Final Report

THC Usage Cognitive Impairment Classification using fNIRS Data

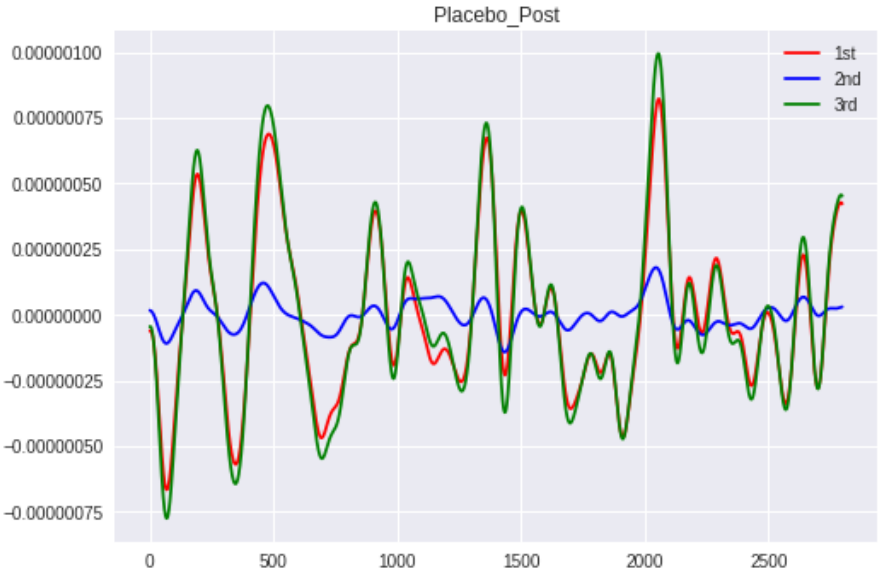
NetID: yy852

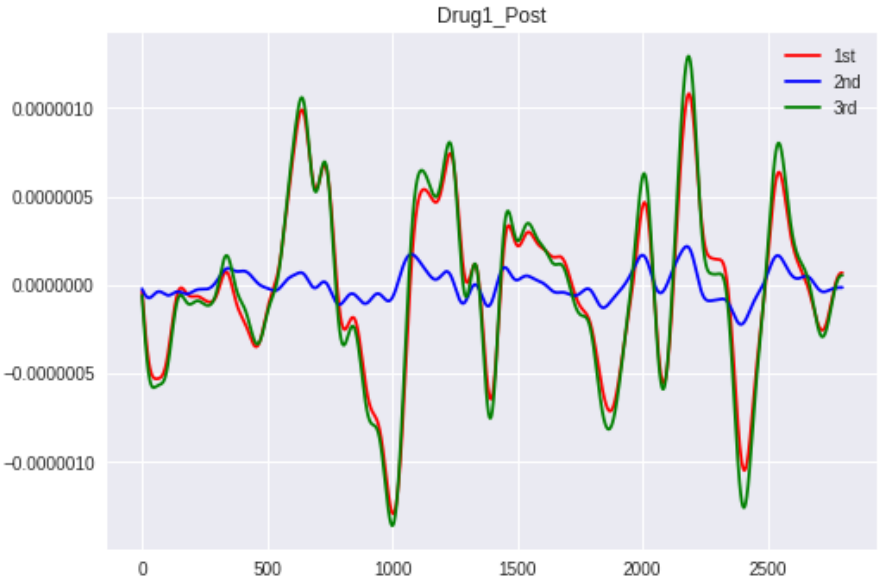
Name: Yiming Yan

1. Overview

In this project, data includes 25 subjects with impairment and 61 subjects without impairment. I try **Linear SVM with L1 penalty**, **Linear SVM with L2 penalty**, **RBF kernel SVM**, **Random Forest** and **LSTM** to make predictions. As for data augmentation, I use different combinations of number of sliding windows and length of windows. Then, I use dynamic correlation for LSTM model.

All training data use 1st data, which shows below.





2. Methods and Results

In all SVM and random forest models, I use 20% data for testing and 80% data for training. Also, I used 4-fold cross validation. Each fold and testing data are guaranteed for two kinds of labels, impairment and non-impairment. Taking into account subjects differences, training, testing and validation data are split on subject level. I use different sliding window settings for data augmentation and vectorize the higher half diagonal elements. All correlations are normalized by Fisher-Z transform.

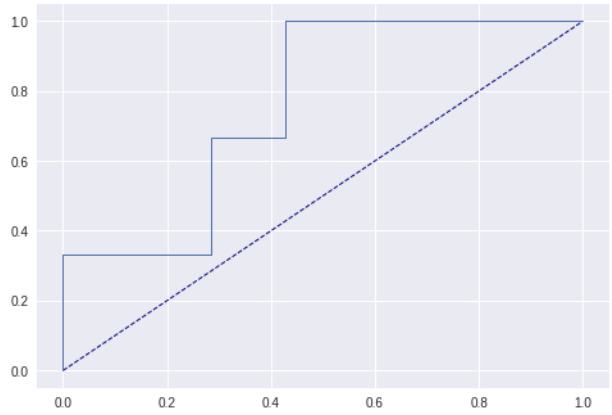
Each method has two kinds of accuracy in the following results show. '*crossvalid*' means the accuracy on 4-fold cross validation and '*Test acc*' means the prediction accuracy for this model on the testing data.

2.1 SVM and Random Forest

- **Use original data without data augmentation**

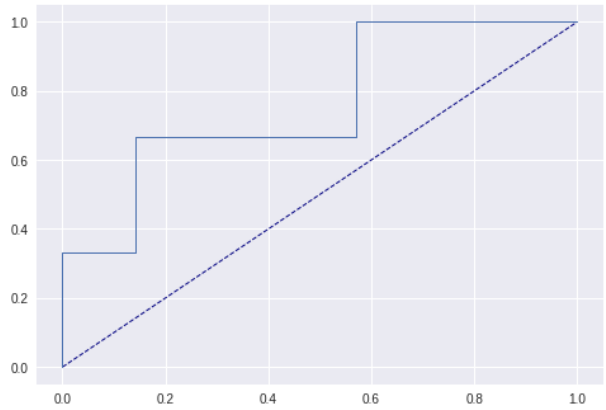
1. LinearSVC with L1 penalty

```
crossvalid: 0.526316, 0.473684, 0.684211, 0.684211
Test acc: 0.500000, 0.500000, 0.500000, 0.800000
```



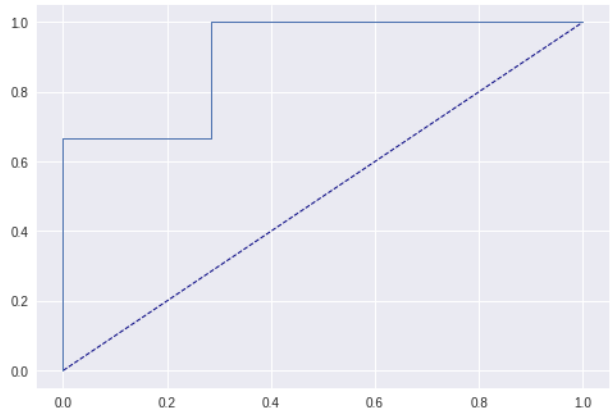
2. LinearSVC with L2 penalty

```
crossvalid: 0.578947, 0.473684, 0.684211, 0.631579
Test acc: 0.700000, 0.600000, 0.600000, 0.700000
```



