Problem set 4: Implement a BERT-based classifier

NetID: gff29

Name: Gabby Fite

People with whom you discussed this problem set:

Did you consult AI concerning any aspect of this problem set (see the class AI policy)? Yes

---

I used AI to help with some of the initial data processing, I wanted to convert the poems into strings rather than lists of strings. I asked chatgpt the best way to do this, it gave me the function apply(lambda lines: " ".join(lines)) which was helpful. I used chatgpt to help set up my charts for the final analysis. It helped with giving me the syntax to add the chart titles.

Instructions for submission

1. Supply your name, NetID, and other information above.
2. Submit your completed work via CMS.
3. **Remember to execute every cell of code! Unexecuted code will receive zero credit.** The best way to make sure that everything is in order is to restart your kernel and run all cells immediately prior to submission.
4. Make sure to print all outputs with informative labels.
5. You will submit *both* this fully executed notebook *and* a PDF copy of it. TAs will grade the PDF copy. TAs may refer to or run the code as necessary, but they will not execute it to fill in missing outputs.
6. To generate a PDF version of the completed notebook, export the notebook to HTML, open the resulting HTML file in your browser, and print the rendered HTML to PDF from your browser. You may produce the PDF in other ways, but you are solely responsible for verfiying that it is correct and complete.

Instructions

The corpus

Below is a link to a small set of Chinese poems from the Tang dynasty (618-907 CE), sourced from Leslie Wong.

The corpus comprises 97 poems. Poems are available in both Chinese and English. You may work with either language. The data also include a range of overlapping English-language subject tags for the poems.

The task

Your task is to design, implement, and evaluate a BERT (or other large language model)-based classifier that labels poems as either `nature` or `other`. `Nature` poems are those tagged with one or more of the tags indicated in the starter code below. Poems that do not have one of these tags are considered `other`.

You have broad latitude in how you approach this task. At minimum, you must:

1. Load the data from the indicated online source and process it into a form suitable for further analysis. This includes creating a `nature` column of gold labels where poems with one of the `nature` tags are assigned a `1` and poems without any of the `nature` tags are assigned a `0`.
2. Measure baseline classification performance using a `majority class` classifier and one of the token-based classifiers in the list below. Make sure to use sane parameters for both the tokenizer and the classifier:
   A. Naïve Bayes
   B. SVM
   C. Logistic regression
   D. Random forest
3. Design, implement, and evaluate the performance of a classifier based on (Distil)BERT or other large language model. Measure performance using one or more metrics directly comparable to the one(s) used for your baseline method.
4. Examine specific poems classified correctly and incorrectly by your baseline system(s) and your LLM-based system. Pay particular attention to cases where the systems disagree. Offer a reasoned analysis of the apparent strengths and weaknesses of the systems.
5. Carry out any additional experiments, analyses, or visualizations that help you and your reader understand the performance characteristics of your systems and/or the features of the poetry corpus.
6. Reflect on the overall performance of your system and evaluate any tradeoffs it presents relative to other possible approaches.

Notes and advice

- The corpus is small. How will you account for this fact as you configure both your classification systems and your evaluation of their performance?
- Your LLM-based system should achieve substantially better performance than most simple token-based baselines.

- Make sure your code is well organized and adequately commented. It should be clear to a reader what each step of your method accomplishes.
- Be sure to output information (in whatever form is appropriate) that summarizes what's happening at each step. For example, how many poems are in your dataset? How many of them belong to each labeled class? What baseline system are you using and how are you evaluating its performance?
- As ever, all outputs should be clearly labeled and formatted so that they are easy to read and to understand. This includes any figures, which should be thoughtfully constructed and well labeled.
- Your grade will be based on the performance of your system and on the rigor with which you evaluate and analyze it. Think carefully about every step of your process, document and justify your choices, and be thorough in your analyses and reflections. You have substantial latitude in this problem set; your grade depends on putting this flexibility to good use.
- It's probably easiest to work in Google Colab, but the dataset is small enough that you may be able to work locally if you prefer.

In [37]:
```
# corpus source url
poems_json = 'https://raw.githubusercontent.com/Leslie-Wong-H/Chinese-Poetry-Biling

# tags that indicate nature poems
nature_tags = set([
    'landscape',
    'spring',
    'mountain',
    'night',
    'windy',
    'summer',
    'winter',
    'autumn',
    'moon'
])
```

Setup

In [38]:
```
## your imports here
import pandas as pd
import numpy as np
import os
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from transformers import DistilBertTokenizerFast, DistilBertModel,DistilBertForSequ
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_recall_fscore_suppo
import torch
from transformers import Trainer, TrainingArguments
import os
```

Read corpus

In [39]:
```
## your code to read corpus json file into a dataframe
poems = pd.read_json(poems_json)

english_poems = [poem['content'] for poem in poems['English']]
english_tags = poems["tags"]

english_poems_df = pd.DataFrame({"content": english_poems, "tags": english_tags})


english_poems_df["content"] = english_poems_df["content"].apply(lambda lines: " ".j

english_poems_df.head(5)
```

Out[39]:

	content	tags
0	Oh, I return to the homeland I left while youn...	[homesick]
1	The slender tree is dressed in emerald all abo...	[windy, summer]
2	The yellow sand uprises as high as white cloud...	[border, homesick]
3	The sun beyond the mountain glows; The Yellow ...	[dream, ambition]
4	My boat is moored near an isle in mist grey; I...	[night, homesick]

In [40]:
```
nature_labels = []

for poem in english_poems_df['tags']:
    tag_set = set(poem)
    if tag_set & nature_tags:
        nature_labels.append(1)
    else:
        nature_labels.append(0)
english_poems_df['label'] = nature_labels
print("Count Nature Poems: ", nature_labels.count(1))
print("Count Non-Nature Poems: ",nature_labels.count(0))

print("New Data Frame:", english_poems_df.head(5))
```

```
Count Nature Poems:  36
Count Non-Nature Poems:  61
New Data Frame:                                                          content
tags  \
0  Oh, I return to the homeland I left while youn...           [homesick]
1  The slender tree is dressed in emerald all abo...      [windy, summer]
2  The yellow sand uprises as high as white cloud...   [border, homesick]
3  The sun beyond the mountain glows; The Yellow ...    [dream, ambition]
4  My boat is moored near an isle in mist grey; I...    [night, homesick]

   label
0      0
1      1
2      0
3      0
4      1
```

Baseline performance

In [41]:
```python
#splitting data into testing and training sets
train_data, test_data = train_test_split(english_poems_df, test_size=0.2, random_st
print(len(train_data))
print(len(test_data))
```

```
77
20
```

In [42]:
```python
# your majority class classifier
def majority_classifier(data, training_data):
    #We know from our calculations above that the most common label is non-nature
    label_counts = training_data["label"].value_counts()
    majority_label = 1
    if label_counts[0] > label_counts[1]:
        majority_label = 0
    predictions = []
    for d in data['content']:
        predictions.append(majority_label)
    return predictions

predictions_majority = majority_classifier(test_data, train_data)
print("Predictions: ", predictions_majority)
print("Actual: ", test_data['label'].tolist())
correct = []
incorrect = []


for i in range(0, len(test_data)):
    if test_data.iloc[i]['label'] == predictions_majority[i]:
        correct.append(test_data.iloc[i])
    else:
        incorrect.append(test_data.iloc[i])

print("\n")
print("Number of correct predictions: ", len(correct))
print("\n")
print("Number of incorrect predictions: ", len(incorrect))
print("\n")
print("Overall Accuracy: ", len(correct)/len(test_data))
```

```
Predictions:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Actual:  [1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0]


Number of correct predictions:  10


Number of incorrect predictions:  10


Overall Accuracy:  0.5
```

Note: For our randomly selected dataset, there are 50/50 nature vs not nature poems! This makes the baseline majority classifier have an precision score of 50% for our evaluation set.

