CSC311 ML Challenge

Antonio Salvatore LaPlaca
Konrad Wozniak
Michael Kwan
Xingjian (James) Zhang

December 5, 2022

Contents

1 Data

We explored this data by first parsing through our features to get a qualitative understand of them and their possible relations to the problem. Our problem being that we had a set of data from fellow classmates that was used to classify three different abstract images quantitatively and qualitatively, and we as a team have to figure out what the best model, implementation, and features were required to give a high accuracy prediction of what the photo being described was, based on the same inputs/features.

We chose to drop the column `user_id` since the ID of whoever gave their input does not have any meaningful impact on figuring out what image that row belonged to. The remaining labels (aside from the label column) became reasonable to use as input features in testing.

We had a belief that there may have been some possible significant trends within the features; `q_scary`, `q_dream`, and `q_desktop`. After performing some basic cleaning and removing the string values, we created some visual heat maps for each of the features, with the X-Axis representing the `label` feature, being compared to the previously mentioned 3 features in the Y-Axis. We found some differences between the features, but not as significant as we may have hoped. Nonetheless, we looked to further create models to test this possible significance and determine if they were significant enough to create predictions with reasonable accuracy. The following heat maps represent the `q_scary`, `q_dream`, and `q_desktop` features in relation to the `label` in that corresponding order;

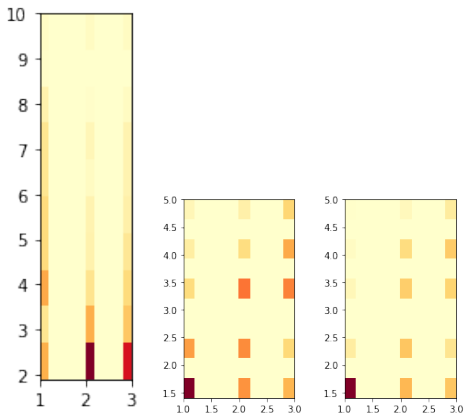


Figure 1: From Left to Right: q_scary, q_dream, and q_desktop

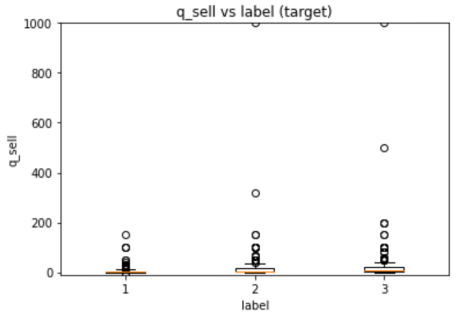
While the quantitative features `q_scary`, `q_dream`, `q_desktop`, `q_sell` `q_temperature` were examined for use in our models. However, in order to prevent large variances in data from negatively impacting a model we use (ex. Making one feature seem more important than others), we need to make sure that the values were normalized. The normalization was done by making each feature conform to an almost normal distribution.

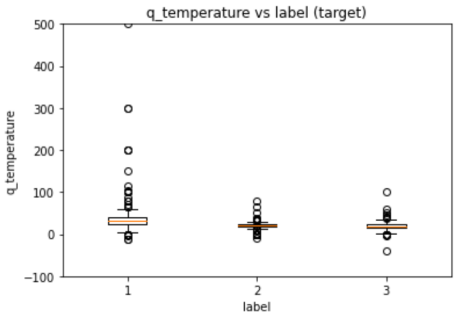
The quantitative features were normalized using the following formula; $\frac{x_i^{(k)} - \mu_{x^{(k)}}}{\sigma_{x^{(k)}}}$

Deciding which data to use was crucial for deciding which model we would end up using. We decided that some of the feature data would not be useful/significant in our prediction model through looking at initial test results and discussion of intuitions. We eventually decided that the stories (consisting of string sentences) may have much better connection and accuracy for predicting the image described. Since the first image looks like a boy playing soccer, words like "ball" would probably show up often. We explored in depth the impact of the temperature and sell value of each data point. After cleaning and filtering the data to closely inspect the sell and temp values, we got the following data descriptions and box-plot:

```
data.describe()
```

	q_sell	q_temperature	label
count	597.000000	5.970000e+02	597.000000
mean	677.755829	1.675042e+12	2.000000
std	15722.635448	4.092728e+13	0.817181
min	-500.000000	-4.100000e+01	1.000000
25%	0.500000	1.700000e+01	1.000000
50%	5.000000	2.200000e+01	2.000000
75%	13.000000	3.000000e+01	3.000000
max	384061.180000	1.000000e+15	3.000000