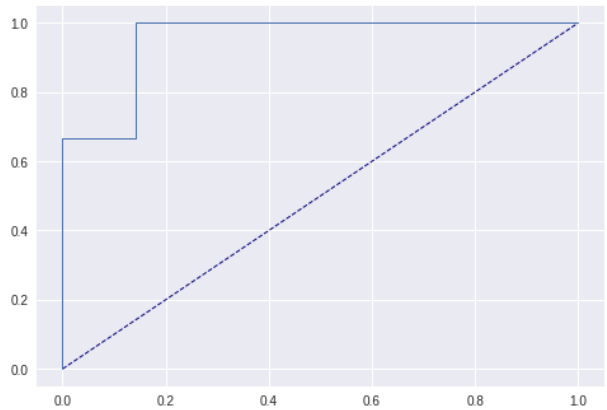
3. RBF kernel SVM

```
crossvalid: 0.736842, 0.631579, 0.684211, 0.789474
Test acc: 0.700000, 0.700000, 0.700000, 0.700000
```



4. NuSVM

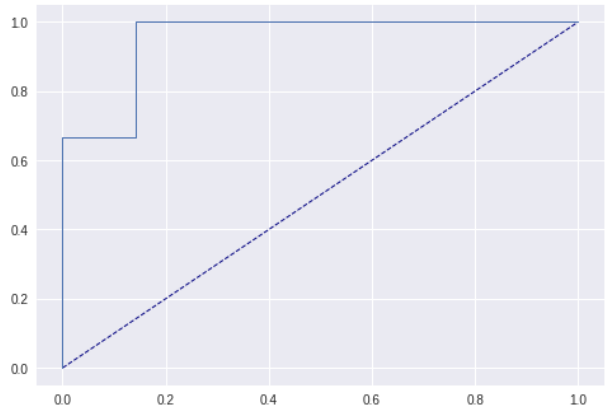
```
crossvalid: 0.578947, 0.473684, 0.578947, 0.526316
Test acc: 0.700000, 0.700000, 0.700000, 0.800000
```



5. Random Forest

```
crossvalid: 0.736842, 0.368421, 0.631579, 0.578947
```

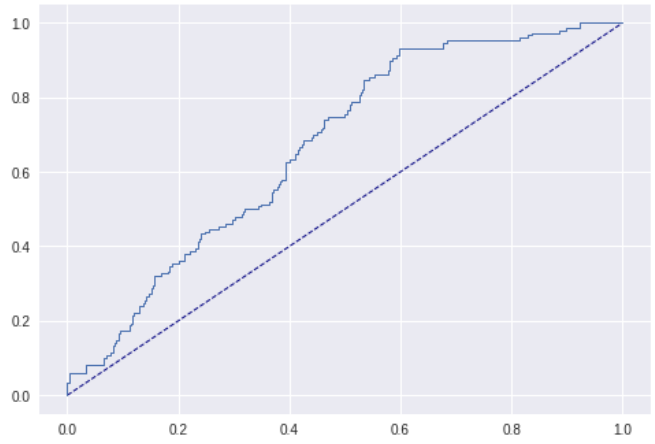
```
Test acc: 0.700000, 0.700000, 0.800000, 0.800000
```



- **50 windows, length 500**

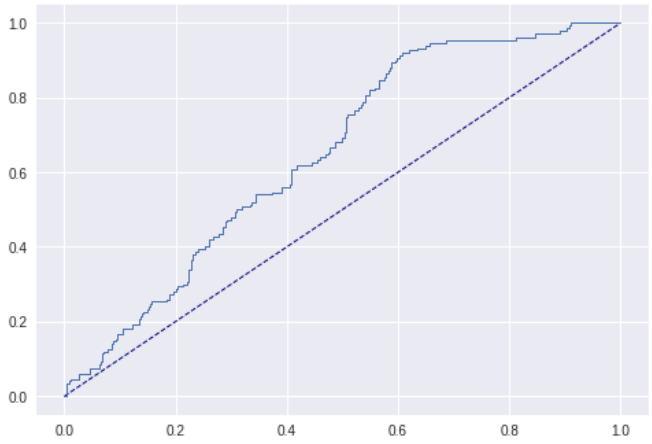
1. LinearSVC with L1 penalty

```
crossvalid: 0.531579, 0.640000, 0.713684, 0.524211
Test acc: 0.578000, 0.662000, 0.548000, 0.574000
```



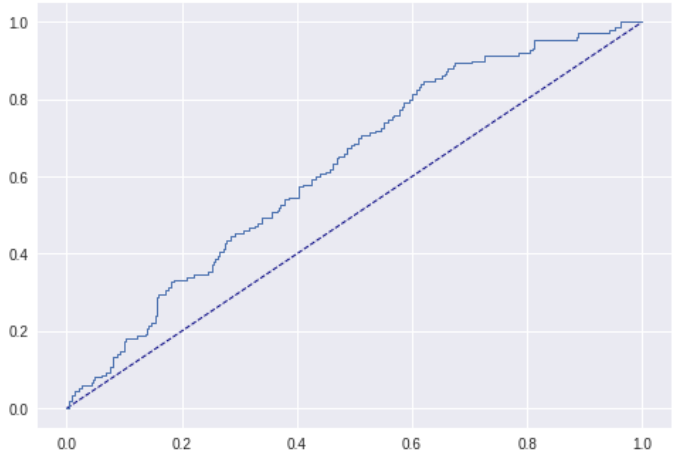
2. LinearSVC with L2 penalty

```
2crossvalid: 0.530526, 0.628421, 0.721053, 0.529474
Test acc: 0.588000, 0.662000, 0.548000, 0.562000
```



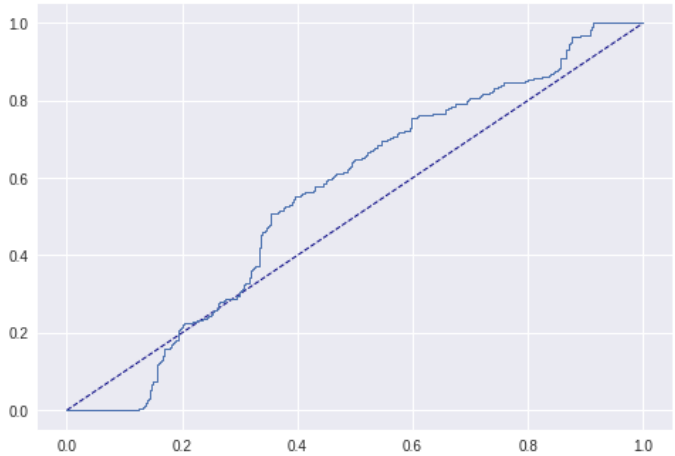
3. RBF kernel SVM

```
crossvalid: 0.584211, 0.694737, 0.681053, 0.526316
3Test acc: 0.674000, 0.666000, 0.616000, 0.568000
```



4. Random Forest

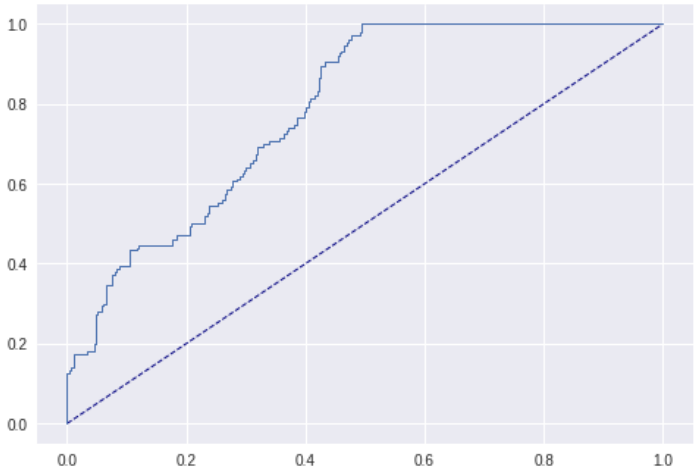
```
crossvalid: 0.643158, 0.674737, 0.733684, 0.583158
Test acc: 0.684000, 0.654000, 0.666000, 0.584000
```