In [62]:
```
# your token-based classifier
vectorizer = TfidfVectorizer(
    norm = "l2",
    lowercase = True
)

train_tokens = vectorizer.fit_transform(train_data['content'])
test_tokens = vectorizer.transform(test_data['content'])

train_labels = train_data['label']
test_labels = test_data['label']



random_forest = RandomForestClassifier(n_estimators=100, max_depth=20).fit(train_to

predictions_random_forest = random_forest.predict(test_tokens)
correct_naive = []
incorrect_naive = []

print("Random Forest Predictions: ", predictions_random_forest)
print(classification_report(test_labels.tolist(),
                            predictions_random_forest, zero_division=0))
```

```
Random Forest Predictions:  [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
              precision    recall  f1-score   support

           0       0.53      1.00      0.69        10
           1       1.00      0.10      0.18        10

    accuracy                           0.55        20
   macro avg       0.76      0.55      0.44        20
weighted avg       0.76      0.55      0.44        20
```

We notice that our Random Forest classifier is bad at classifying nature poems. This causes the f1-score and precision for the nature poems to be 0 or 1, if they it correctly classifies one poem. So when the random forest model does label a poem as nature, it usually is correct.

For the parameters, I chose to decrease the number of possible trees (n_estimators) to be 100 since that is closer to the training set size. Additionally, in order to increase the f1-score I increased the allowable depth of the tree to 20. This increased the f1-score but did not help with precision.

Overall this classifier was best at labeling non-nature poems, which is detrimental to its accuracy. But it has a high f1 score for correctly identifying most of the non-nature poems which make up a majority of this data set. Hence, the slightly unbalanced nature of this data set, along with its small size, may have influence the classifier to perform this way.

DistilBERT

In [44]:
```
# DistilBertTokenizer
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
train_encodings = tokenizer(train_data['content'].tolist(), truncation=True, paddin
test_encodings  = tokenizer(test_data['content'].tolist(), truncation=True, padding
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:160
1: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `Tru
e` by default. This behavior will be depracted in transformers v4.45, and will be th
en set to `False` by default. For more details check this issue: https://github.com/
huggingface/transformers/issues/31884
  warnings.warn(
```

For the tokenizer, I set truncation=True, padding=True, and max_length=512 to makes the set of poems more uniform. The max_length is set to 512 since the maximum sequence length for bert models. I set return tensors to true since I was getting an error message that recommended.

In [45]:
```
# MyDataset class
#Used the same setup as the MyDataset in lecture
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)
```

In [46]:
```
train_dataset = MyDataset(train_encodings, train_labels.tolist())
test_dataset = MyDataset(test_encodings, test_labels.tolist())

bert_model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels= 2
).to("cuda")
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the mo
del checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bi
as', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
```

In [47]:
```
# TrainingArguments
training_args = TrainingArguments(
    output_dir='./results',
    logging_dir='./logs',
    num_train_epochs=6,
    per_device_train_batch_size=10,
    per_device_eval_batch_size=5,
    warmup_steps=10,
    logging_strategy ='epoch',
    eval_strategy='epoch',
)
```

```
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in
v5. Use the --report_to flag to control the integrations used for logging result (fo
r instance --report_to none).
```

For the training args, I decided to evaluated based on epochs since it was more intuitive for me. Additionally, I trained on 6 epochs, since after testing different configurations the f1-score and validation lost remained relatively the same after epoch 6. Additionally, made the batch size 10 since is about 70/6, very roughly. I then made the evaluation batch size 5 since I thought a two to one ratio for the testing vs eval batches would be useful to the model. Since I was evaluating in epochs I also chose to log in epochs.

```
In [48]: # fine-tuning
         def f1_score_method(pred):
           labels = pred.label_ids
           preds = pred.predictions.argmax(-1)
           score = f1_score(labels, preds, average='weighted')
           return {
               'f1': score,
           }
         trainer = Trainer(
             model=bert_model,
             args=training_args,
             train_dataset=train_dataset,
             eval_dataset=test_dataset,
             compute_metrics=f1_score_method
         )

         os.environ["WANDB_DISABLED"] = "true"

         trainer.train()
         trainer.save_model('distilbert-nature-poems')
```

```
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
```

[48/48 00:25, Epoch 6/6]

Epoch	Training Loss	Validation Loss	F1
1	0.667400	0.719965	0.333333
2	0.631500	0.770998	0.333333
3	0.541000	0.691527	0.333333
4	0.368300	0.697594	0.601140
5	0.246600	0.780324	0.601140
6	0.160100	0.940211	0.523810

```
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
```

In [49]:

```
# evaluate performance
trainer.evaluate()
predicted_results_bert = trainer.predict(test_dataset)
predicted_labels = predicted_results_bert.predictions.argmax(-1)
predicted_labels_bert = predicted_labels.flatten().tolist()
print(classification_report(test_labels,
                            predicted_labels_bert))
```

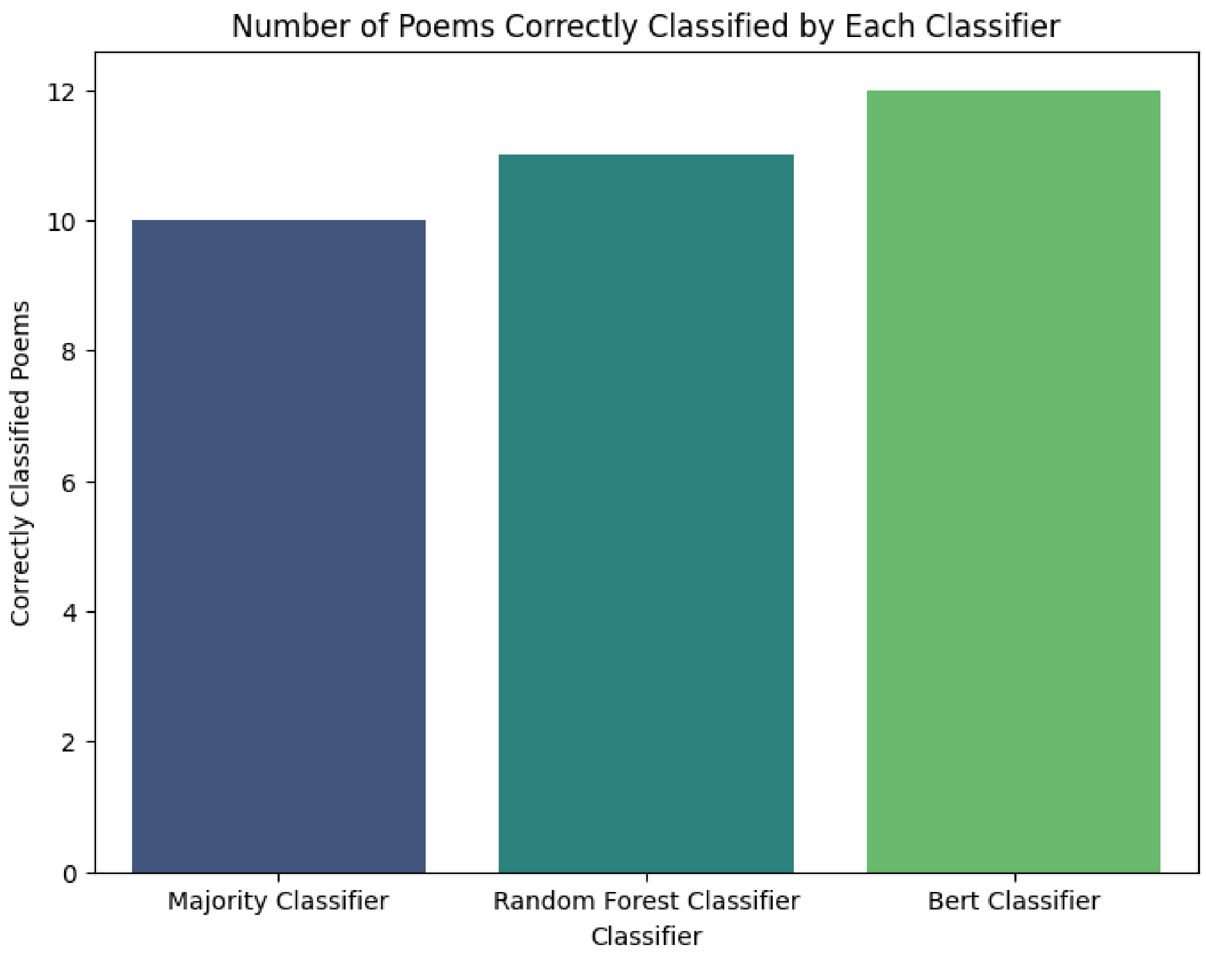
```
<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

<ipython-input-45-06f10affe382>:9: UserWarning: To copy construct from a tensor, it
is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach
().requires_grad_(True), rather than torch.tensor(sourceTensor).
  item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
```

```
              precision    recall  f1-score   support

           0       0.56      1.00      0.71        10
           1       1.00      0.20      0.33        10

    accuracy                           0.60        20
   macro avg       0.78      0.60      0.52        20
weighted avg       0.78      0.60      0.52        20
```

