What Makes a Good Walk?

Peter Mills

As a keen hillwalker and mountain lover, living in Scotland, I often find myself using the brilliant website: https://www.walkhighlands.co.uk/. It is an outstanding resource, with detailed walk reports, a helpful forum and, personalise-able maps for Munro bagging (the practise of attempting to climb every Scottish mountain over 3,000 feet). One particular feature caught my attention, each walk on the site has an average user rating, this got me wondering, what makes a highly rated walk, could I create a model to predict this? The goal of this project will be to explore this question, primarily as an enjoyable learning opportunity but possibly giving insights into how popular routes might be created.

This lead me to these project sections:

1. Project Ethics
2. Data Collection and Processing
3. Data Visualisation
4. Modelling

1 Project Ethics

"Climbing and mountaineering are activities with a danger of personal injury or death. Participants in these activities should be aware of and accept these risks and be responsible for their own actions and involvement."
-Walkhighlands terms of use

Mountain Safety: The walkhighlands terms of use are a clear example that anyone promoting mountaineering activities needs to address safety. To be clear, this project is aimed at understanding what makes a route popular and not what an individual should attempt. Before attempting any trip in the mountains, it is essential that one has the necessary skills and experience particularly as I write this in the winter where conditions require even more care.

Recommendations: The subject of ethics surrounding algorithms used to create recommendations is far too complex to completely address here, in fact it could easily be a subject for a whole career let alone a small project. Nevertheless, we need to consider if the models we train are going to potentially cause harm, rather than benefit. One the positive side, predicting what may be popular could potentially be useful for the site's owners when selecting what to add in the future and the educational value to me as an aspiring Data Scientist is also significant. Possible concerns arise if the models are used in isolation to select walks. What if a model highly values walks in certain areas at the expense of less popular areas, depriving already over looked communities of much needed tourism? Ultimately, if these models were to be used with no (or limited) human oversight, this concern would need to be addressed, perhaps even by removing location from the features used by the model. As this is an educational project, this is not necessary but is an interesting point to consider and could form the basis for future research.

Webscraping The last thing I want to achieve is an inadvertent denial of service (DoS) attack or a copyright infringement, both need to be considered before ethical webscraping. The key step to avoid accidentally creating a DoS attack, is to check the robots.txt file for the page, to confirm that there are no restrictions on the pages I planned to scrape. Conveniently, none of the pages we are interested in were disallowed. If this had not been the case, I would need to seek permission from the site, or reevaluate my plan.

Copyright: *"You are allowed to copy limited extracts of works when the use is non-commercial research or private study"*-Gov.uk website. My project certainly constitutes non-commercial research and private study, additionally I chose not to scrape the route descriptions. By not including this information in my data, I'm avoiding my data set being used for the same purpose as the site, reducing any impact on it.

2 Data Collection

My first goal is to create a pandas DataFrame containing all the day walks for Highland and Island areas, the majority walks on the site. I will not include Lowland and long distance walks as these are not completely comparable. This might make for some future research expanding this project to all of the site's content.

Information about each walk is given on its own page, which I need to scrape individually. Here is an example page: https://www.walkhighlands.co.uk/cairngorms/cairn-toul.shtml. Unfortunately, the site does not have a single list of these pages so generating a list of the urls needed was a more involved process. To add to this the site uses relative urls, which required scraping multiply pages to get the full url for each page.

Here is one of the functions I wrote which used regular expression (regex) to locate the walk page links, notably the try statements where needed to deal with the varying formats taken by links on the page.

[5]:
```
def find_walks(areas, href):
        walk_links=[]
        for area in areas:
        url=re.findall('https://www.walkhighlands.co.uk/[a-z-]*', area)[0]+'/'
        soup=bs(requests.get(area).content,'html.parser')
        text=soup.find('tbody')
        try:
                walk_links+=[url+a.get('href') for a in text.find_all('a') if a.
                get('href') not in href and re.match('[a-zA-Z-]*.shtml', a.
                get('href'))]
        except:
                #print(area) #find any problematic walk links
                pass


        try:
                walk_links+=[base+a.get('href') for a in text.find_all('a') if a.
                get('href') not in href
                and re.match('/[a-zA-Z-]*/[a-zA-Z-]*.shtml', a.get('href'))]

        except:
                #print(area) #find any problematic walk links
                pass


        return walk_links
```

Once I have a function for each step written and tested, I can simply chain them together, to give me a complete list of pages to scrape.