- **50 windows, length 1000**

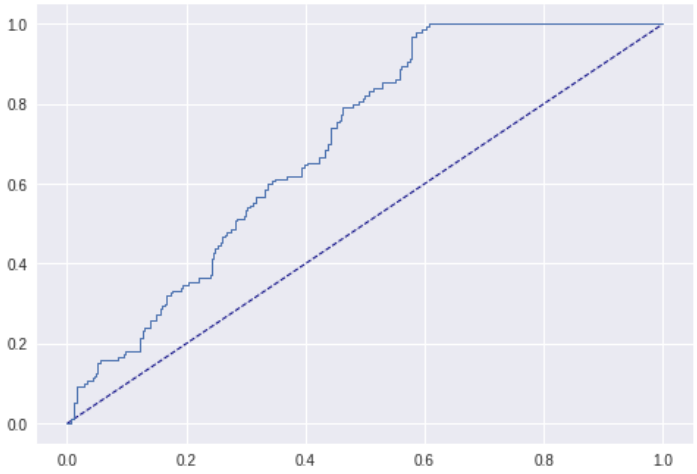
1. LinearSVC with L1 penalty

```
crossvalid: 0.558947, 0.510526, 0.574737, 0.625263
Test acc: 0.722000, 0.858000, 0.788000, 0.678000
```



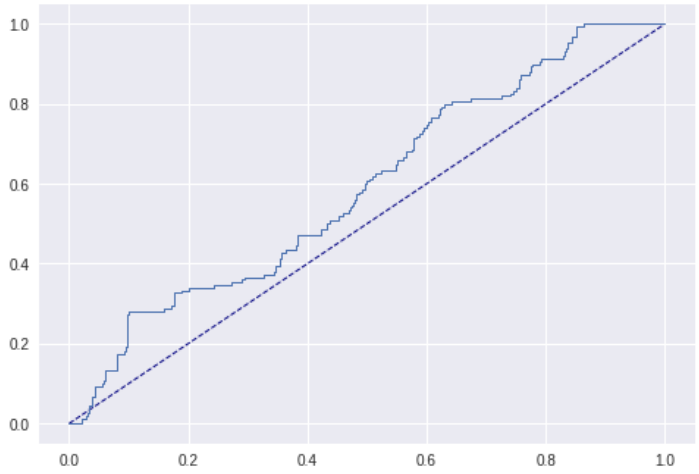
2. LinearSVC with L2 penalty

```
crossvalid: 0.560000, 0.482105, 0.613684, 0.672632
Test acc: 0.758000, 0.798000, 0.754000, 0.654000
```



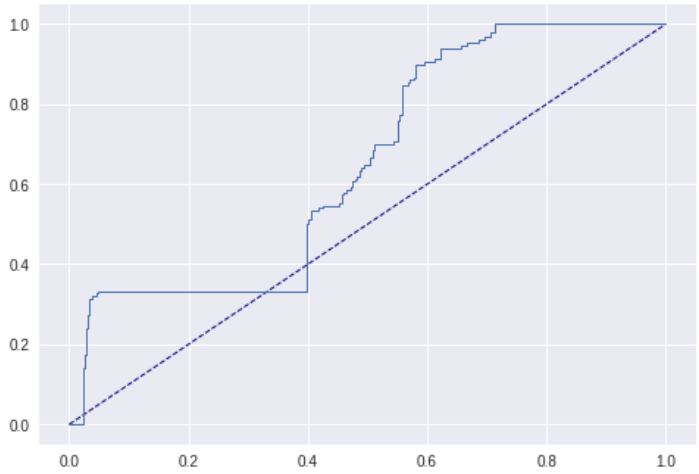
3. RBF kernel SVM

```
crossvalid: 0.529474, 0.517895, 0.645263, 0.647368
3Test acc: 0.638000, 0.728000, 0.754000, 0.574000
```



4. Random Forest

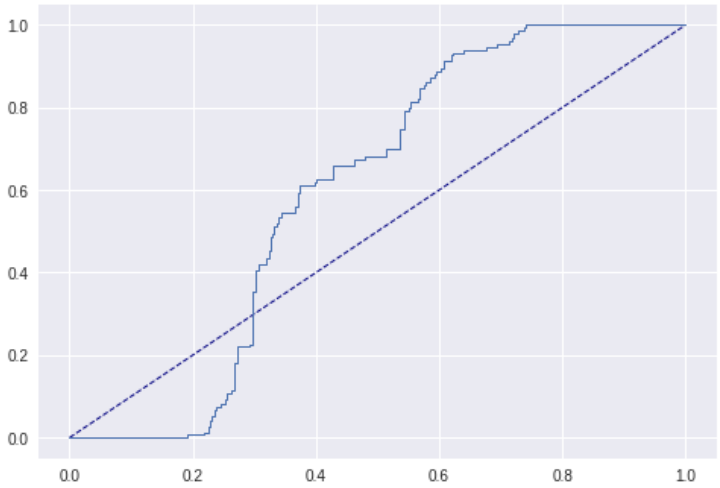
```
crossvalid: 0.629474, 0.710526, 0.717895, 0.638947
4Test acc: 0.686000, 0.710000, 0.676000, 0.662000
```



- **50 windows, length 1500**

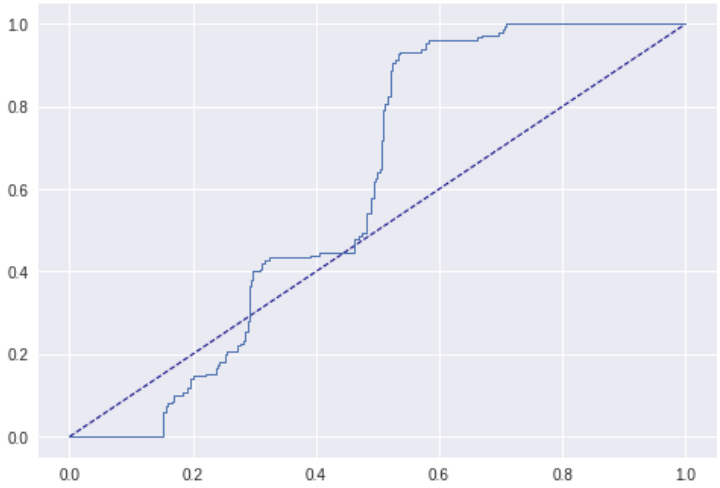
1. LinearSVC with L1 penalty

```
crossvalid: 0.637895, 0.474737, 0.808421, 0.656842
Test acc: 0.452000, 0.682000, 0.546000, 0.594000
```



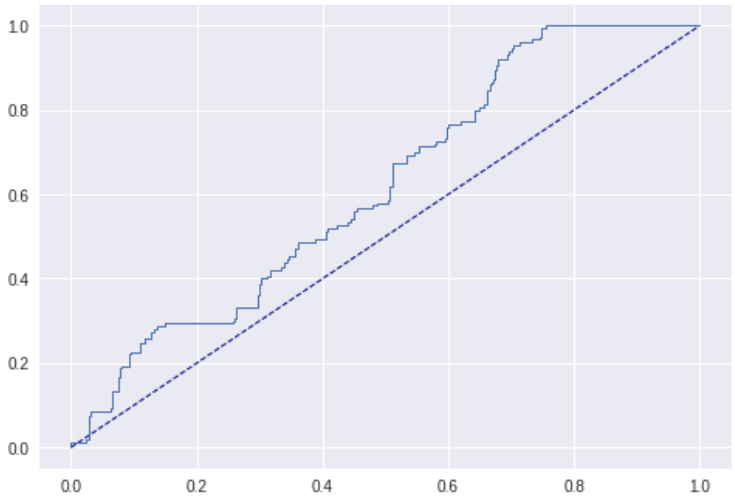
2. LinearSVC with L2 penalty

```
crossvalid: 0.534737, 0.529474, 0.805263, 0.626316
Test acc: 0.498000, 0.524000, 0.500000, 0.606000
```