Examine poems

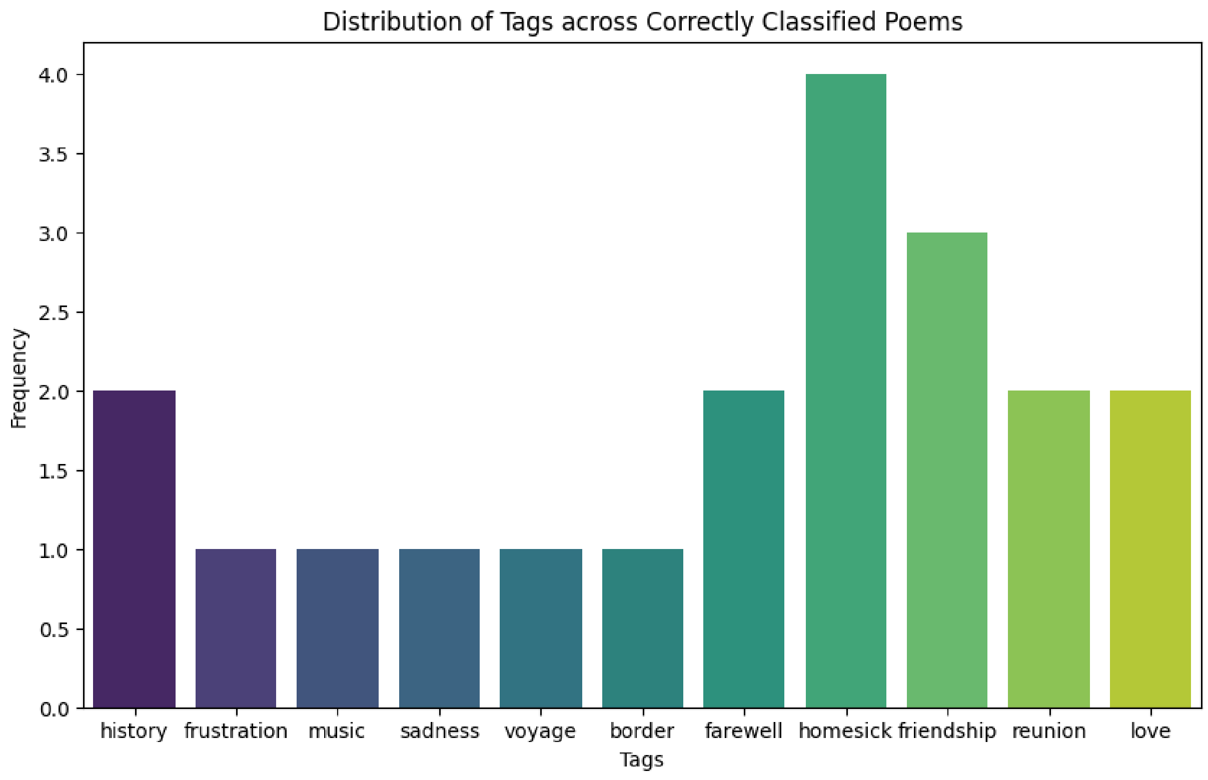
In [63]:
```
## examine poems classified correctly
correct_bert = []
incorrect_bert = []
correct_random_forest = []
incorrect_random_forest = []
correct_majority = []
incorrect_majority = []
test_labels_list = test_labels.tolist()

disagree_bert_correct = []
disagree_bert_false = []

missed_by_all = []
correct_by_all = []
for i in range(0, len(test_labels)):
  c = 0
  w = 0
  check_br = 0
  check_bw = 0
  real_label = test_labels_list[i]
  poem = test_data['content'].iloc[i]
  all_data = test_data.iloc[i]
  if real_label == predicted_labels_bert[i]:
    correct_bert.append(all_data)
    c += 1
    check_br += 1
  else:
    incorrect_bert.append(all_data)
    check_bw += 1
    w += 1
  if real_label == predictions_random_forest[i]:
    correct_random_forest.append(poem)
    c += 1
    check_bw += 1
  else:
    incorrect_random_forest.append(poem)
    w += 1
    check_br += 1
  if real_label == predictions_majority[i]:
    correct_majority.append(poem)
    c += 1
    check_bw += 1
  else:
    incorrect_majority.append(poem)
    w += 1
    check_br += 1
  if c == 3:
    correct_by_all.append(all_data)
  if w == 3:
    missed_by_all.append(all_data)
  if check_br == 3:
    disagree_bert_correct.append(all_data)
  if check_bw == 3:
    disagree_bert_false.append(all_data)
```

```
data = {
    'Classifier': ['Majority Classifier', 'Random Forest Classifier'
    , 'Bert Classifier'],
    'Correctly Classified Poems': [len(correct_majority), len(correct_random_forest
                                   len(correct_bert)]
}

results_df = pd.DataFrame(data)
plt.figure(figsize=(8, 6))
sns.barplot(x='Classifier', y='Correctly Classified Poems',
            data=results_df, hue = 'Classifier', palette='viridis')

plt.title('Number of Poems Correctly Classified by Each Classifier')
plt.xlabel('Classifier')
plt.ylabel('Correctly Classified Poems')

plt.show()
```