[8]:
```
walk_pages=find_walks(find_areas(region_links), excluded_href)
```

Now the most time consuming step in the whole process, to actually scrape all 1570 pages, using my get_walk_featues function.

[14]:
```
final_array=[]
for page in walk_pages:
        final_array+=[get_walk_features(page)]
```

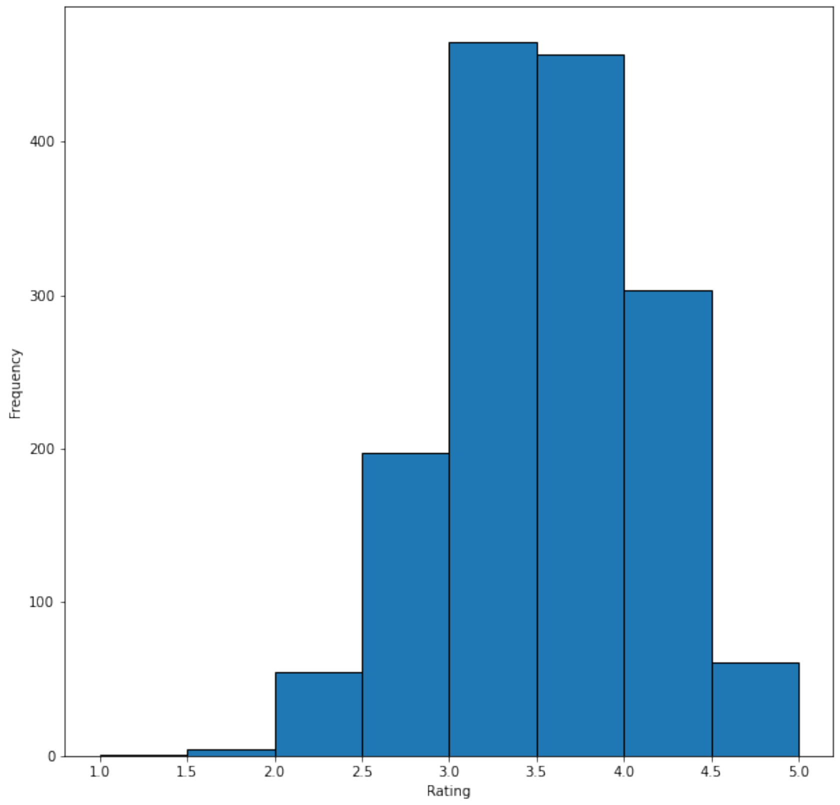
I would be interested to know if there is a more efficient way to do this. Once this step is completed, it is simple to create a DataFrame.

[16]:
```
df=pd.DataFrame(final_array,columns=['name','info','region',
'dist','ascent','corbett','munro','grade','bog','rating'])
```

I then processed this DataFrame in a separate notebook. This involved: cleaning null values, one-hot encoding the region feature, creating keyword features, and performing a train-test split in preparation for model testing.