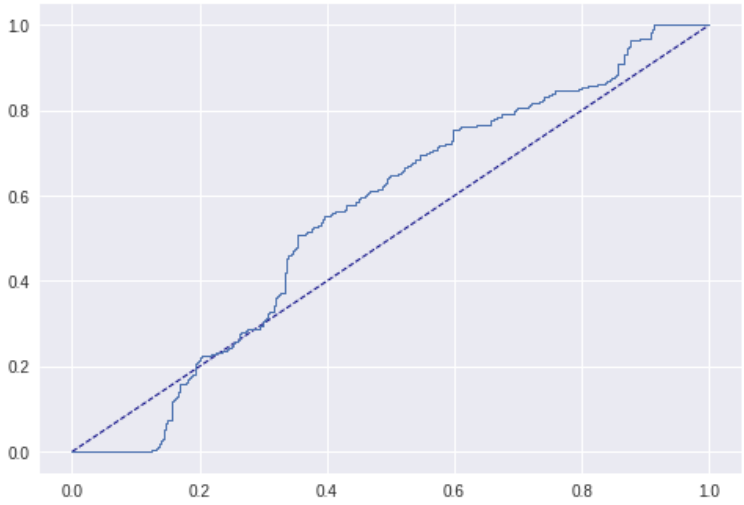
3. RBF kernel SVM

```
crossvalid: 0.586316, 0.577895, 0.808421, 0.578947
Test acc: 0.590000, 0.678000, 0.526000, 0.596000
```



4. Random Forest

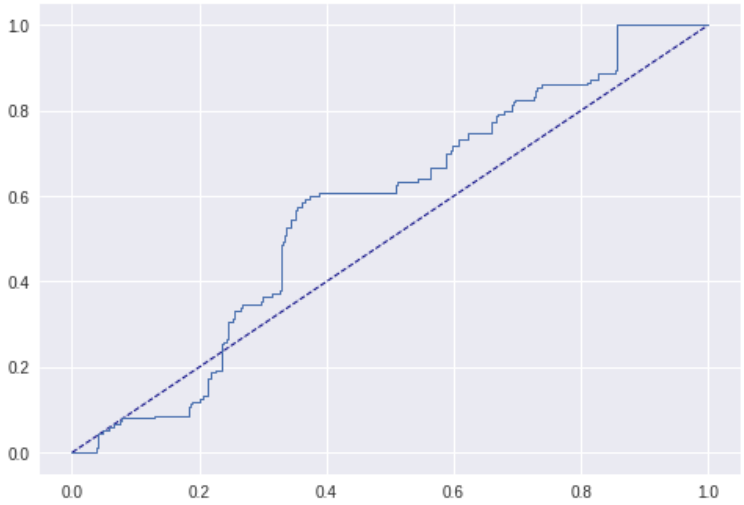
```
crossvalid: 0.546316, 0.623158, 0.834737, 0.613684
Test acc: 0.612000, 0.666000, 0.552000, 0.764000
```



- **50 windows, length 2000**

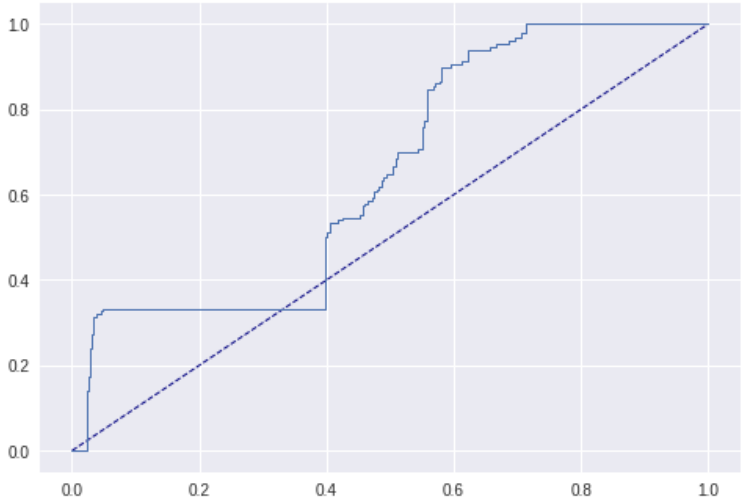
1. LinearSVC with L1 penalty

```
crossvalid: 0.515789, 0.536842, 0.451579, 0.611579
Test acc: 0.540000, 0.738000, 0.542000, 0.614000
```



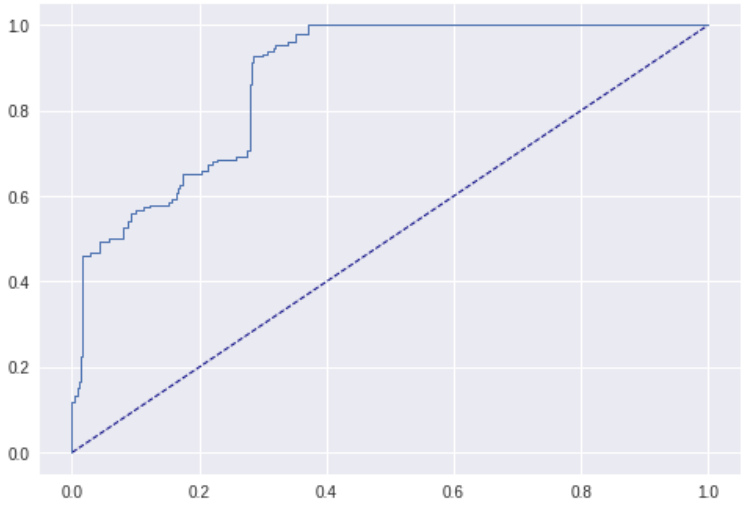
2. LinearSVC with L2 penalty

```
crossvalid: 0.535789, 0.569474, 0.411579, 0.625263
Test acc: 0.616000, 0.700000, 0.538000, 0.540000
```



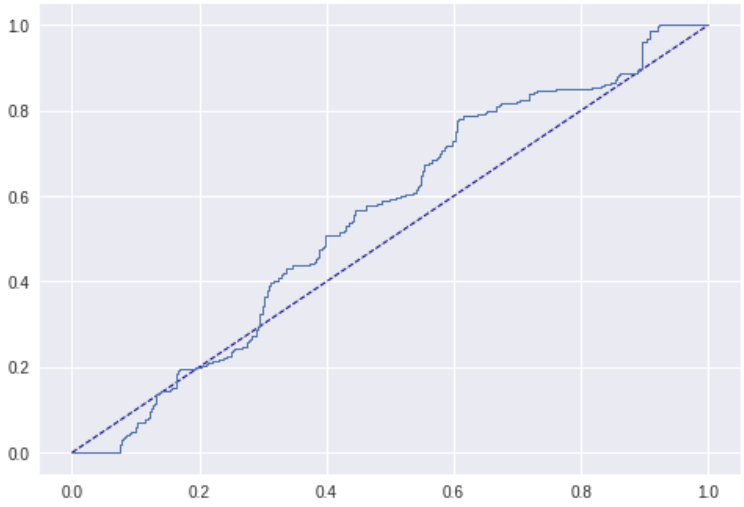
3. RBF kernel SVM

```
crossvalid: 0.482105, 0.542105, 0.521053, 0.666316
Test acc: 0.638000, 0.920000, 0.774000, 0.740000
```



