Executive Report: Siblings Or Dating Classification Model

By Max Votaw

1. Problem

It can be quite challenging to determine if 2 people are siblings or dating based on only a photograph. In the real world, you can draw conclusions based on the social interactions between the two people, but based only on a single photograph the distinction is a lot harder to make. Without knowing the classification already, siblings can seem very intimate with each other while romantic partners can seem very facially similar, which leads to a lot of confusion. An internet meme that asks whether 2 parties are siblings or dating got quite popular, and led to the creation of siblings or dating social media accounts. People are often split about 50/50 on most images hosted by these accounts, evidenced by the polls run by said accounts.

2. Objective

The goal of the project is to use machine learning to determine if the people are siblings or dating. Instead of just using pixel-based analysis, this project uses a multi-model approach, implementing aspects of pose detection, facial recognition features, and raw image data to come to the most accurate conclusion.

3. Data Collection and Preprocessing

I pulled data from 2 main data sources:

- The subreddit r/siblingsordating
- The instagram account @siblingsordating

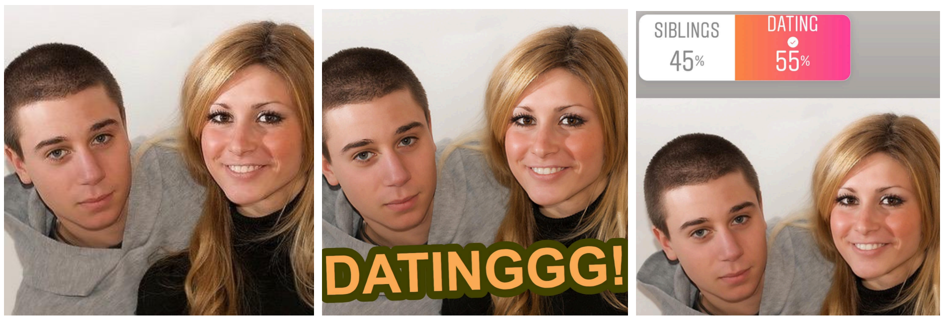
To label the reddit data, I used praw to extract every post in siblingsordating and search for obfuscated text in either the post description or comments by the original poster to determine the correct label for each image. Here is a sample of one of the posts' output to my parser:

```
post_url,image_url,label,op_comment
https://reddit.com/r/siblingsordating/comments/1kfxeuo/siblings_or_dating/,
https://i.redd.it/aovtadtvq3ze1.jpeg,dating,They are >!dating!<
```

I searched for the obfuscation text in the comments, ">!...!<, and used fuzzy matching to account for small alterations in the text such as "datinggg", "siblings!" or "soblings".

The instagram data was more complicated to extract, as the label was placed on the image itself in subsequent slides instead of text comments. To label the instagram data, I used PaddleOCR to extract text from the labeled slides, ignore any slides with poll data, and then label the original image with fuzzy matching based on the label slide. Here is a sample of 3 slides and my code's output that I labeled.

```
text: datinggg
[✓] 2364759404618559674_39225567241_1.jpg: dating
```



I ignored slides that contained the "%" character, as this told me it was a poll and that I needed to search the other slide. As you can see, I needed to use fuzzy matching to ensure that text that didn't explicitly match "siblings" or "dating" was still correctly labeled. These two data sources were my original dataset.

After labeling, I resized all images to 224x224 pixels and converted them to RGB to make sure the data was consistent. I used face detection to find and align both people in each photo, keeping both faces together as one input for the model. Then, I applied a pose estimation model to get body key points for both individuals and saved these as feature vectors. These key points, along with the original images, were used together as input for the neural network. Finally, I split the data into training, validation, and test sets with a 70%, 15%, and 15% split to make sure the model could be properly trained and evaluated.

4. Evaluation Metrics

To ensure that my model is performing well, I used a combination of accuracy, precision score, recall score, and f1 score.

```
acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred)
rec = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
print(f'Accuracy: {acc}, Precision: {prec}, Recall: {rec}, F1: {f1}')
```

I used a few different validations methods to verify my model's performance.

1. **Accuracy** measures how often the model gets the answer right out of all the predictions it makes.

2. **Precision** tells you how many classifications were actually correct, a high precion means few false positives.
3. **Recall** finds the true positive cases. A high recall means low false negatives.
4. **F1 score** balances precision and recall into a single number. This is important because sometimes a model might be very precise but miss a lot (low recall), or catch many but also make more mistakes (low precision). The F1 score gives an overall sense of performance when you want to weigh precision and recall equally

Because siblings vs. dating is a binary task where both types of mistakes matter, it’s important to use all these metrics to get a complete picture of how well the model performs.

5. Model Architecture and Training

I built a model that uses a Convolutional Neural Network to analyze images. A CNN works by looking at small parts of an image to find patterns like edges and shapes, gradually learning to recognize complex features such as faces. This CNN extracts important features from the photos of both people. I also use a pose estimation model MediaPipe to get body position data, and these are combined to make a final prediction.

```
mtcnn = MTCNN(image_size=160, margin=20, keep_all=False, post_process=True,
device=device)
facenet = InceptionResnetV1(pretrained='vggface2').eval().to(device)

mp_pose = mp.solutions.pose
pose_model = mp_pose.Pose(static_image_mode=True,
min_detection_confidence=0.5)
```

6. Evaluation and Fine-Tuning

I used a validation set of images that the model has never seen before to evaluate the model. This checks that the model wasn’t overfitted on the training data, as I want it to learn the features that define whether 2 individuals are siblings or dating, not learn the images themselves. I also tried different hyperparameters to improve its performance

```
acc = accuracy_score(all_labels, all_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(all_labels,
all_preds, average='weighted')
print(f"Accuracy: {acc:.4f}")
```

```
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")
```

7. Testing and Final Results

After training and tuning, I tested the model on a completely new set of images to see how well it generalizes. The test accuracy shows how well the model predicts whether two people in a photo are siblings or dating.

```
print("Final Test Set Evaluation:")
validate(model, test_loader)
```

```
Final Test Set Evaluation:
Accuracy: 0.7143
Precision: 0.7091
Recall: 0.7143
F1-score: 0.7022
0.7142857142857143
```

The model achieved a test accuracy of 71.4%, with precision, recall, and F1-score all around 70%. I polled some students in RISC on 10 different images, and their scores averaged 75% accuracy on the same task. This means that the model is performing close to the Bayes error rate. This indicates the model has learned useful patterns and performs reasonably well given the visual ambiguity of distinguishing siblings from couples. Unfortunately, my model was overfitting quite a bit, as the validation accuracy plateaued around 60%.

8. Future Goals

As you can see in my final results, my model performed relatively well but feel victim to overfitting. In the future, I want to improve my model by increasing the dataset adding more images would help to reduce overfitting, and hopefully increase my validation accuracy. Also, the main source of data for my model were these social media accounts, where the question of siblings or dating is intentionally ambiguous. If I were to add more data that depicted less ambiguous data, it might help my model better learn the differences between siblings and dating images, and then perform better on the more ambiguous images. I also want to preprocess the images a bit more, like applying filters and tilting them various degrees so that the model has more data to work with, and will overfit less.