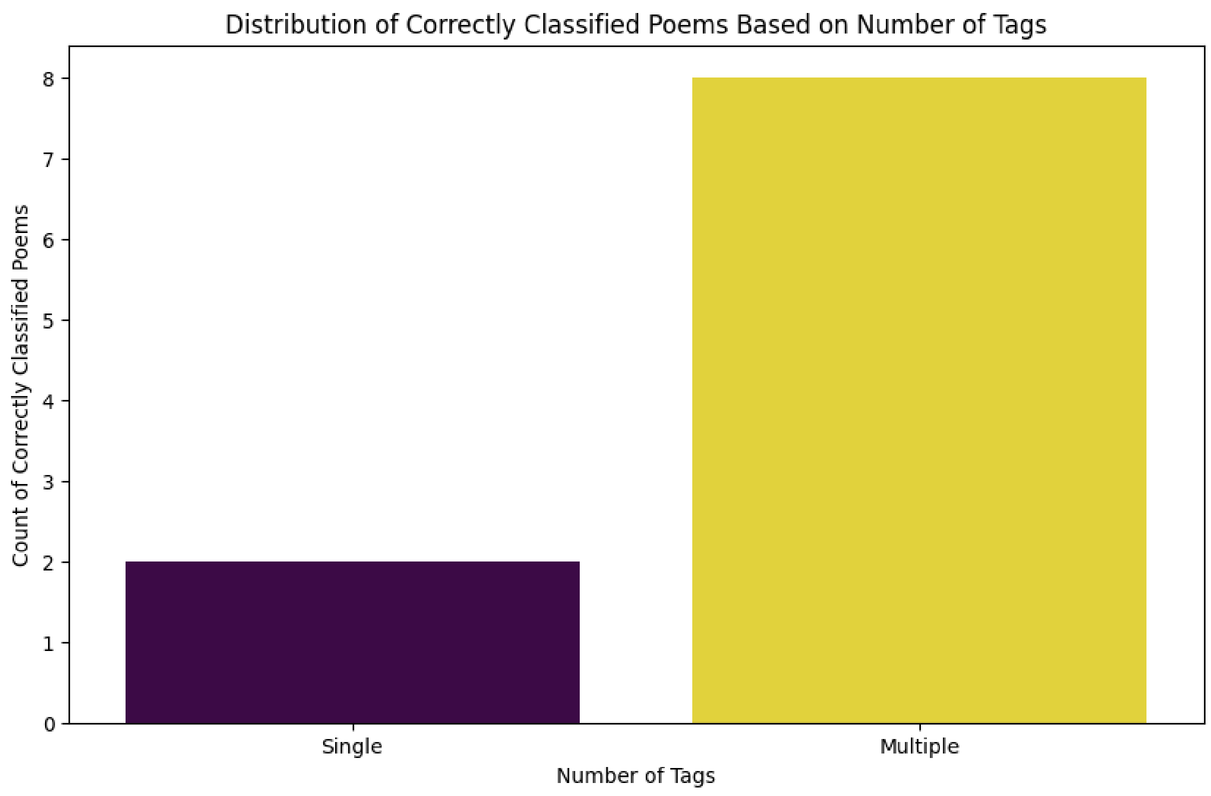
Analysis of Poems that All Classifiers Labeled Correctly

In [64]:
```
correct_tags = []
correct_labels = []

for x in correct_by_all:
  tags = x['tags']
  for t in tags:
    correct_tags.append(t)
  correct_labels.append(x['label'])

unique_tags = list(set(correct_tags))

tag_counts = []

for t in unique_tags:
  tag_counts.append(correct_tags.count(t))

#print(correct_tags)
data_2 = {
    "Tags" : unique_tags,
    "Counts" : tag_counts
}

tags_dataframe = pd.DataFrame(data_2)


plt.figure(figsize=(10, 6))
sns.barplot(x='Tags', y='Counts',
            data=tags_dataframe, hue = 'Tags',palette='viridis')
plt.title('Distribution of Tags across Correctly Classified Poems')
plt.xlabel('Tags')
plt.ylabel('Frequency')

plt.show()
```



We notice that these are all not nature tags. This makes sense since all of the systems correctly identified all of the non-nature poems. Additionally, we see that poems with the tags homesick and friendship were the most successful at being correctly classified. Friendship and Homesick both correspond to a non-nature labeling. Hence, this visualization demonstrates how all of the systems did well labeling the non-nature poems. The trade-off of this precision in non-nature poems will be addressed in the section about the poems we did not classify correctly.

In [65]:
```
combo_tags = []
non_combo_tags = []
for x in correct_by_all:
  tags = x['tags']
  if len(tags) > 1:
    combo_tags.append(tags)
  else:
    non_combo_tags.append(tags)



data_combo = {
    "Number of Tags" : ['Single', 'Multiple'],
    "Counts" : [len(non_combo_tags), len(combo_tags)]
}
tags_combo_dataframe = pd.DataFrame(data_combo)


plt.figure(figsize=(10, 6))
sns.barplot(x='Number of Tags', y='Counts',
            data=tags_combo_dataframe, hue = 'Counts',palette='viridis'
            , legend = False)
plt.title('Distribution of Correctly Classified Poems Based on Number of Tags')
plt.xlabel('Number of Tags')
plt.ylabel('Count of Correctly Classified Poems')

plt.show()
```



We notice here that the classifiers did a good job classifying poems with multiple parameters. However, upon further investigation, most of the poems had more than one tag. Thus, this statistic was not as influential as I was expecting. In future experiments it would be interesting to see how poems with similar tags compared in how accurate the classifiers could label them.

This Section Covers Poems that the Bert Classifier correctly classified but Baselines did not Correctly Classify

In [66]:
```
print("Number of Poems that Baseline got Wrong but Bert got Right:"
      , len(disagree_bert_correct))
print("Number of Poems that Baseline got Right but Bert got Wrong:",
      len(disagree_bert_false))

tags_disagree_br = []
poems_disagree_br = []
labels_disagree_br = []
for d in disagree_bert_correct:
  tags = d['tags']
  for t in tags:
    tags_disagree_br.append(t)
  poems_disagree_br.append(d['content'])
  labels_disagree_br.append(d['label'])
print("Tags that stumped the baseline classifiers:", tags_disagree_br)
print("\n")
print("Labels that stumped the baseline classifiers:", labels_disagree_br)
print("\n")
print("Poems that stumped the baseline classifiers")
for p in poems_disagree_br:
  print(p)
  print("\n")

#These are all nature poems
```