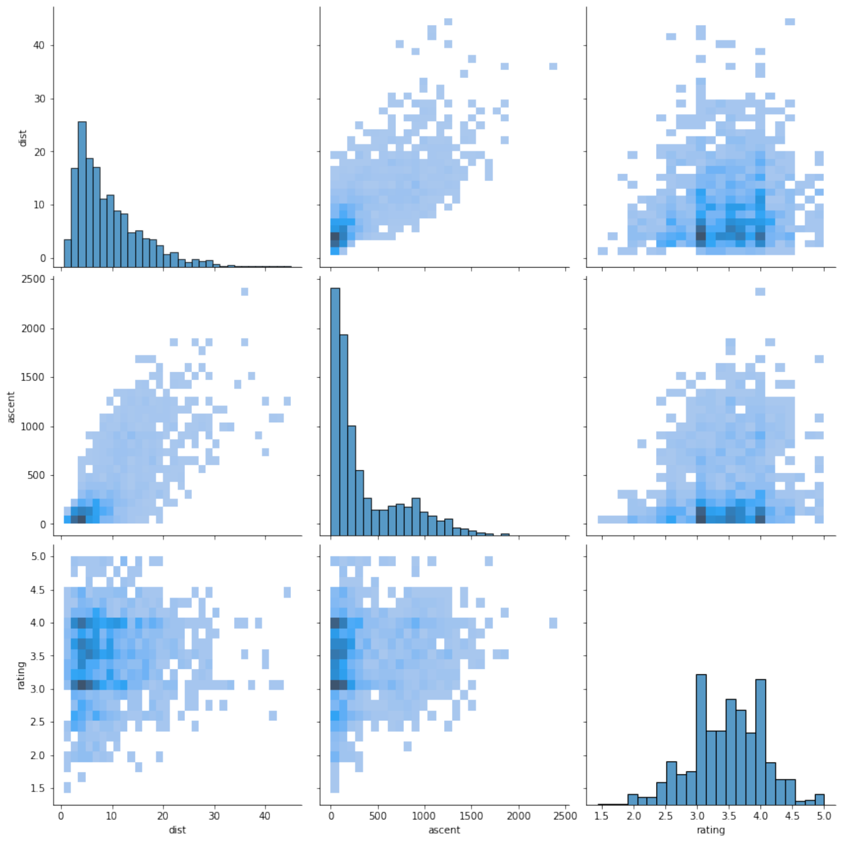
3 Data Visualisation

Let us start by looking at our target, how are ratings distributed? A bar char is a simple way to check we have a good range of values and whether there is skew.

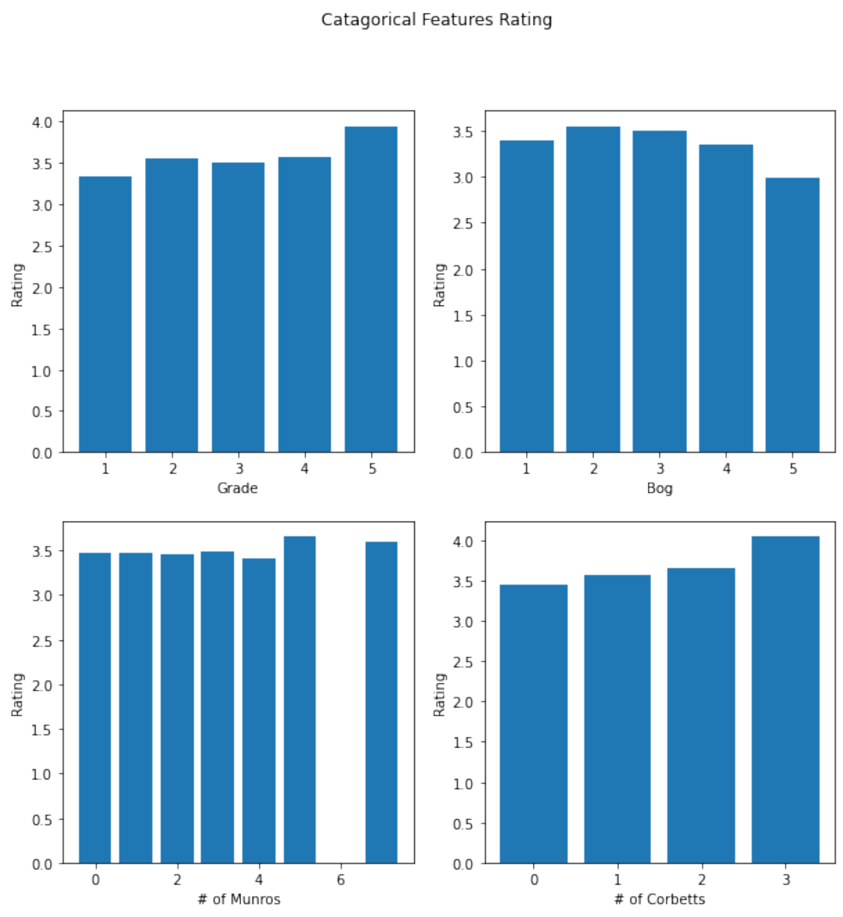


We can see that ratings are centred around 3.5 and are somewhat negatively skewed, ratings on the higher end being a little more common than lower ones. If we had found the majority of walks where rated a single value, this would have raised questions about the viability of the project but this distribution we see gives us options.