4. Random Forest

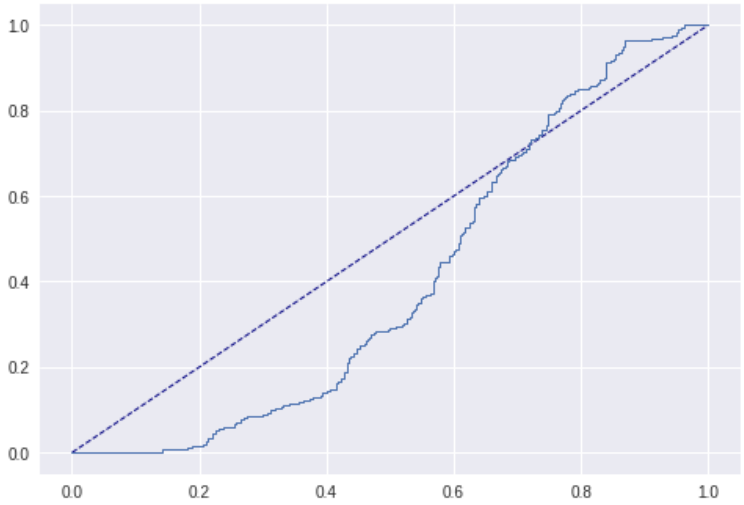
```
crossvalid: 0.643158, 0.669474, 0.544211, 0.754737
Test acc: 0.568000, 0.712000, 0.696000, 0.59600
```



- **100 windows, length 500**

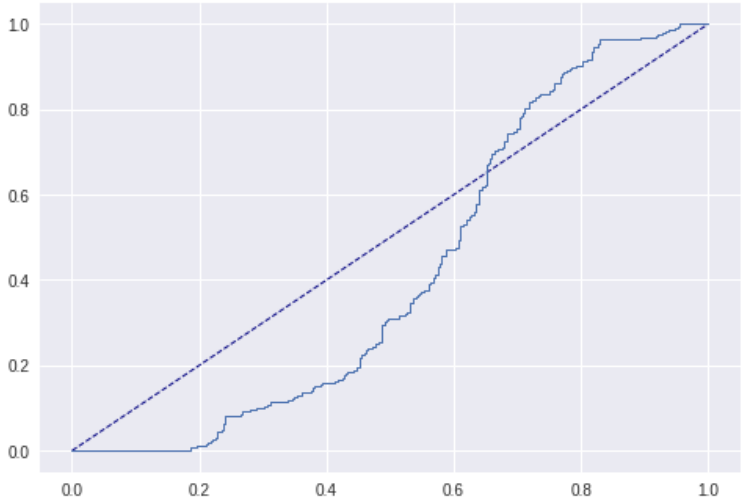
1. LinearSVC with L1 penalty

```
crossvalid: 0.614211, 0.724211, 0.601053, 0.668421
Test acc: 0.667000, 0.592000, 0.550000, 0.530000
```



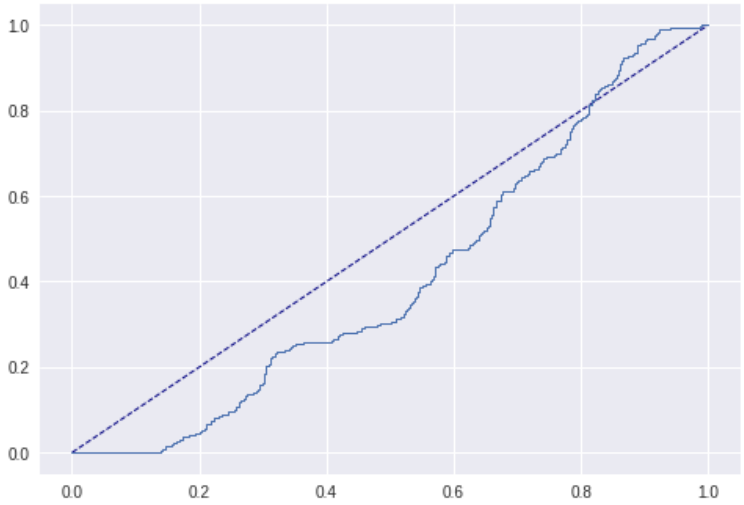
2. LinearSVC with L2 penalty

```
crossvalid: 0.613158, 0.712105, 0.603684, 0.648947
Test acc: 0.652000, 0.594000, 0.562000, 0.536000
```



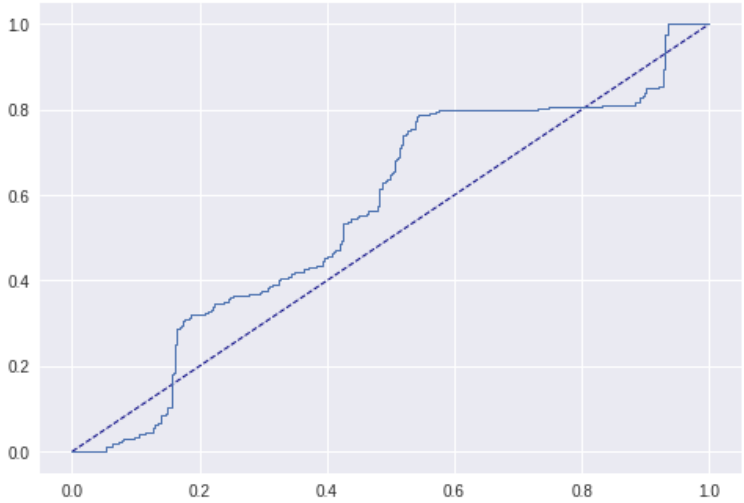
3. RBF kernel SVM

```
crossvalid: 0.598421, 0.714211, 0.638947, 0.610526
Test acc: 0.704000, 0.625000, 0.650000, 0.557000
```



4. Random Forest

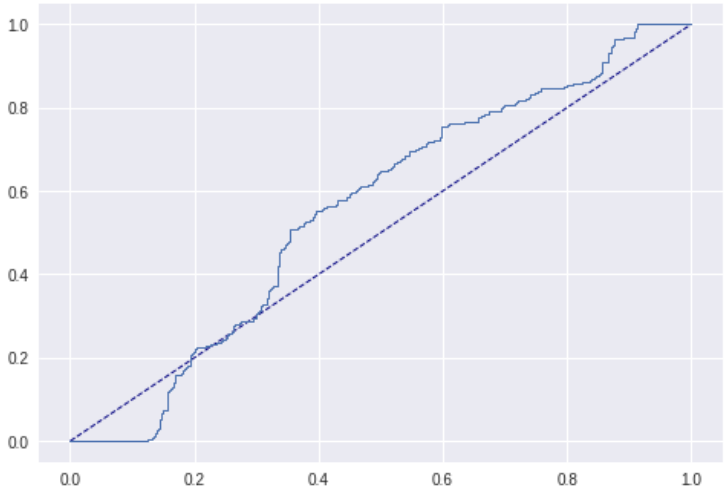
```
crossvalid: 0.640526, 0.697368, 0.698421, 0.666842
Test acc: 0.657000, 0.660000, 0.678000, 0.642000
```



- **100 windows, length 1000**

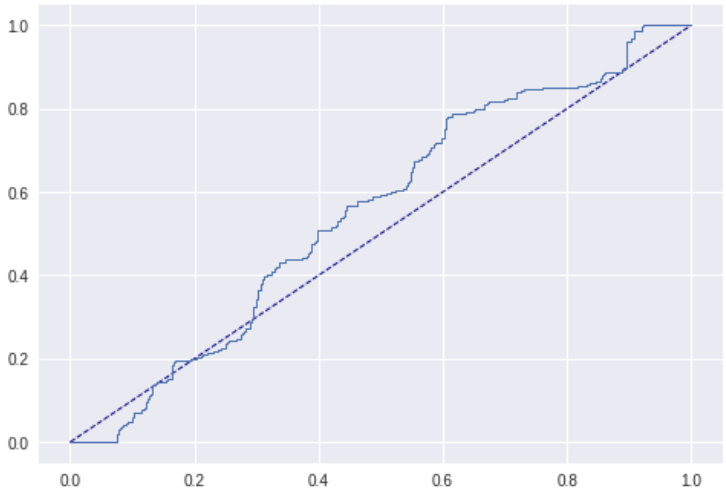
1. LinearSVC with L1 penalty

```
crossvalid: 0.692632, 0.614737, 0.565263, 0.507895
Test acc: 0.853000, 0.720000, 0.600000, 0.599000
```