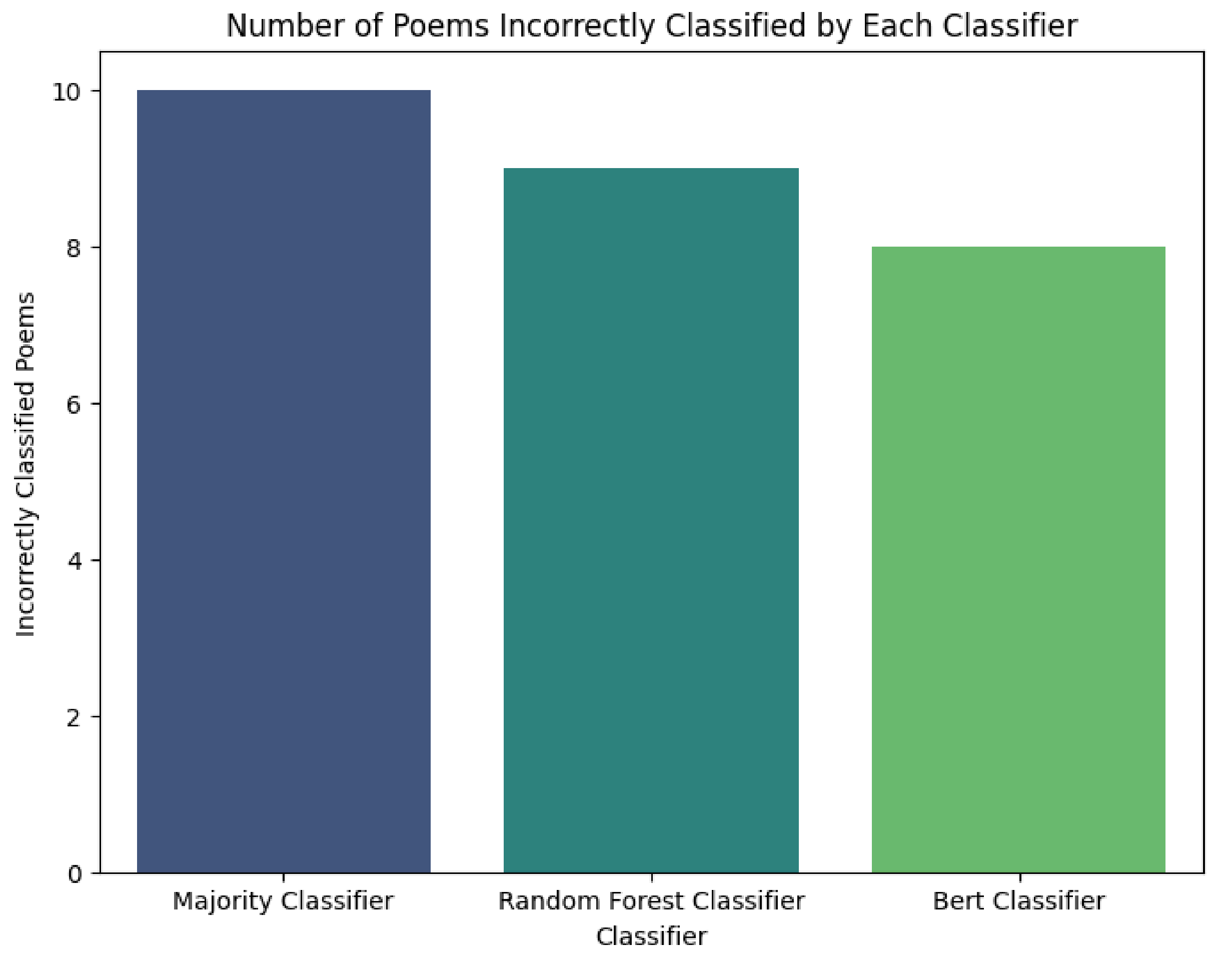
```
Number of Poems that Baseline got Wrong but Bert got Right: 2
Number of Poems that Baseline got Right but Bert got Wrong: 0
Tags that stumped the baseline classifiers: ['spring', 'solitude', 'history', 'sprin
g']


Labels that stumped the baseline classifiers: [1, 1]


Poems that stumped the baseline classifiers
Alone I like the riverside where green grass grows And golden orioles sing amid the
leafy trees. When showers fall at dusk, the river overflows; A lonely boat athwart t
he ferry floats at ease,


Beside the Bridge of Birds rank grasses overgrow; Over the Mansion Street the settin
g sun hangs low. Swallows that skimmed by painted eaves in bygone days Are dipping n
ow among the humble homes' doorways,
```

We see that the two poems where both baseline disagreed with the bert are nature poems. Additionally, I noticed that these poems look to be longer than average. That also may be a reason the baseline classifiers failed to predict correctly.

For the Examination of the Poems Classified Incorrectly, I performed the same analysis

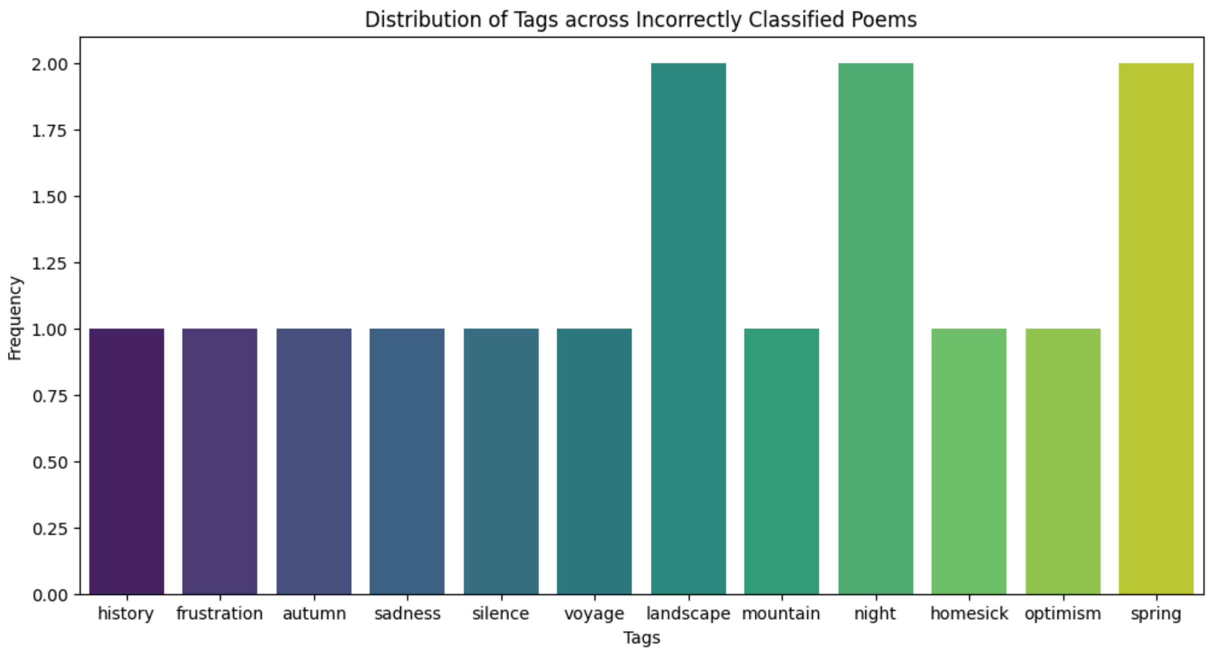
In [68]:
```
## examine poems classified incorrectly
print("Number of Poems Misclassified by all classifiers: ", len(missed_by_all))
print("\n")
data = {
    'Classifier': ['Majority Classifier', 'Random Forest Classifier'
    , 'Bert Classifier'],
    'Incorrectly Classified Poems': [len(incorrect_majority), len(incorrect_random_
                                   len(incorrect_bert)]
}

results_df = pd.DataFrame(data)
plt.figure(figsize=(8, 6))
sns.barplot(x='Classifier', y='Incorrectly Classified Poems',
            data=results_df, hue = 'Classifier', palette='viridis')

plt.title('Number of Poems Incorrectly Classified by Each Classifier')
plt.xlabel('Classifier')
plt.ylabel('Incorrectly Classified Poems')

plt.show()
```

```
Number of Poems Misclassified by all classifiers:  7
```



In [69]:
```
incorrect_tags = []
incorrect_labels = []

for x in missed_by_all:
  tags = x['tags']
  for t in tags:
    incorrect_tags.append(t)
  incorrect_labels.append(x['label'])

unique_tags = list(set(incorrect_tags))

tag_counts = []

for t in unique_tags:
  tag_counts.append(incorrect_tags.count(t))

#print(correct_tags)
data_3 = {
    "Tags" : unique_tags,
    "Counts" : tag_counts
}

tags_dataframe = pd.DataFrame(data_3)
```