Now we can get an idea how features in the data are related using a seaborn pair plot for the continuous features. A histogram plot is preferable here as this allows us to clearly visualise the data when there are many points in a small area of the chart.

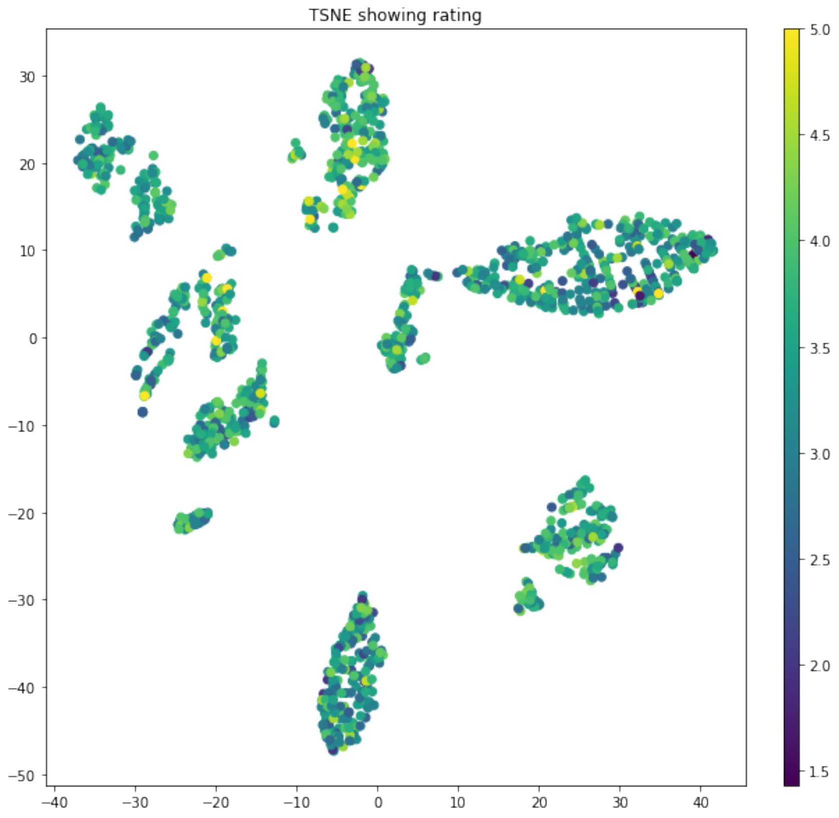


Immediately we can see a positive correlation between distance of the walk and ascent, which is no surprise. We can't see a strong correlation between either of these features and rating, although it looks like lower rated walks do tend to be shorter and flatter. Now we can look at the discrete features in our data.

