'] = df['religion'].map(religion_map)

drink_map = {"not at all": 0, "rarely": 1, "socially": 2, "often": 3, "very often": 4, "desperately": 5}
df['drink_code'] = df['drinks'].map(drink_map)

sex_map = {'m': 0, 'f': 1}
df['sex_code'] = df['sex'].map(sex_map)
```

2) Normalize:

```
# create df from all data with just the data i need
feature_data = df[['religion_code', 'drink_code', 'sex_code']]
feature_data.dropna(inplace=True)
#convert to np array
x = feature_data.values
#normalize
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
feature_data = pd.DataFrame(x_scaled, columns=feature_data.columns)

x = feature_data[['religion_code', 'drink_code']]
y = feature_data[['sex_code']]
```

3) Plot:

```
N = 1
x_list = []
y_list = []
while N < 100:

    classifier = KNeighborsClassifier(n_neighbors = N, weights='distance')
    classifier.fit(x_train, encoded_y_train)
    valid_acc_y = classifier.score(x_test, encoded_y_test)
    x_list.append(N)
    y_list.append(valid_acc_y)
    N += 1

plt.plot(x_list, y_list)
plt.xlabel("k")
plt.ylabel("Validation Accuracy")
plt.title('Sex Qualifier Accuracy')
plt.show()
```

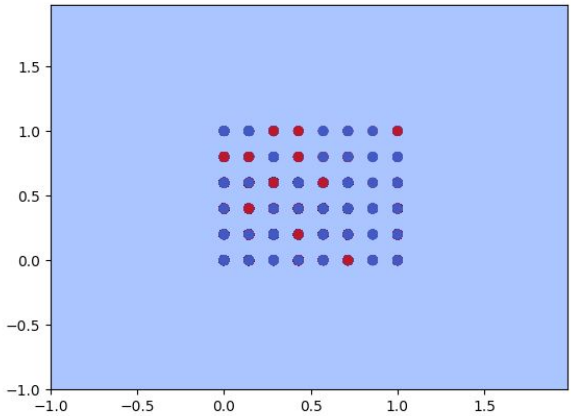


Accuracy levels off at N = 30

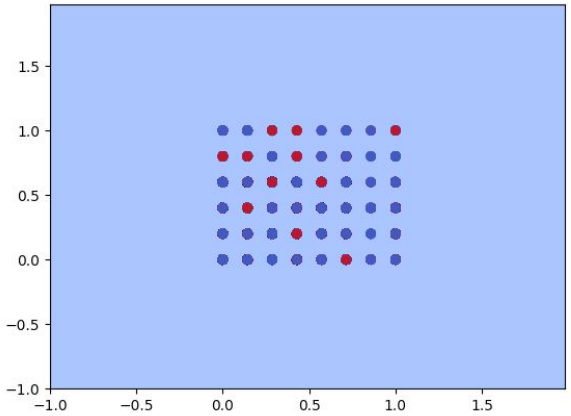
Predicted sex based on religion and alcohol consumption SVM

Plot of data using Linear and RBF SVC. This data is non linearly separable. Code on following slide; not sure how to proceed.

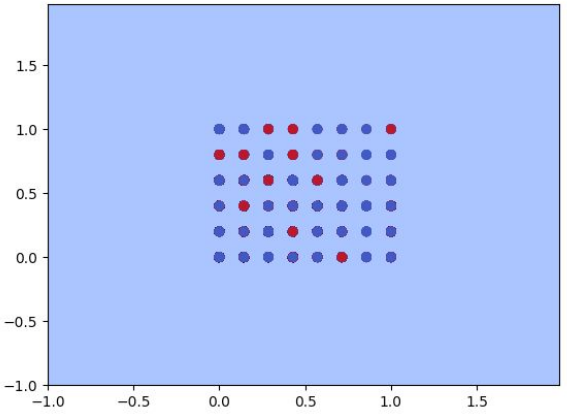
SVC using Linear Kernel



SV using RBF Kernel
Gamma = 0.3



SV using RBF Kernel
Gamma = 5.0



```
religion_map = {'agnosticism':0, 'other':1, 'atheism':2, 'christianity':3, 'judaism':4, 'catholicism':5, 'islam':6, 'buddhism':7}
df['religion_code'] = df['religion'].map(religion_map)

drink_map = {"not at all": 0, "rarely": 1, "socially": 2, "often": 3, "very often": 4, "desperately": 5}
df['drink_code'] = df['drinks'].map(drink_map)

sex_map = {'m': 0, 'f': 1}
df['sex_code'] = df['sex'].map(sex_map)

# create df from all data with just the data i need
feature_data = df[['religion_code', 'drink_code', 'sex_code']]
feature_data.dropna(inplace=True)
#convert to np array
x = feature_data.values
#normalize
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
feature_data = pd.DataFrame(x_scaled, columns=feature_data.columns)

x = feature_data[['religion_code', 'drink_code']]
y = feature_data['sex_code']

h=.02 # step size in the mesh
#create the training and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=6)
```

```
classifier = SVC(kernel='linear')
classifier.fit(x_train.values, y_train.values)
# create a mesh to plot in
x_min, x_max = x_train.values[:,0].min()-1, x_train.values[:,0].max()+1
y_min, y_max = x_train.values[:,1].min()-1, x_train.values[:,1].max()+1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))


Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.axis('tight')
plt.scatter(x_train.values[:,0], x_train.values[:,1], c=y_train.values, cmap=plt.cm.coolwarm)
plt.show()
```