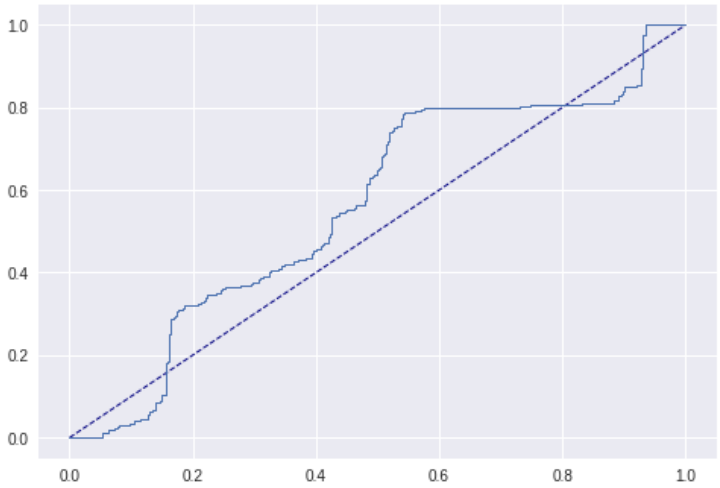
2. LinearSVC with L2 penalty

```
crossvalid: 0.711579, 0.608947, 0.577368, 0.589474
Test acc: 0.785000, 0.734000, 0.630000, 0.598000
```



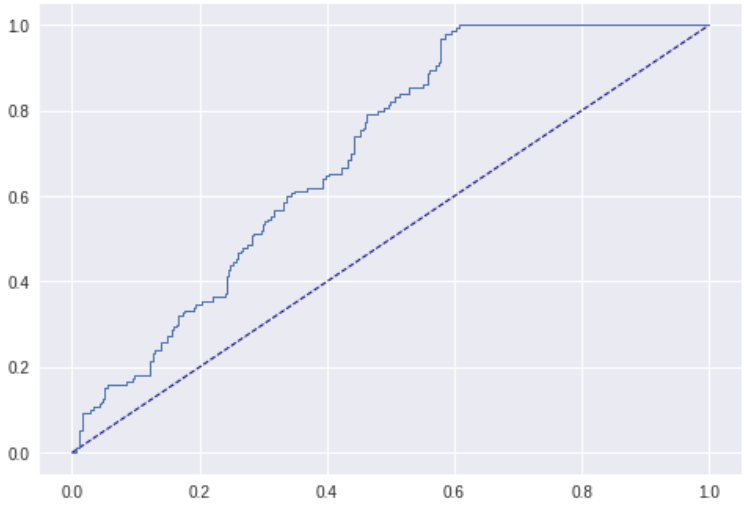
3. RBF kernel SVM

```
crossvalid: 0.707895, 0.576316, 0.551053, 0.636316
Test acc: 0.709000, 0.729000, 0.664000, 0.659000
```



4. Random Forest

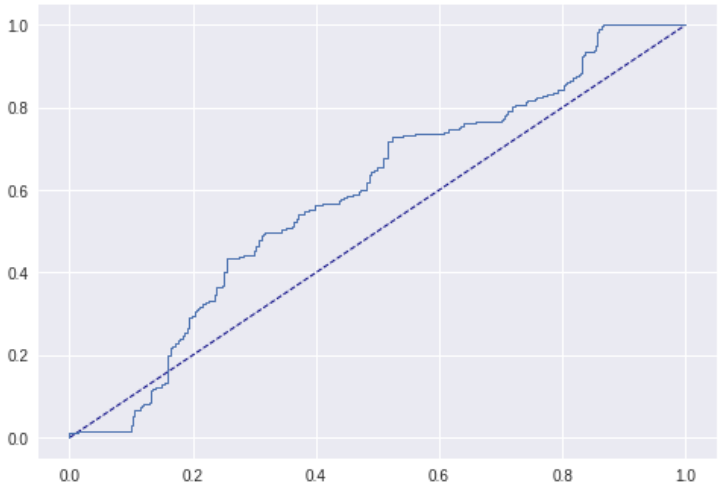
```
crossvalid: 0.721053, 0.639474, 0.633684, 0.645263
Test acc: 0.688000, 0.652000, 0.738000, 0.698000
```



- **100 windows, length 1500**

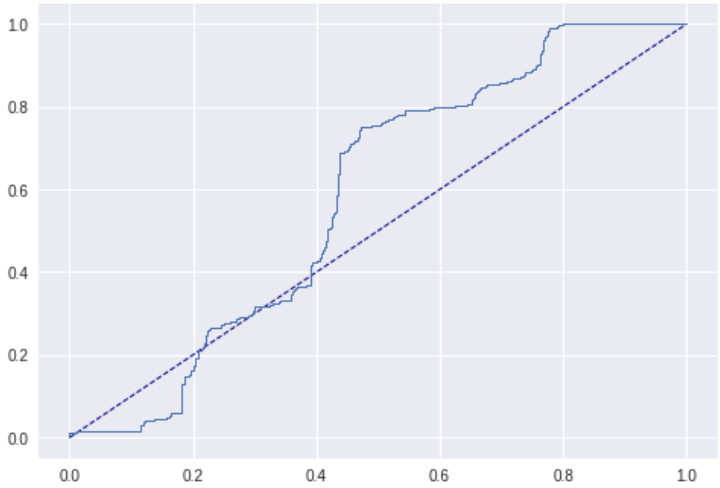
1. LinearSVC with L1 penalty

```
crossvalid: 0.571579, 0.554737, 0.522632, 0.650526
Test acc: 0.694000, 0.575000, 0.945000, 0.673000
```



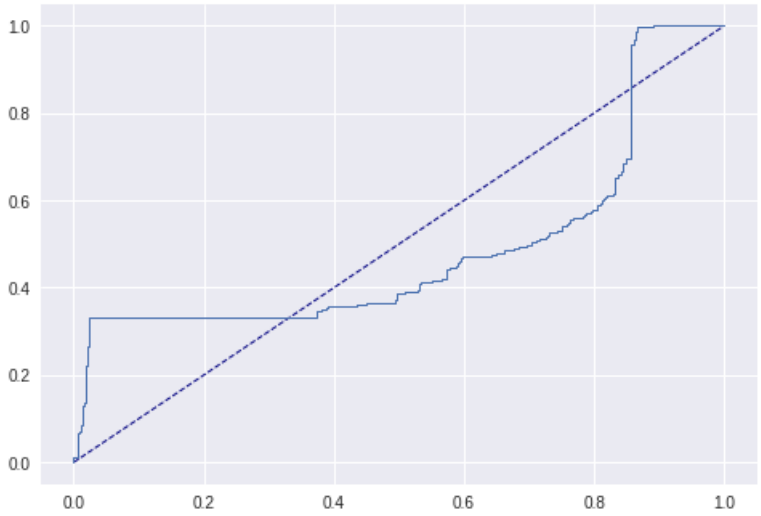
2. LinearSVC with L2 penalty

```
crossvalid: 0.625789, 0.534211, 0.547895, 0.588947
Test acc: 0.620000, 0.690000, 0.850000, 0.694000
```