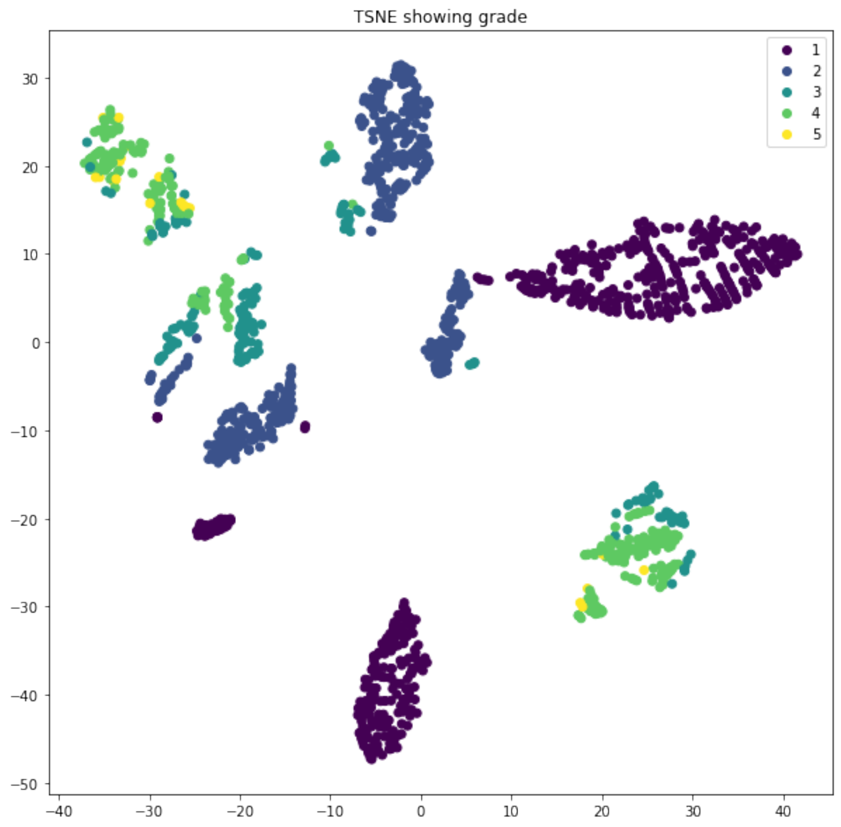


We can see walks with higher grades and less bog tend to be more highly rated. Though it looks like the number of Corbetts on a route might also be a useful feature, we should note that it is highly skewed with the majority of walks containing none (see notebook).

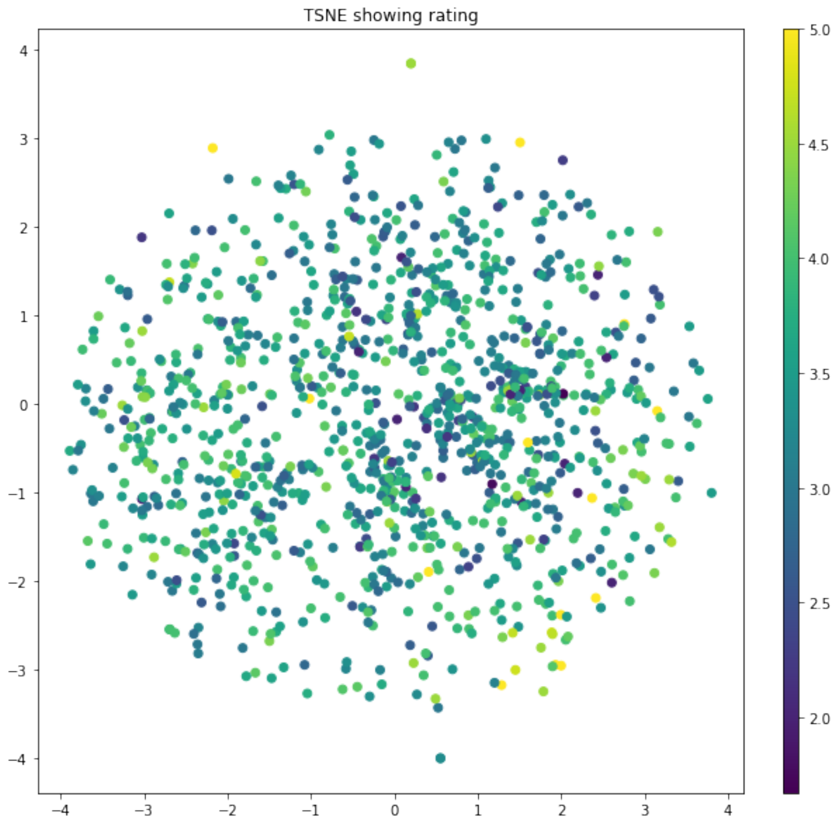
Handling a data set with a high number of features does beg the question, can we use dimensionality reduction to help visualise this? This project is my first experience using t-distributed stochastic neighbor embedding (t-SNE) so taking time to tweak the parameters and consider clusters in the data is essential. It can be easy to misinterpret t-SNE plots, particularly as some one who is new to them. I do not have space here to discuss the full process behind this it is in the accompanying notebook for visualisation. Now some t-SNE plots, the first we will look at is the data without region or keyword features.



When varying the perplexity, the clusters we see remained, likely indicating some structure here. Using the colouring, it is apparent that ratings vary widely in each cluster but perhaps some clusters contain more high ratings than others. This may be useful later when modelling, if there are clusters in the data we can separate. If we use the colouring to explore which features may be causing the clustering we find:



The grade the walk has been given is really the key feature driving the clustering. This highlights an issue with using t-SNE for data like this, where we have continuous and discrete features. Discrete features create this problem for a distance based algorithm like t-SNE, points sharing a value are likely to appear close in the distance metric even if otherwise different. With the inclusion of more discrete features, namely the one-hot encoded regions and keywords, we end up with no visible structure.