The data description shows that the means are heavily skewed by outliers, as shown by the maximum values of `q_sell` and `q_temp`. Even after ignoring these outliers, most points lie in a similar range in between 0-100. There definitely is a trend with `q_temp`, as for picture 1 most points lie higher than picture 2. `q_sell` also shows a slight trend, with increased prices occurring at categories three and two, rather than one. The question persists, are these trends worth pursuing? This was further explored when exploring the logistic regression model. In the end, our accuracy working with these features ended up being lower than that of naive Bayesian inference on the story feature, and hence we focused our attention on that model and data.

Yet another filtering process we did on the training data was ignoring data points that have story strings which were left blank. This was a decision that we made based off of the heuristic that if a student did not bother writing a story, it was a strong indicator that the data was likely to be poor quality or meaningless. This is because if a student did not feel like writing a story, they likely did not put any thought or effort into the rest of their answers. Such answers were likely to be skewed and uninformative, and if used in training would add unnecessary variation and outliers we would have to deal with later if not removed from training and testing.

2 Model

2.1 k-nearest neighbours

In k-nearest neighbours, we take into consideration all the numerical features. In the `knn_pred.py` file submitted, it takes as input the `clean_quercus.csv` file, and fits a k-nearest neighbours model using scikit learn. The 'vector points' consist of each row in the cleaned dataset that can be seen in the file `knn_cleaned_data.csv`.

After observing multiple test accuracies when running the script, and observing the model's performance for various values of k, the accuracies were near the 50-60% range. This accuracy was not good enough to be used as our final model, so we moved on to other models.

2.2 Naive Bayes q_story data exploration

By looking over the data set, we've decided to use naive Bayes on `q_story` which is more reasonable since the stories are randomly generated by students.

The way we cleaned the data was by only using the `q_story` column and label column. Firstly we dropped NaN data and created a new list, then we append the useful data into it. Since the first index of the data will be `q_story` and label, so that we only need the sentences and label from index 1 to the end.

Secondly, we created an array of all the unique words.

Thirdly, we produced the bag-of-word representation of the data, along with a vector of labels. By implementing a `make_bow` function. The `make_bow` function should take two parameters (data and vocab, which are what we have modified in the previous step), and returns X: A data matrix of bag-of-word features.t: A numpy array of shape [len(data)], with t[i] == 1 if 'data[i]' is in label 1, and 't[i] == 2' if the data is in label and 't[i] == 3' if 'data[i]' is in label 3 and t[i] == "Error" otherwise. Then we can simply separate data into training/validation, then call `make_bow` function to produce the bag-of-word features. Since the data was originally separated into labels 1, 2 and 3, so we had to randomly shuffle the data. Then we can check the occurrence frequency of each word that appears in the vocab, and then sort them from most to least.

We decide to implement `naive_bayes_map` function to compute the parameters π and θ_{jc} that maximizes the posterior of the provided data (X, t). 'X' - a matrix of bag-of-word features of shape [N, V], where N is the number of data points and V is the vocabulary size. X[i,j] should be either 0 or 1. Produced by the `make_bow()` function. 't' - a vector of class labels of shape [N], with t[i] being either 1, 2 or 3. Produced by the `make_bow()` function. The function `naive_bayes_map` should return 'pi' - a scalar; the MAP estimate of the parameter π and 'theta' - a matrix of shape [V, 3], where 'theta[j, c]' corresponds to the MAP estimate of the parameter $\theta_{jc} = p(x_j|c)$.

In the end, we implemented `make_prediction` function to check the accuracy of prediction. Where the function should take 3 parameters 'X' - a matrix of bag-of-word features of shape [N, V], where N is the number of data points and V is the vocabulary size. X[i,j] should be either 0 or 1. Produced by the `make_bow()` function. 'pi' - a scalar; the MAP estimate of the parameter π and 'theta' - a matrix of shape [V, 3], where 'theta[j, c]' corresponds to the MAP estimate of the parameter θ_{jc}. The `make_prediction` function should return an ndarray that allow us to use in the `accuracy` function we implemented afterward. The accuracy we've got from naive Bayes model was 81 percent.