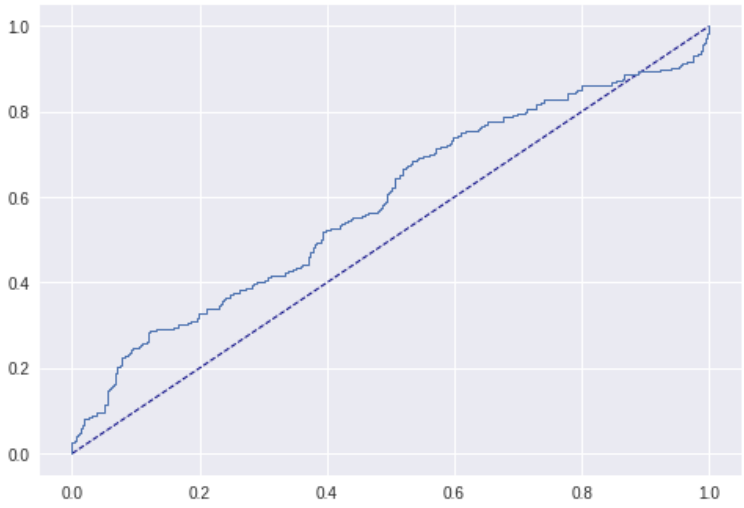
3. RBF kernel SVM

```
crossvalid: 0.610000, 0.654737, 0.576842, 0.594737
Test acc: 0.645000, 0.749000, 0.704000, 0.754000
```



4. Random Forest

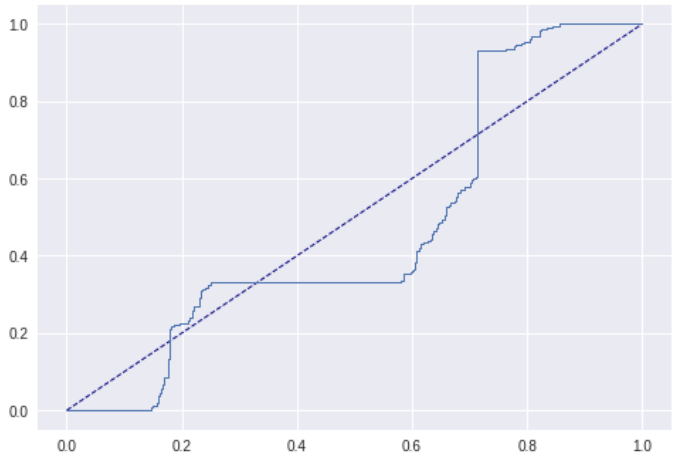
```
crossvalid: 0.617895, 0.796842, 0.656842, 0.553158
Test acc: 0.725000, 0.708000, 0.675000, 0.686000
```



- **100 windows, length 2000**

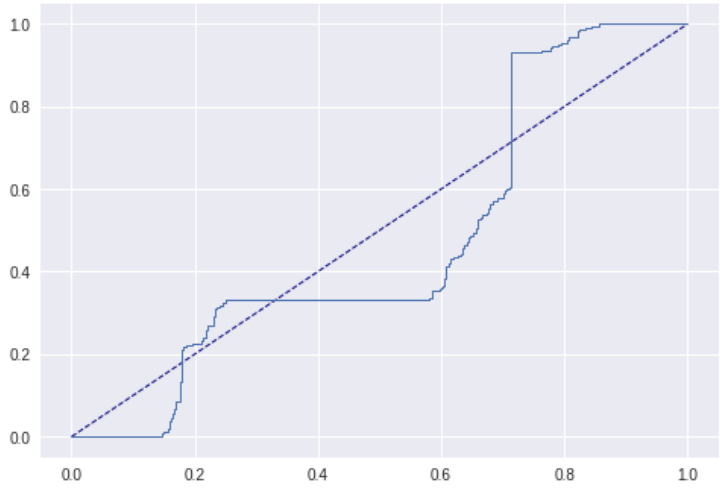
1. LinearSVC with L1 penalty

```
crossvalid: 0.739474, 0.594211, 0.565263, 0.405263
Test acc: 0.374000, 0.458000, 0.575000, 0.418000
```



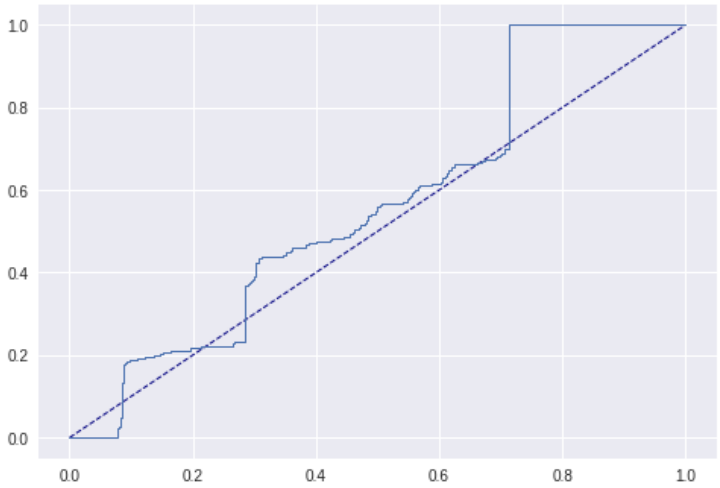
2. LinearSVC with L2 penalty

```
crossvalid: 0.737368, 0.607895, 0.533684, 0.400526
Test acc: 0.624000, 0.459000, 0.693000, 0.597000
```



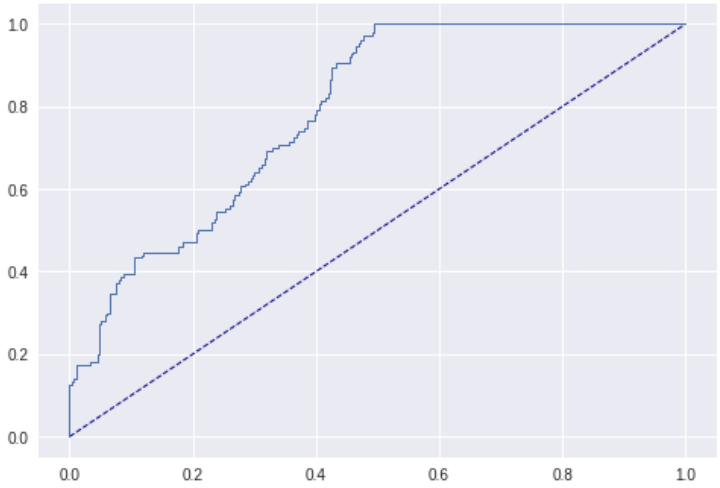
3. RBF kernel SVM

```
crossvalid: 0.737368, 0.607895, 0.533684, 0.400526
Test acc: 0.624000, 0.459000, 0.693000, 0.597000
```



4. Random Forest

```
crossvalid: 0.741579, 0.675263, 0.577368, 0.754211
Test acc: 0.741000, 0.734000, 0.700000, 0.750000
```



2.2 LSTM

For the **LSTM** model, I use dynamic correlation to process the data. That is, I sample using 50 or 100 windows for each subject sequentially at equal intervals with window length 300, 1500 or 2000. Then compute the correlation matrix for every window, so I get 50 or 100 correlation matrixes (200 X 200) for each subject. I use auto-encoder and vectorization for correlation matrixes to reduce the high data dimension.

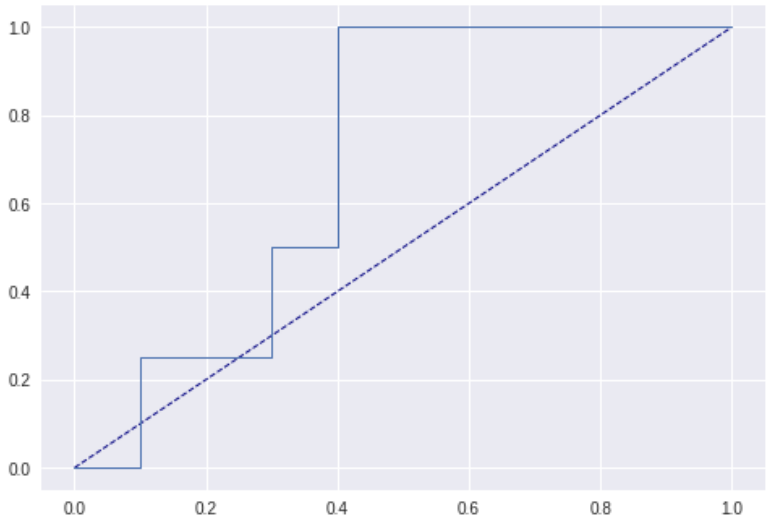
Data includes 25 impaired subjects with label 1 and 61 unimpaired subjects with label 0, and divided 86 subjects into three parts, 70 % training data, 10 % validation data and 20% test data. All these three parts data are independent and each are guaranteed for two kinds of labels. Similarly, 4-fold cross validation is used.

