2nd mini project - classification

สมาชิกกลุ่ม

- 6310400941 นางสาวจิรัชญา ผ่านพินิจ
- 6310400967 นายณภัทร ดลภาวิจิต
- 6310401980 นายวงศกร เสนีวงศ์ ณ อยุธยา

ขอบเขตปัญหา

ใช้ข้อมูลอาการป่วยของวัว เพื่อจำแนกว่าวัวที่ป่วยจะรอดชีวิต เสียชีวิต หรือควรถูกฉีดยาให้ตาย

ชุดข้อมูล

ข้อมูลลักษณะของวัว เช่น เคยผ่าตัดหรือไม่ วัย ระดับความเจ็บปวด โปรตีนทั้งหมด ชีพจร อุณหภูมิ ชีพจรที่คลำ เป็นต้น มีทั้งสิ้นจำนวน 299 ตัวอย่าง ผู้ทดลองต้องการทำนายสถานะการมีชีวิตของวัวซึ่งตรงกับคอลลัมน์ “outcome Class” ในตารางชุดข้อมูล สามารถเข้าถึงได้จาก Cow

Pre-processing

- แยกคอลลัมน์ที่มีจำนวนของข้อมูลที่ไม่ซ้ำกันมีน้อยกว่า 10 ให้นับว่าเป็นข้อมูลจำแนกประเภท มิเช่นนั้นให้นับว่าเป็นข้อมูลเชิงตัวเลข
- ข้อมูลคอลลัมน์ “lesion” เป็นข้อมูลที่ประกอบด้วยหลายข้อมูลย่อยตามข้อมูลอภิพันธุ์ที่ให้มา เราจึงแยกข้อมูลออกด้วย Regular Expession ทำให้ได้ข้อมูลเพิ่มเป็น 'lesion site', 'lesion type', 'lesion subtype', และ 'lesion code'
- ข้อมูลว่าง: ในข้อมูลตัวเลขเราเลือกใช้ค่าเฉลี่ย และในข้อมูลจำแนกประเภทเรากำหนดให้เป็นประเภท “[NAN]”

Decision Tree Classification

การทดลอง

1. ทดลอง Feature selection
2. เปรียบเทียบประสิทธิภาพโมเดล decision Tree ด้วยชุดข้อมูลที่ sampling ด้วยเทคนิคที่แตกต่างกันเพื่อลดปัญหา imbalance
   2.1. ชุดข้อมูลที่ไม่มีการ sampling
   2.2. ชุดข้อมูลที่มีการ oversampling ด้วยเทคนิค random sampling
   2.3. ชุดข้อมูลที่มีการ oversampling ด้วยเทคนิค SMOTE (Synthetic Minority Over-sampling Technique)
   2.4. ชุดข้อมูลที่มีการ undersampling ด้วยเทคนิค TOMEK
   2.5. ชุดข้อมูลที่มีการ oversampling ด้วยเทคนิค ADASYN
3. เปรียบเทียบประสิทธิภาพโมเดล decision tree ที่ใช้เกณฑ์การคัดเลือกฟีเจอร์แบบ gini และ entropy โดยใช้ชุดข้อมูล sampling ที่ดีที่สุดของการทดลองก่อนหน้า
4. ทดลองเพื่อหาพารามิเตอร์ที่ดีที่สุดในการสร้างโมเดล decision tree ด้วย GridSearchCV

การทดลองที่ 1 : Feature selection

ทดลอง Feature selection โดยใช้ 2 เทคนิค

1. ใช้ LDA กับฟีเจอร์ที่เป็น Numerical
   1.1. คัดเลือกฟีเจอร์ที่เป็น Numerical

```
feature_num = ['temperature', 'pulse', 'respiratory_rate',
'packed_cell_volume', 'total_protein', 'abdomo_protein']
X_num_feature = X[feature_num].copy()
```

   1.2. ทำ Feature selection ด้วย LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA
lda = LDA(n_components=2)
X_num_select = lda.fit_transform(X_num_feature, y)
```

1.3. แสดงผลลัพธ์

```
print('Original numerical feature number:', X_num_feature.shape[1])
print('Reduced numerical feature number:', X_num_select.shape[1])
>>> Original numerical feature number: 6
>>> Reduced numerical feature number: 2
```

2. ใช้ Chi squareกับฟีเจอร์ที่เป็น Nominal

2.1. คัดเลือกฟีเจอร์ที่เป็น Nominal

```
feature_nom = [i for i in np.array(X.columns) if i not in
np.array(feature_num)]
X_nom_feature = X[feature_nom].copy()
```

2.2. ทำ Feature selection ด้วย Chi square

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
chi2_features = SelectKBest(chi2, k=15)
X_nom_select = chi2_features.fit_transform(X_nom_feature, y)
```

2.3. แสดงผลลัพธ์

```
print('Original nominal feature number:', X_nom_feature.shape[1])
print('Reduced nominal feature number:',X_nom_select.shape[1])
>>> Original nominal feature number: 20
>>> Reduced nominal feature number: 15
```