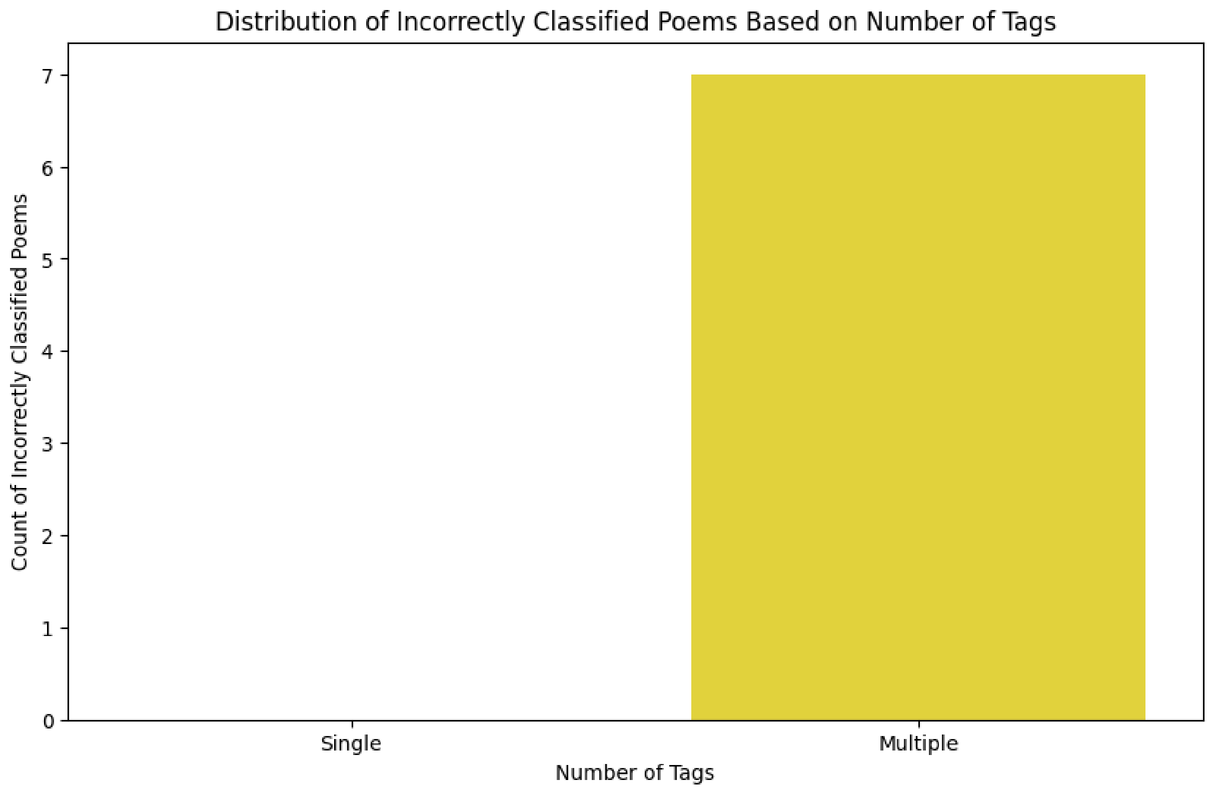
```
plt.figure(figsize=(12, 6))
sns.barplot(x='Tags', y='Counts',
            data=tags_dataframe, hue = 'Tags',palette='viridis')
plt.title('Distribution of Tags across Incorrectly Classified Poems')
plt.xlabel('Tags')
plt.ylabel('Frequency')

plt.show()
```



The tags that got misclassified the most are the nature tags. This aligns with our findings since we found that all of the classifiers had a bias towards the non-nature poems. Hence, although there is a mix of both non-nature and nature tags in the overall visualization, we can see that the tags that got mislabeled the most are the nature tags.

In [70]:
```
combo_tags = []
non_combo_tags = []
for x in missed_by_all:
  tags = x['tags']
  if len(tags) > 1:
    combo_tags.append(tags)
  else:
    non_combo_tags.append(tags)



data_combo_2 = {
    "Number of Tags" : ['Single', 'Multiple'],
    "Counts" : [len(non_combo_tags), len(combo_tags)]
}
tags_combo_dataframe_2 = pd.DataFrame(data_combo_2)
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Number of Tags', y='Counts',
            data=tags_combo_dataframe_2, hue = 'Counts',palette='viridis'
            , legend = False)
plt.title('Distribution of Incorrectly Classified Poems Based on Number of Tags')
plt.xlabel('Number of Tags')
plt.ylabel('Count of Incorrectly Classified Poems')

plt.show()
```



In [71]:
```
print(len(disagree_bert_false))
#for this iteration there are no poems where the bert classifier failed to classify
#classified correctly
```

```
0
```

In [72]:
```
print(len(incorrect_bert))

for p in incorrect_bert:
  print(p)
  print("\n")

print("POEMS")
print("\n")
for p in incorrect_bert:
  print(p['content'])
  print("\n")
```

```
8
content    Oho! Behold! How steep! How high! The road to ...
tags                          [mountain, history, frustration]
label                                                        1
Name: 62, dtype: object


content    Under western cliff a fisherman passes the nig...
tags                                          [night, silence]
label                                                        1
Name: 40, dtype: object


content    What a charming picture when spring comes to t...
tags                                       [landscape, spring]
label                                                        1
Name: 93, dtype: object


content    Good rain knows its time right; It will fall w...
tags                                       [spring, landscape]
label                                                        1
Name: 83, dtype: object


content    No dust is raised on the road wet with morning...
tags                                          [spring, voyage]
label                                                        1
Name: 10, dtype: object


content    A drizzling rain falls like tears on the mourn...
tags                                      [sadness, landscape]
label                                                        1
Name: 31, dtype: object


content    Since olden days we fel in autumn sad and drea...
tags                                       [optimism, autumn]
label                                                        1
Name: 26, dtype: object


content    My boat is moored near an isle in mist grey; I...
tags                                        [night, homesick]
label                                                        1
Name: 4, dtype: object


POEMS


Oho! Behold! How steep! How high! The road to Shu is harder than to climb the sky. S
ince the two pioneers Put the kingdom in order, Have passed forty-eight thousand yea
rs, And few have tried to pass its border. There's a bird track o'er Great White Mou
ntain to the west, Which cuts through Mountain Eyebrows by the crest. The crest crum
```

bled, five serpent-killing heroes slain, Along the cliffs a rocky path was hacked th
en. Above stand peaks too high for the sun to pass o'er; Below the torrents run back
and forth, churn and roar. Even the Golden Crane can't fly across; How to climb ove
r, gibbons are at a loss. What tortuous mountain path Green Mud Ridge faces! Around
the top we make nine turns each hundred paces, Looking up breathless, I can touch th
e stars nearby; Beating my breast, I sink aground with long, long sigh. When will yo
u come back from this journey to the west? How can you climb up dangerous path and m
ountain crest, Where you can hear on ancient trees but sad birds wail And see the fe
male birds fly, followed by the male? And hear home-going cuckoos weep Beneath the m
oon in mountains deep? The road to Shu is harder than to climb the sky, On hearing t
his, your cheeks would lose their rosy dye. Between the sky and peaks there is not a
foot's space, And ancient pines hang, head down, from the cliffs' surface, And catar
acts and torrents dash on boulders under, Roaring like thousands of echoes of thunde
r. So dangerous these places are, Alas! Why should you come here from afar? Rugged i
s the path between the cliffs so steep and high, Guarded by one And forced by none.
Disloyal guards Would turn wolves and pards, Man-eating tigers at day-break And at d
usk blood-sucking long snake. One may make merry in the Town of Silk, I know, But I
would rather homeward go. The road to Shu is harder than to climb the sky, I'd turn
and westward look with long long sigh.

Under western cliff a fisherman passes the night; At dawn he makes bamboo fire to bo
il water clean. Mist clears off at sunrise but there's no man in sight; Only the fis
herman's song turns hill and rill green. He goes down mid-stream and turns to look o
n the sky. What does he see but clouds freely wafting on high.

What a charming picture when spring comes to the lake! Amid the rugged peaks water's
smooth without a break. Hills upon hills are green with thousands of pine trees, The
moon looks like a pearl swimming in waves with ease. Like a green carpet early paddy
fields undulate, New rushes spread out as silk girdle fascinate. From fair Hangzhou
I cannot tear myself away, On half my heart this lake holds an alluring sway.