The model has three layers. Two for LSTM and one for dense layer. The activation function of dense layer is sigmoid. Loss function is 'binary_crossentropy' and optimizer is 'rmsprop'.

In the following, accuracy and ROC represent average accuracy on testing data and average ROC respectively.

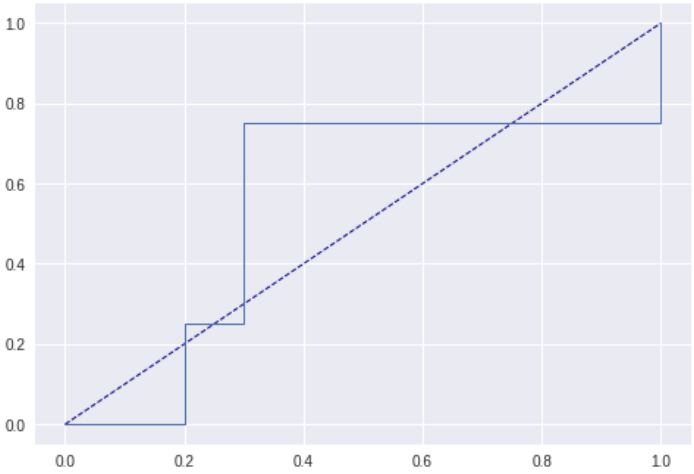
- **9 windows, length 300** ***(No overlap)***

  Accuracy: 65.3125%



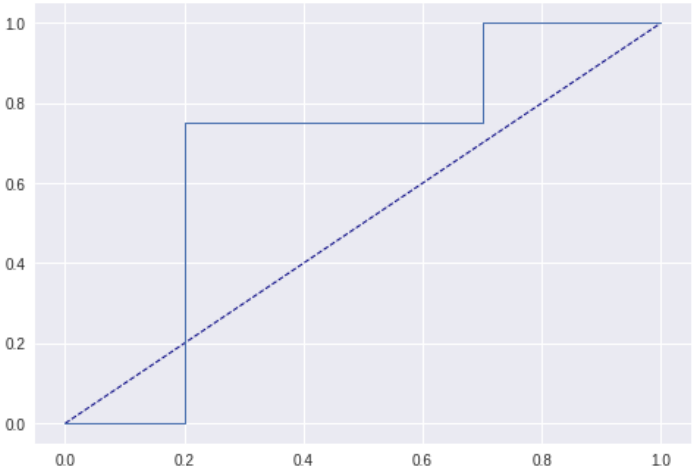
- **50 windows, length 1500**

  Accuracy: 61.2300%



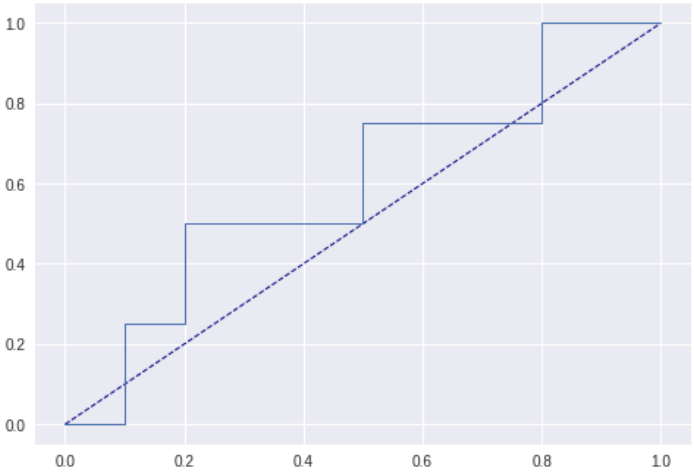
- **50 windows, length 2000**

  Accuracy: 66.7200%



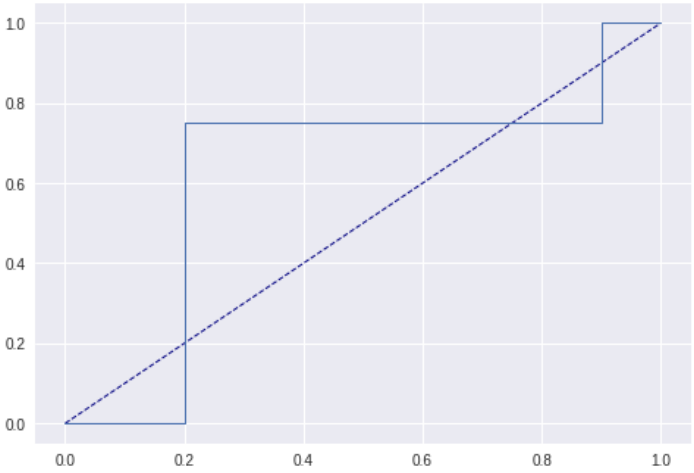
- **100 windows, length 1500**

  Accuracy: 57.9000%



- **100 windows, length 2000**

  Accuracy: 58.6500%



3. Conclusion

3.1. Conclusion in SVM and Random Forest

1. In section 2.1, we discussed about SVM and random forest. From the results, we can draw the conclusion that the random forest model has relatively stable and better results comparing to SVM models. Random forest models perform well especially in the case 100 X 1500 or 100 X 2000. Below is the accuracy.

```
4-fold Cross Validation: 0.741579, 0.675263, 0.577368, 0.754211
Test Accuracy: 0.741000, 0.734000, 0.700000, 0.750000
```

AUC: 0.712700

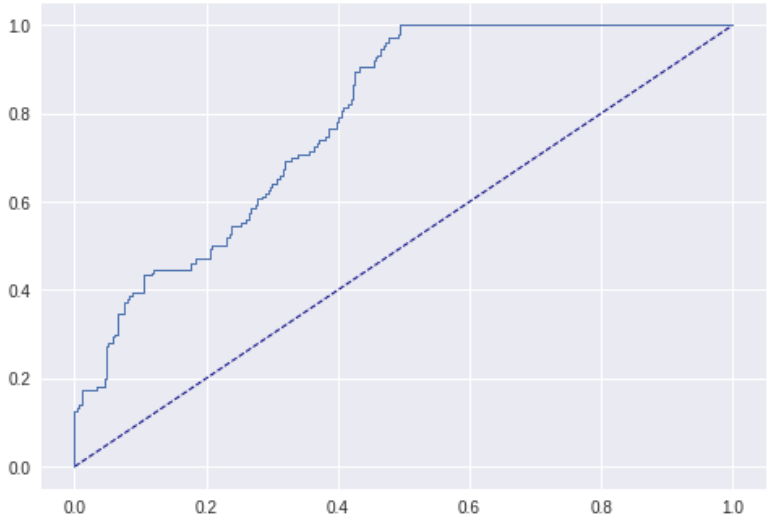


Figure. Random Forest Model Average ROC at 100 X 2000

2. In the setting 50 X 1000, all these models have relatively good performances.

3.2. Conclusion in LSTM

1. In training process, LSTM model cannot reach a small loss. Although the loss can decrease as the training progressed, the loss and accuracy on testing data are not improved.
2. If the overlap between two windows is big, the prediction probability result shows small differences. In 9 X 300 setting, the differences between predictions of different subjects are obvious. However, it does not show a good accuracy either. Maybe it is caused by small data size, or we can explore other settings for this model.

Appendix: Source Code

Please see other two Python files.

The original files are Jupyter Notebook format, so some code block need to be executed block by block.