Though this plot looks quite random, I would be surprised if there are no correlations between certain key words. For example I would expect routes with a large amount of ascent to include words like climb, steep, descend, and so on. Clearly a simple dimensionality reduction can't highlight any patterns like this for us, perhaps they are not there or different technique could be used. Understanding how to use keyword features in greater depth is certainly a topic for my future research.

4 Modelling

As the target is continuous, my first instinct is to try regression. By using regression, we are effectively aiming to predict the exact rating for each walk, which is an ambitious approach. Unfortunately, the data does not contain strong enough signals for my models to be effective in this. We can see this by comparing a dummy regressor to my trained models

[3]:
```
dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(X_train, y_train)
y_predict_train=dummy_regr.predict(X_train)
y_predict=dummy_regr.predict(X_test)

print('train_error:',mean_squared_error(y_predict_train,y_train,squared=False))
print('test_error:',mean_squared_error(y_predict,y_test,squared=False))
```

```
train_error: 0.5775009941808824
test_error: 0.5863082156825604
```

[5]:
```
for feats, train, test in [['Key Words used:',X_train,X_test],
['Key Words not used:',X_train_reduced,X_test_reduced]]:
        print(feats)
        for name, estimator in pipelines.items():
        estimator.fit(train,y_train)
        y_predict_train=estimator.predict(train)
        y_predict=estimator.predict(test)

        train_error=mean_squared_error(y_predict_train,y_train,squared=False)
        test_error=mean_squared_error(y_predict,y_test,squared=False)
        print(name, train_error,test_error)
```

```
Key Words used:
rf 0.20206088753917748 0.5335716034762994
xgb 0.08241249107392869 0.5624033976531516
Key Words not used:
rf 0.2196130865569877 0.595686035648377
xgb 0.16981489163998792 0.6652050347082112
```

We can see that the mean squared error (on the test data) for Random Forest and XGBoost is hardly an improvement over the dummy, in fact when keyword features are not used the models actually perform worse than the dummy. Using GridSearchCV to tune hyper-parameters only offers a marginal improvement and neither model can be considered useful.

Does this mean we have failed to answer our question? Not necessarily, when approaching ratings prediction, we can treat it as a regression task or as a classification. All we have to do is split the continuous ratings into categories by looking at intervals. In the modelling notebook, I look at two ways to do this, either splitting the ratings into 2 or 3 intervals. Let's look at the former here as this is most successful. The question we are now answering is, can we predict when a walk will receive a rating of 4 or higher? This is obviously much less ambitious that our initial question when looking at regression but is still of interest to someone adding new walks to the site. No need to use a dummy here as AUC score gives us an idea of how a no skill model would perform so let's look strainght at the models.

```
[18]: pipelines_binary={
              'rf_class': make_pipeline(RandomForestClassifier()),
              'xgb_class': make_pipeline(XGBClassifier())}
```

```
[26]: for feats, train, test in [['Key Words used:',X_train,X_test],
              ['Key Words not used:',X_train_reduced,X_test_reduced]]:
              print(feats)

              for name, classifier in pipelines_binary.items():
                      classifier.fit(train,y_train_binary)
                      y_predict_train=classifier.predict_proba(train)[:, 1]
                      y_predict=classifier.predict_proba(test)[:, 1]

                      print(name, roc_auc_score(y_train_binary
                      , y_predict_train),roc_auc_score(y_test_binary, y_predict))
```