Good rain knows its time right; It will fall when comes spring. With wind it steals
in night; Mute, it moistens each thing. O'er wild lanes dark cloud spreads; In boat
a lantern looms. Dawn sees saturated reds; The town's heavy with blooms.

No dust is raised on the road wet with morning rain; The willows by the hotel look s
o fresh and green. I invite you to drink a cup of wine again; West of the sunny pass
no more friends will be seen.

A drizzling rain falls like tears on the mourning day; The mourner's heart is going
to break on his way. "Where can a wine-shop be found to drown my sad hours?" A cowhe
rd points to a cot amid apricot flowers.

Since olden days we fel in autumn sad and drear, But I say spring cannot compete wit
h autumn clear On a fine day a crane cleaves the clouds and soars high; It leads the
poet's lofty mind to azure sky.

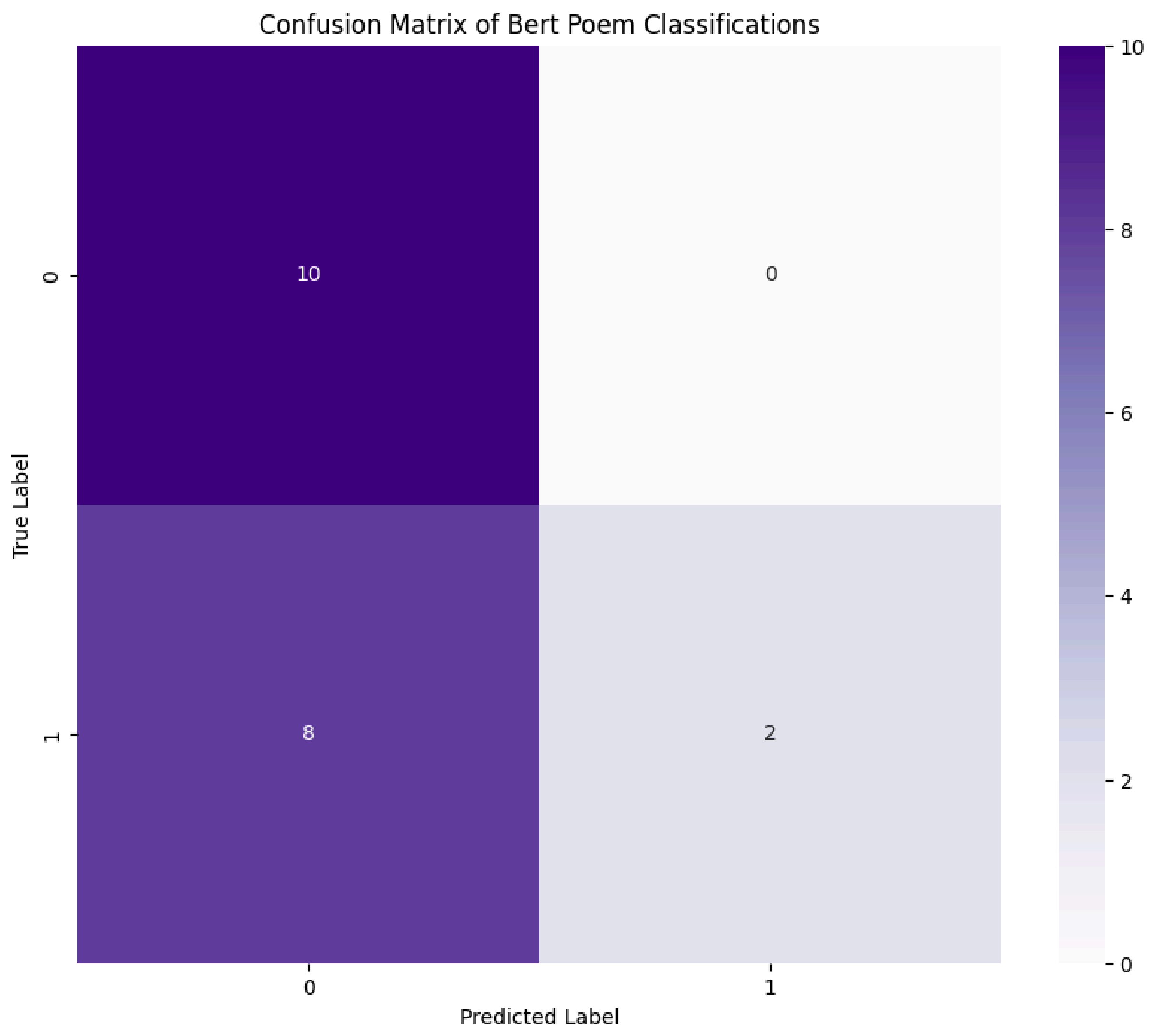
My boat is moored near an isle in mist grey; I'm grieved anew to see the parting da
y. On boundless plain trees seem to scrape the sky; In water clear the moon appears

```
so nigh.
```

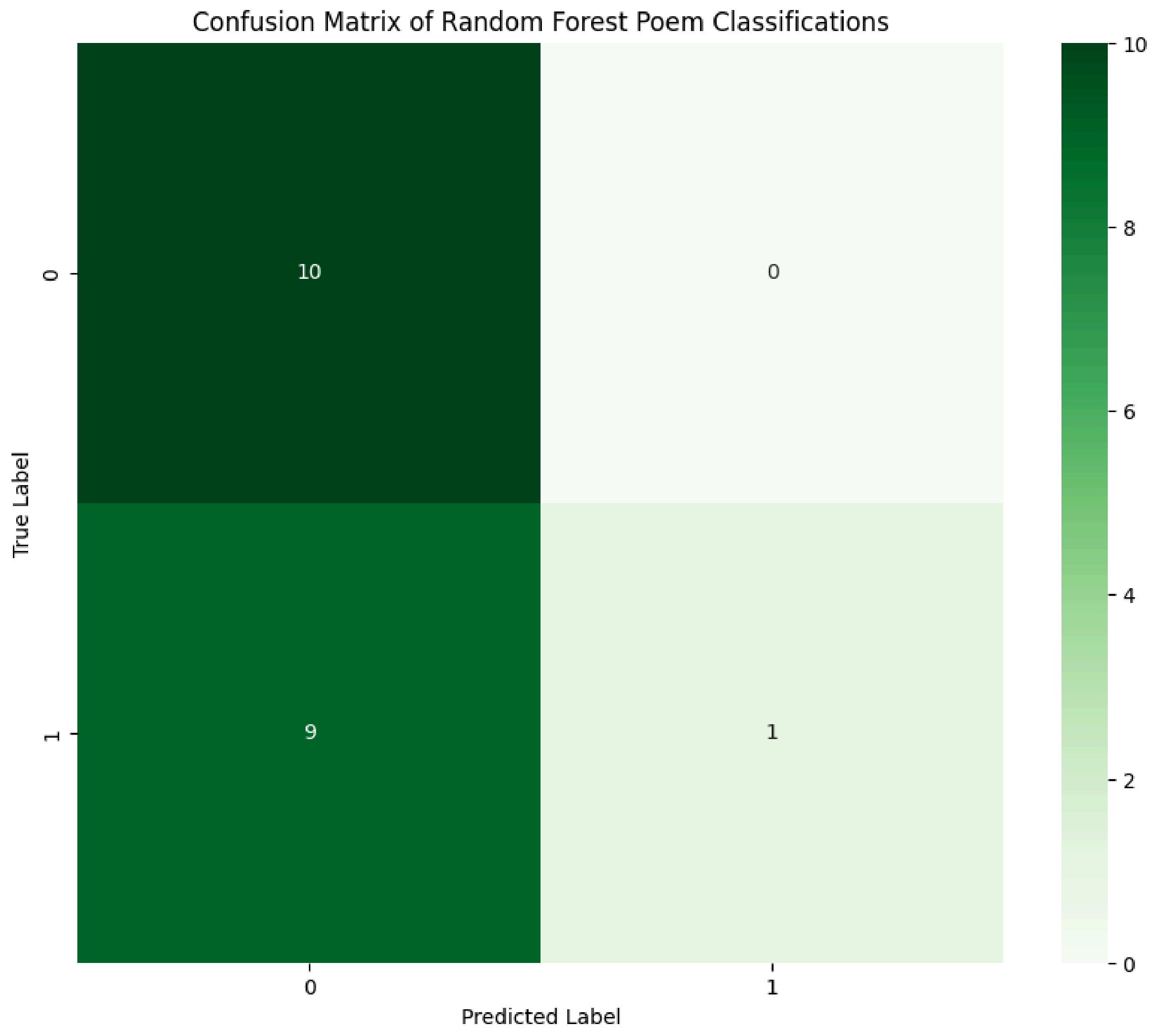
We note here that all of the poems that Bert misclassified are nature poems, additionally they all have more than 1 tag. My bert model struggles with non-nature poems. But we notice in the variety of the poems that there is no connection to the poor classification and length. Additionally, there does not seems to be a connection between less obvious nature tags and very obvious nature tags. We see that both poems with "landscape" and poems with "night" are misclassified. Both poems are nature, but night I would argue is not as obviously a nature tag/subject. So further investigation is needed as to why these specific poems were misclassified.