2.3 Multi-class logistic regression

We considered doing multi-class logistic regression on all of the data but story, as the story feature did not work itself very well to regression. However, all of the other features are possible to be used for regression. All of the other features except for q_sell and q_temp were converted to one-hot vector encodings, as they all were non-ordinal categorical data. The -1 category used to denote empty input made some of the data not-ordinal where otherwise it may have been. All of this data processing was done by data_cleaner.py, which will also be submitted. At first, we just tried using logistic regression on only q_temp and q_sell. This resulted in mediocre, but better than random test accuracy.

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(fit_intercept=False, max_iter=100000000)

# fit the model

lr.fit(X_train_norm, t_train)

print(f"Training accuracy: {lr.score(X_train_norm, t_train)}")
print(f"Validation accuracy: {lr.score(X_valid_norm, t_valid)}")
print(f"Test accuracy: {lr.score(X_test_norm, t_valid)}")
```

```
Training accuracy: 0.6050420168067226
Validation accuracy: 0.525
Test accuracy: 0.4
```

Next, we tried regression on all features excluding story.

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(fit_intercept=False, max_iter=100000000)

# fit the model

lr.fit(X_train_norm, t_train)

print(f"Training accuracy: {lr.score(X_train_norm, t_train)}")
print(f"Validation accuracy: {lr.score(X_valid_norm, t_valid)}")
print(f"Test accuracy: {lr.score(X_test_norm, t_valid)}")
```

```
Training accuracy: 0.803921568627451
Validation accuracy: 0.6916666666666667
Test accuracy: 0.30833333333333335
```

The validation accuracy is roughly 70%. Hence, Majority of these other features were somewhat impact in predicting the category of the image. However, it is important to not, that since we are using Sklearn in this scenario, we are not actually setting any of the hyper-parameters ourselves. This Sklearn package regularizes data on its own, simply based off of the training data given to it. Hence, in this case, the validation set is basically just a test set. The test set was just a vestigial remnant of earlier code in this case. Since we had greater accuracy in using naive Bayesian inference to predict, based on the story, the category of an image, a proper non-Sklearn model of Logistic regression was not developed. To see the entire and thorough investigation of this model, refer to MLChallengelab03.ipynb. Further note, that for this model, we bounded the q_sell and q_temp features to a constant upper bound, to prevent this data from being skewed too much by the sever outliers. These two features were also normalized. Please refer to the attached i-python file for more details.

3 Model Choice and Hyperparameters

3.1 Hyper-Parameters and Validation Accuracy

```
C:\Users\Owner\AppData\Local\Programs\Python\Python3
MAP Train Acc: 0.962882096069869
MAP Valid Acc: a = 1 b = 1 0.8157894736842105
MAP Valid Acc: a = 1 b = 2 0.8157894736842105
MAP Valid Acc: a = 1 b = 3 0.8157894736842105
MAP Valid Acc: a = 1 b = 4 0.8157894736842105
MAP Valid Acc: a = 1 b = 5 0.8157894736842105
MAP Valid Acc: a = 1 b = 6 0.8157894736842105
MAP Valid Acc: a = 1 b = 7 0.8157894736842105
MAP Valid Acc: a = 1 b = 8 0.8157894736842105
MAP Valid Acc: a = 1 b = 9 0.8157894736842105
MAP Valid Acc: a = 2 b = 1 0.8157894736842105
MAP Valid Acc: a = 2 b = 2 0.8070175438596491
MAP Valid Acc: a = 2 b = 3 0.8070175438596491
MAP Valid Acc: a = 2 b = 4 0.8070175438596491
MAP Valid Acc: a = 2 b = 5 0.8070175438596491
MAP Valid Acc: a = 2 b = 6 0.8070175438596491
MAP Valid Acc: a = 2 b = 7 0.8070175438596491
MAP Valid Acc: a = 2 b = 8 0.8070175438596491
MAP Valid Acc: a = 2 b = 9 0.8070175438596491
MAP Valid Acc: a = 3 b = 1 0.8157894736842105
MAP Valid Acc: a = 3 b = 2 0.8157894736842105
MAP Valid Acc: a = 3 b = 3 0.8157894736842105
MAP Valid Acc: a = 3 b = 4 0.8157894736842105
MAP Valid Acc: a = 3 b = 5 0.8157894736842105
MAP Valid Acc: a = 3 b = 6 0.8157894736842105
MAP Valid Acc: a = 3 b = 7 0.8157894736842105
MAP Valid Acc: a = 3 b = 8 0.8070175438596491
MAP Valid Acc: a = 3 b = 9 0.8070175438596491
```

