Project1_edgarsp2

December 3, 2019

0.1 [10 Points, half a page] Project description and summary.

In this project we are using the Wine Review dataset. We start by exploring the data and find interesting insights. We then fit 2 separate regression models to predict wine points. The first model I used is a linear regression with price and province as features. The results were okay, a 7.20 mean squared error and 0.23 Rˆ2 score. The second model is a gradient boosted model with TF-IDF features to process text data. This model performed better achieving a 3.76 mean squared error and a 0.59 Rˆ2 score. On the wine recommendations, a similar model is used for text searching. We search for nearest neighbors giving a keyword and then filter the results by the price. This shows good results overall.

0.2 [10 Points, within 1 page] Data processing.

Data processing for the linear regression model is simple. I first fill any missing values with the price mean. This was done to avoid any invalid values. I then had to one hot encode the province. This means splitting the province column data to many columns depending on the number of categories present in that column. Each column contains 0 or 1 corresponding to which column it has been placed. For the gradient boosted recommendations model, the data processing was a little more involved. I first remove any popular words in the english dictionary like "the" and "a". This reduces the amount of noise in the wine descriptions. I also converted word stems to the main word. Once I had a cleaned list of wine description, I then converted them to a matrix of term frequency-inverse document frequency(TF-IDF) features. This is a statistical measure used to evaluate how important a word is to a document in a collection

0.3 [10 Points, within 1 page] Descriptive statistics with tables/figures.

0.3.1 Get overall description and info of dataset

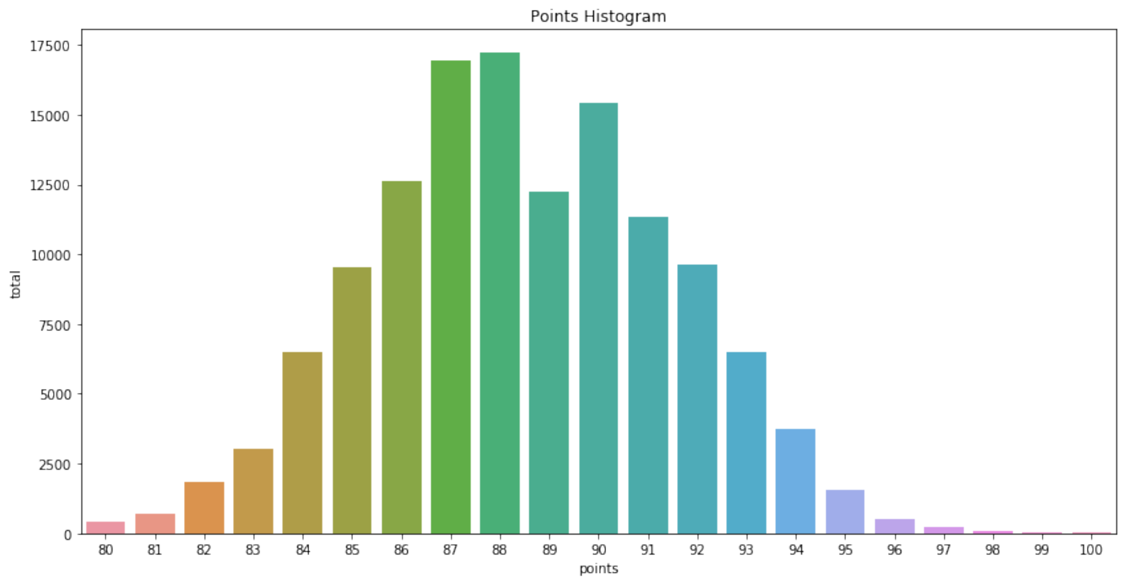
[854]:
```
wine_df.describe()
```

[854]:
```
              points          price
count  129971.000000  120975.000000
mean       88.447138      35.363389
std         3.039730      41.022218
min        80.000000       4.000000
25%        86.000000      17.000000
50%        88.000000      25.000000
75%        91.000000      42.000000
```

```
max        100.000000    3300.000000
```

Wine point distribution

[18]:
```
points = wine_df["points"].value_counts()
points = pd.DataFrame({'points': points.index,'total': points.values})
plt.figure(figsize = figure_size)
plt.title('Points Histogram')
sns.set_color_codes("pastel")
sns.barplot(x ='points', y='total', data=points)
locs, labels = plt.xticks()
plt.show()
```



0.4 [35 Points, within 3 pages] Regression model analysis.

