

Brief summary

Brief summary

1. Label

2. Import Data

3. Data Preprocessing

1. For Multi-layer Neural Network

2. For other Classifiers

4. Training model

1. Multi-layer Neural Network

2. Other Classifiers

5. Conclusion

1. Label

- Manually tag each waveform
- To calculate Easily, label value 0-4 corresponds to 140-148

2. Import Data

- $X : 100(num) * 302(factor)$
- $Y : 100 * 1$
- We can choose to use PCA for the number of factor is large

```
1. dframe = pd.read_excel("test.xlsx")
2. test = DataFrame(dframe)
3. test=test.T
4. X = test.iloc[ : , :-1].values
5. #X = finalData # Using PCA compressed data, the test results are not
   ideal
6. Y = test.iloc[ : , 302].values
```

3. Data Preprocessing

1. For Multi-layer Neural Network

- First we need to split the dataset : Using Sklearn split automatically(or using "k-fold cross validation")
- Change y to One-hot vector like 0 → [1, 0, 0, 0, 0]; 1 → [0, 1, 0, 0, 0];

```
1.
2.     # Using sklearn to split dataset 1:4
3.     from sklearn.model_selection import train_test_split
4.     x_train, x_test, y_train, y_test = train_test_split(X, Y,
5.                                                         test_size=0.20, random_state=0)
6.
7.     # shape each y to [0,0,1,0,0]....
8.     y_train = np_utils.to_categorical(y_train)
9.     y_test = np_utils.to_categorical(y_test)
10.    num_classes = y_test.shape[1]
```

2. For other Classifiers

Random tree, random forest, SVM...

- Split the dataset,type meets requirements.
- Set Random_state as 0 ; it is like random seed which needs to set before optimizing

```
1.     from sklearn.model_selection import train_test_split
2.     x_train, x_test, y_train, y_test = train_test_split(X, Y,
3.                                                         test_size=0.20, random_state=0)
```

4. Training model

1. Multi-layer Neural Network

- Construct a three-layer neural network based on keras .
- Choosing 'relu' as the activation function
- Using Adam to optimize
- Adding validation data (20%) to reduce overfitting
- According to test, we choose epochs as 300

```

1.  # 3-layers
2.  model = Sequential()
3.  model.add(Dense(90, activation='relu', input_dim=302))
4.  model.add(Dense(30, activation='relu'))
5.  model.add(Dense(5, activation='softmax'))
6.
7.  # using Adam to optimize
8.  rms = keras.optimizers.Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
9.  model.compile(loss='binary_crossentropy',
10.               optimizer=rms,
11.               metrics=['accuracy'])
12.  # add validation data to reduce overfitting
13.  model.fit(x_train, y_train, validation_split=0.2,
14.           epochs=300,
15.           batch_size=10)
16.  score = model.evaluate(x_test, y_test, batch_size=10)
17.  print(model.metrics_names)
18.  print(score)

```

Results: Accuracy is 98.00%

We can continue to optimize and improve accuracy.

2. Other Classifiers

- Using tpot package to select models and optimize automatically

A Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming.

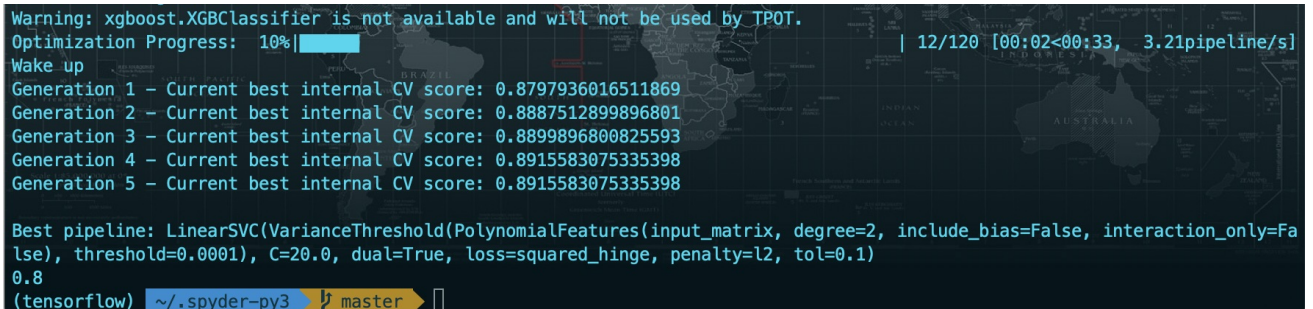
```

1.  tpot = TPOTClassifier(generations=5, population_size=20, verbosity=2)
2.  tpot.fit(X_train, y_train)

```

```
3. print(tpot.score(X_test, y_test))
4. tpot.export('tpot_pipeline.py')
```

- Results: Accuracy is 89.15% --- Linear SVC



Warning: xgboost.XGBClassifier is not available and will not be used by TPOT.
 Optimization Progress: 10% | 12/120 [00:02<00:33, 3.21pipeline/s]
 Wake up
 Generation 1 - Current best internal CV score: 0.8797936016511869
 Generation 2 - Current best internal CV score: 0.8887512899896801
 Generation 3 - Current best internal CV score: 0.8899896800825593
 Generation 4 - Current best internal CV score: 0.8915583075335398
 Generation 5 - Current best internal CV score: 0.8915583075335398
 Best pipeline: LinearSVC(VarianceThreshold(PolynomialFeatures(input_matrix, degree=2, include_bias=False, interaction_only=False), threshold=0.0001), C=20.0, dual=True, loss=squared_hinge, penalty=l2, tol=0.1)
 0.8
 (tensorflow) ~/..spyder-py3 master

- Once TPOT is finished searching , it provides us with the Python code for the best pipeline it found so we can tinker with the pipeline from there.
- According to this , using SVC(linear) + PCA

```
1. from sklearn.svm import SVC
2. clf = SVC(C=1, kernel='linear', probability = True, random_state=0)
3. clf.fit(x_train, y_train)
```

Results : Accuracy is 90%

Reason :

- The number of features is much larger than the number of samples
- The number of samples is small .

We can continue to optimize and improve accuracy.

5. Conclusion

- Better to Use Multi-layer Neural Network
- Better to have more data
- PCA is not useful for this data , maybe useful for a larger factors
- Maybe we can use DP Alg to compressed data (waveform)

2018-11-10