In our Bayesian inference model, our two hyperparameters are the a and b constants used in our MAP prediction. The above figure displays the output of running naive_bayes_hyperparameter.py. This is the python version of our final implementation of our naive Bayesian model. You can verify that this script computes the validation accuracy of 100 different combinations of a and b, both hyper-parameters in this situation. All values of a and b from 1 to 10 were tested. The results indicated that hyper-parameter choice did not impact validation accuracy by much, only setting the validation accuracy as low as 75%. Hence, in this case, the validation accuracy may be used as a proxy to estimate the test accuracy of our model. We now know, that we can expect our model to have 70-80% accuracy based on these results.

3.2 Model Choice

Since this model was the most accurate one out of all of the trialed models, we chose this model as our final one. Much of this decision was based off of our assessment of the various features of each data point. The most reliable and robust features with good specificity was determined to be the story. This was backed up by the accuracy results of all of our different models. Each one of our models specialized in a different area of the data, and our Bayesian model was the only one well adapted to the story feature, using the bag-of-words approach for 3 different classes.

3.3 Data-Splitting

In order to split our data, we randomly shuffled our entire data set, set roughly 80% (459 data points) as training data and took roughly 20% and set it aside to be validation data inside of our Naive Bayes MAP implementation.

4 Prediction

Our expected accuracy of our model on the test set would be approximately %70 - 80%. This is due to our above testing estimates using the validation set as a proxy for our naive Bayesian network. Our empirical evidence for this is our validation accuracies as obtained for a multitude of different hyperparamaters, as described in the hyper-parameter tuning section.

5 Workload Distribution

5.1 Antonio Salvatore La Placa

I did the first analysis and attempted data cleaning to view any spatially visible significance of the features `q_scary`, `q_dream`, and `q_desktop`. I created visualizations using heat maps utilizing `numpy` and `pandas`. I also fixed the Naive Bayes Maximum A Posteriori (MAP) implementation, debugging and correcting some major errors, investigating features and variables, as well as implementing the final version of the `make_prediction` function within `pred.py` and outputting the first final version of the working code with a validation accuracy of 81 percent, which originated from my solution from lab 8.

5.2 Michael Kwan

My work consisted of cleaning up the data in the file given, exploring the data with a k-nearest neighbours model, and part of creating this report file in latex (nearest neighbours parts). The files: `knn_pred.py`, `data_cleaner.py`, `knn_cleaned_data` were also created for the exploration of the data in k nearest neighbours.

In the file `knn_pred.py` which requires `data_cleaner.py` to help with cleaning data, it takes as input a data file (which is the file: `clean_quercus.csv`) and uses it to create a k-nearest neighbours model using scikit-learn. It also creates a normalized csv file that ends up being used in the exploration of the knn algorithm.

5.3 Konrad Wozniak

I created one of the many scripts used to clean the quercus dataset. Furthermore, I explored the usefulness of q_temp and q_sell as features to make predictions off of. I also explored the multiclass logistic regression model and tested its efficacy. I also preformed the hyper-parameter tuning for our final Bayesian model. Lastly, I made pred.py by taking our final model and storing its parameters in separate files, and setting up pred.py to be able to run our model properly.

5.4 Xingjian(James) Zhang

My work consisted of cleaning the data in the file that was given by professor, and apply it to the model naive Bayes that I've chose to use for the data exploration. I also worked on creating the report file based on what I've got from applying naive Bayes on `q_story`.
I've created both one pi and three pi model of naive Bayes to check the accuracy of the prediction, where one pi model provided the highest accuracy which is what we decided to use for our implementation of `pred.py`.