0.4.1 Linear Regression Model

Prepare data Fill NA values with average. Choose price and country as input features. One hot encode country. Split train/test data

[20]:
```
cleaned_df = wine_df.copy()
cleaned_df.price = cleaned_df.price.fillna(cleaned_df.price.mean())
selected_features = cleaned_df[['price','province']]
x = pd.get_dummies(selected_features,prefix=['province'])
y = cleaned_df.points.to_numpy()
```

[31]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25,␣
↪random_state=42)
```

Fit linear regression model

[32]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

[34]:
```
y_pred = model.predict(X_test)
```

[35]:
```
f"MSE {mean_squared_error(y_test, y_pred)}"
```

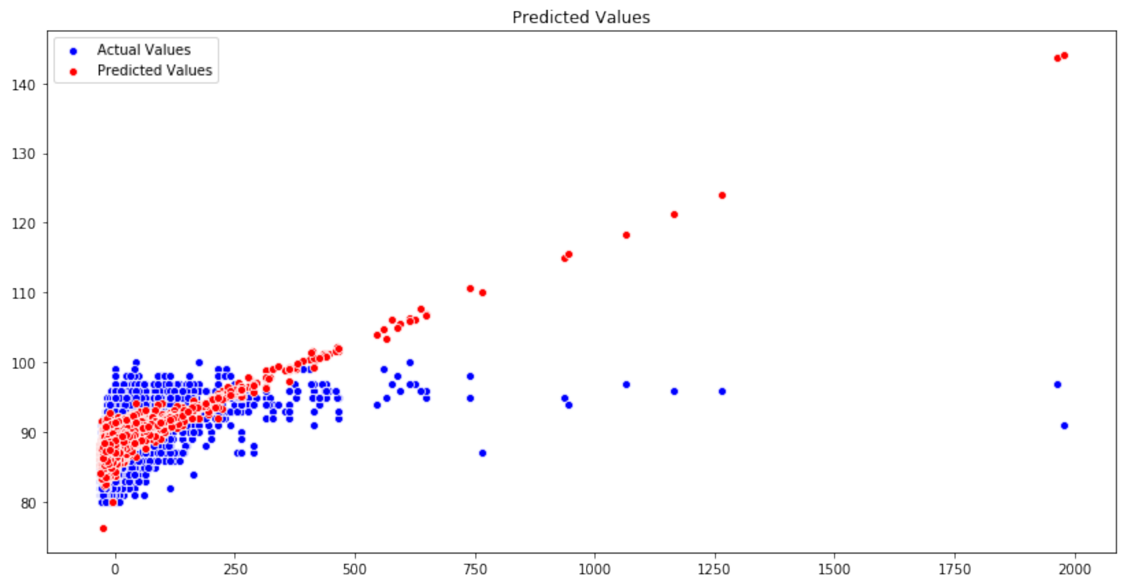
[35]:
```
'MSE 7.204910408282384'
```

[37]:
```
F"R^2 score {model.score(X_test, y_test)}"
```

[37]:
```
'R^2 score 0.230687632251764445'
```

Plot predicted values by doing PCA

[40]:
```
pca = PCA(n_components=1)
X_test_pca = pca.fit_transform(X_test).reshape(-1)
plt.figure(figsize = figure_size)
plt.title('Predicted Values')
sns.scatterplot(X_test_pca, y_test,color='blue', label="Actual Values")
sns.scatterplot(X_test_pca, y_pred,color='red', label="Predicted Values")
plt.show()
```



We can see that the red line is not as accurate. Some of the points are being estimated to be higher than 100, which is not what we want.

0.4.2 Gradient Boosted Model

Prepare data. We will convert the descriptions to lowercase, remove and stop words in the english language, and convert stems to one word.

[858]:
```
cleaned_df = wine_df.copy()
x = cleaned_df['description']
y = cleaned_df['points']
x = x.str.lower()
x = x.apply(lambda elem: re.sub('[^a-zA-Z]',' ', elem))
tokenizer = RegexpTokenizer(r'\w+')
words_descriptions = x.apply(tokenizer.tokenize)
stopword_list = stopwords.words('english')
ps = PorterStemmer()
words_descriptions = words_descriptions.apply(lambda elem: [word for word in␣
 ↪elem if not word in stopword_list])
words_descriptions = words_descriptions.apply(lambda elem: [ps.stem(word) for␣
 ↪word in elem])
x_cleaned = words_descriptions.apply(lambda elem: ' '.join(elem))
```

Let's convert the collection of reviews to a matrix of TF-IDF features.

[100]:
```
vectorizer = TfidfVectorizer(analyzer='word',␣
 ↪token_pattern=r'\w+',max_features=500)
```

[101]:
```
x_vectorized = vectorizer.fit_transform(x_cleaned)
```

Fit the Adaboost regressor model

Split train/test data

[122]:
```
X_train, X_test, y_train, y_test = train_test_split(x_vectorized, y,␣
 ↪test_size=0.25, random_state=42)
regr = GradientBoostingRegressor()
regr.fit(X_train, y_train)
```

[140]:
```
regr.score(X_train, y_train)
```

[140]:
```
0.45377043566858666
```

[141]:
```
regr.score(X_test, y_test)
```

[141]:
```
0.44992286051117486
```

0.4.3 We can do cross validation search to find the best set of params.

[160]:
```
parameters = {'n_estimators':[200, 350], 'learning_rate':[.1,.25], 'max_depth':␣
 ↪[3,4]}
```

[161]:
```
regr = GradientBoostingRegressor()
```

[164]:
```
gsmodel = GridSearchCV(regr, parameters, cv=2, n_jobs=-1, verbose=3)
```

[ ]:
```
gsmodel.fit(X_train, y_train)
```

[173]:
```
f"The best score is {gsmodel.best_score_} with the following params {gsmodel.
↪best_params_}."
```

[173]:
```
"The best score is 0.5754155537124865 with the following params
{'learning_rate': 0.25, 'max_depth': 4, 'n_estimators': 350}"
```

[181]:
```
best_model = gsmodel.best_estimator_
```

[183]:
```