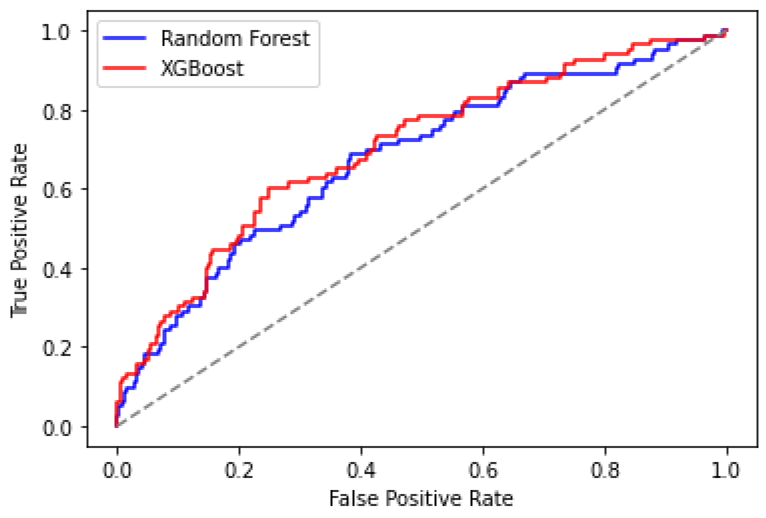
```
Key Words used:
rf_class 1.0 0.6714183466539424
xgb_class 0.9999918750710931 0.6915185494453059
Key Words not used:
rf_class 1.0 0.6498866754145294
xgb_class 0.9938880222298054 0.6608016223309078
```

Here we can already see a significant improvement compared to regression. Though the models are overfitting, they show some ability to separate the classes, with keywords improving both models. Now after carrying a grid search again to help control the overfitting we get the following:

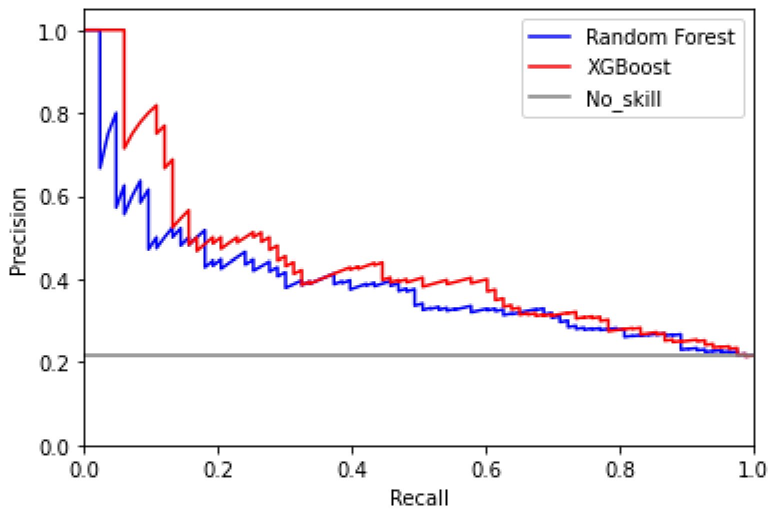
```
[24]: for name, classifier in tuned_classifiers.items():
              y_predict_train=classifier.predict_proba(X_train)[:, 1]
              y_predict=classifier.predict_proba(X_test)[:, 1]

              print(name, roc_auc_score(y_train_binary, y_predict_train),
              roc_auc_score(y_test_binary, y_predict))
```

```
rf_class 0.8147475584588635 0.676010974591435
xgb_class 0.8558596987276362 0.7023738518430156
```

We can see that for the XGBoost model, we have got a reasonable AUC score, showing our model has skill separating the classes. Whether this level of separability is adequate depends largely on how someone plans to use the model. We can plot the full ROC curves to get a more in-depth picture.



Because our data is slightly skewed, with around 24% in the positive class, it is also worth looking at precision-recall. If this skew had been more significant, we would not have used roc_auc scoring when training our models, instead using another scoring such as f1. Here is the precision-recall curves for each model.



This confirms what we saw when looking at the ruc curves, both models demonstrate skill in their predictions although neither are exceptional. Whether the trade off between precision and recall that we can see here could be considered useful depends on the intended use for the model.

5 Conclusion

Looking back at our question, can we predict when a walk will be popular? Well yes and no, we have had some success treating the problem as a classification, where we have trained a somewhat capable model but from the perspective of regression, we were unsuccessful. Far more importantly, was this a useful educational experience? The answer here is a confident yes, applying many of these techniques has given me a new appreciation of real world issues that a data scientist may face. Learning how treating the problem as a regression or classification impacted how effective my models were, was a particular highlight and I really enjoyed the challenge of webscraping with a difficult goal. Thank you for reading this write up, I hope you found the topic as interesting as I did.