Confusion Matrices

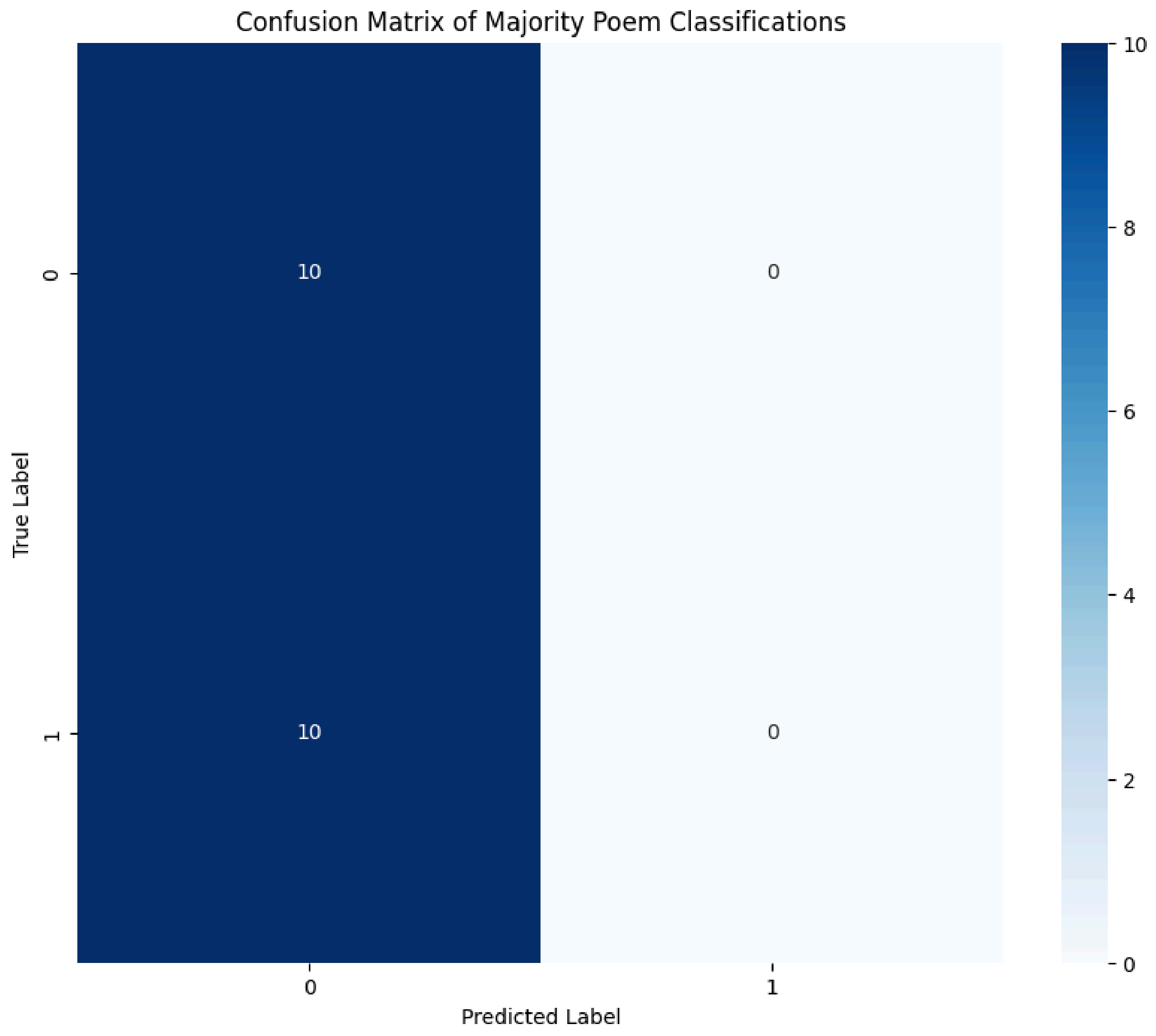
In [73]:
```python
cm = confusion_matrix(test_labels, predicted_labels_bert, labels=[0, 1])
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples", xticklabels=[0,1], yticklabels
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix of Bert Poem Classifications")
plt.show()
```



```
In [74]: cm_2 = confusion_matrix(test_labels, predictions_random_forest, labels=[0, 1])
         plt.figure(figsize=(10, 8))
         sns.heatmap(cm_2, annot=True, fmt="d", cmap="Greens", xticklabels=[0,1], yticklabel
         plt.xlabel("Predicted Label")
         plt.ylabel("True Label")
         plt.title("Confusion Matrix of Random Forest Poem Classifications")
         plt.show()
```



```
In [75]: cm_3 = confusion_matrix(test_labels, predictions_majority, labels=[0, 1])
         plt.figure(figsize=(10, 8))
         sns.heatmap(cm_3, annot=True, fmt="d", cmap="Blues", xticklabels=[0,1], yticklabels
         plt.xlabel("Predicted Label")
         plt.ylabel("True Label")
         plt.title("Confusion Matrix of Majority Poem Classifications")
         plt.show()
```



We note that for all of the classifiers, none misclassified any non-nature poem as a nature poem. All of our classifiers are bias towards non-nature poems.
This is obvious in the **majority classifier** since the most common label in our data set is non-nature.
Additionally, for the **random forest model**, since we have such a small dataset with most of the non-nature poems being in the training data, our random forest is better at predicting non-nature poems. However, I noticed in testing that when the random forest model does predict a poem to be a nature poem, which it did very rarely, it was correct with a 100% accuracy for its nature predictions. Hence, the trade-off to having this very high accuracy for its nature predictions is that most of the evaluated poems did not meet the threshold to be considered nature. Hence, the overall accuracy was not any better than the baseline.

For the **bert classifier** we were able to predict most of the nature poems correctly. Similar to the random forest model there was never an instance where the bert classifier labeled a non-nature poem as nature. But the bert classifier was a vast improvement from baseline since it was able to classify about 60% of the nature poems correctly. If we were to improve upon the bert classifier more, it would be beneficial to find a way to negate the bias for the non-nature poems. We could try to do this with having a less-random training set with 50/50 nature, non-nature.