f"Score on tersting data is {best_model.score(X_test, y_test)}"
```

[183]:
```
'Score on tersting data is 0.5979222662552914'
```

[187]:
```
y_pred = best_model.predict(X_test)
```

[189]:
```
f"MSE for best model {mean_squared_error(y_test, y_pred)}"
```

[189]:
```
'MSE for best model 3.765614814272545'
```

0.5 [20 Points, within 1 page] Recommend five different wineries.

[560]:
```
wines = wine_df.copy()
```

0.5.1 Data cleanup

Let's convert the variety and description to lowercase. We create a new `variety_description` column which we'll use later for our model. We also drop and null/na values.

[534]:
```
wines['winery'].dropna(inplace=True)
wines['variety'] = wines['variety'].str.lower()
wines['price'].dropna(inplace=True)
wines['variety_description'] = wines['description'] + ' ' +  wines['variety']
wines['variety_description'].dropna(inplace=True)
```

0.5.2 Recommendations model

Clean text data Remove any stop words and get word stems.

[693]:
```
description = wines['variety_description']
description = description.apply(lambda elem: re.sub('[^a-zA-Z]',' ', elem))
tokenizer = RegexpTokenizer(r'\w+')
words_description = description.apply(tokenizer.tokenize)
stopword_list = stopwords.words('english')
ps = PorterStemmer()
words_description = words_description.apply(lambda elem: [word for word in elem␣
↪if not word in stopword_list])
words_description = words_description.apply(lambda elem: [ps.stem(word) for␣
↪word in elem])
description_cleaned = words_description.apply(lambda elem: ' '.join(elem))
wines['variety_description_cleaned'] = description_cleaned
```

[774]:
```
wines_by_winery = []
```

[775]:
```
for winery in wines.winery.unique():
    winery_rows = wines.loc[wines['winery'] == winery]
    price = winery_rows.price.mean()
    points = winery_rows.points.mean()
    count = len(winery_rows)
    descriptions = winery_rows.variety_description_cleaned.str.cat(sep=' ')
    descriptions = winery_rows.variety_description_cleaned.str.cat(sep=' ')
    wines_by_winery.append({'winery': winery.lower().strip(), 'price': price,␣
 ↪'descriptions': descriptions, 'points': points, 'count': count})
```

[777]:
```
wines_by_winery_df = pd.DataFrame(wines_by_winery)
vectorizer = TfidfVectorizer(analyzer='word',␣
 ↪token_pattern=r'\w+',max_features=500)
rec_vectorized = vectorizer.fit_transform(wines_by_winery_df['descriptions'])
model = NearestNeighbors(metric='cosine').fit(rec_vectorized)
```

0.5.3 Let's query the model and see the results

[846]:
```
def get_recommended_wines(search_term, variety,  price_max, model, wine_data,␣
 ↪vectorizer):
    variety = variety.lower().strip()
    search_term += ' ' + variety
    search_term = tokenizer.tokenize(search_term)
    search_term = [ps.stem(word) for word in search_term]
    search_term = ' '.join(search_term)
    distances, indices = model.kneighbors(vectorizer.transform([search_term]),␣
 ↪1000)
    distances = distances.flatten()
    indices = indices.flatten()
    recommended_wines = wine_data.iloc[indices]
    recommended_wines = recommended_wines.sort_values(['points', 'count',␣
 ↪'price'],ascending=False)
    return recommended_wines.loc[recommended_wines['price'] <= price_max][:5]
```

[847]:
```
price_max=20
variety='Pinot Noir'
recommended = get_recommended_wines('fruity', variety, price_max, model,␣
 ↪wines_by_winery_df, vectorizer)
```

[861]:
```
list(recommended.winery)
```

[861]:
```
['mcpherson wine company',
 'pike road',
 'stone wolf',
 'point concepción',
 'te kairanga']
```