จากการทำ feature selection ในส่วนของ chai square พบว่า feature ที่ไม่ได้ถูกเลือกคือ peripheral_pulse, mucous_membrane, nasogastric_tube, nasogastric_reflux และ nasogastric_reflux_ph

รวม Numerical feature และ nominal feature ที่ทำ feature selection แล้ว

```
X_select = np.concatenate([X_num_select, X_nom_select], axis=1)
```

การทดลองที่ 2

เปรียบเทียบประสิทธิภาพโมเดล Decision Tree ด้วยชุดข้อมูลที่ Sampling ด้วยเทคนิคต่างๆ

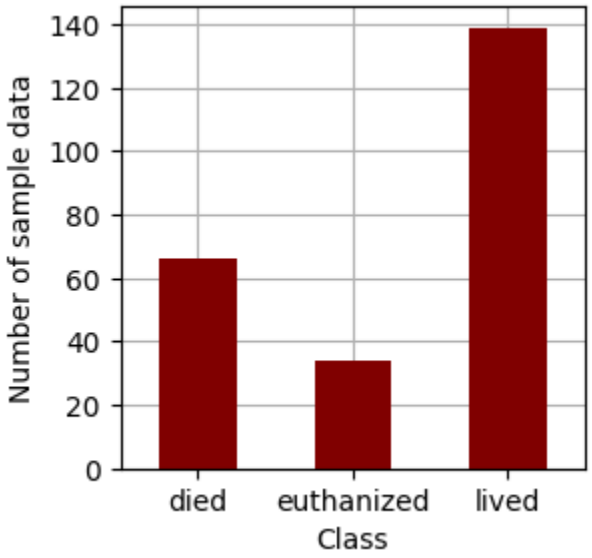
- แยกข้อมูลฝึก/ทดสอบ

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_select, y,
test_size=0.2)
```

- สร้างฟังก์ชั้นสำหรับสร้างโมเดล โดยกำหนดให้พารามิเตอร์เหมือนกันทังหมดสำหรับการทดลองที่ 2

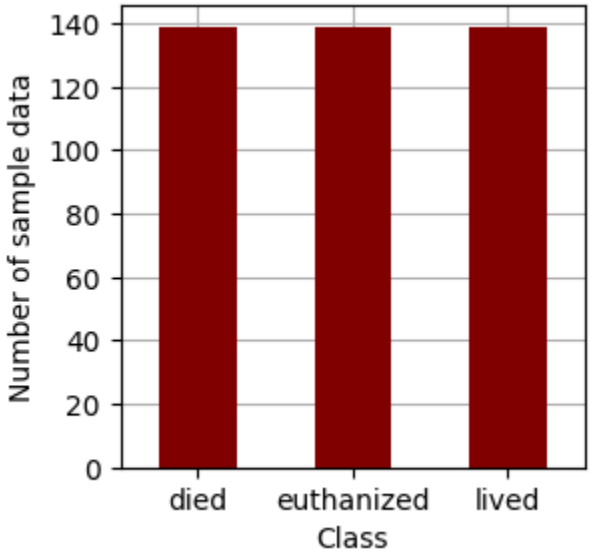
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import metrics

def bulidAndTrainModel(X_train, y_train) :
    dtree = DecisionTreeClassifier(criterion='entropy',
max_depth=30)
    clf = dtree.fit(X_train, y_train)
    pred = clf.predict(X=X_test)
    result['model'].append(clf)
    result['accuracy'].append(metrics.accuracy_score(y_test, pred))

    cm = metrics.confusion_matrix(y_test, pred)
    recall_class_died = cm[0][0]/sum(cm[:,0])
    result['recall'].append(recall_class_died)

    result['F1'].append(metrics.f1_score(y_test, pred,
average='weighted'))

    precision_class_died = cm[0][0]/sum(cm[0,:])
    result['precision'].append(precision_class_died)

    return clf, pred
```

1. ชุดข้อมูลฝึกที่ยังไม่ได้ sampling



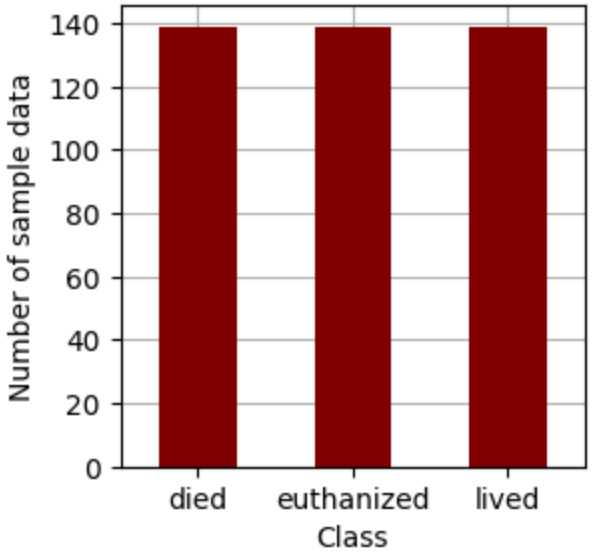
2. ชุดข้อมูลฝึกที่มีการ oversampling ด้วยเทคนิค random sampling

```
from imblearn.over_sampling import RandomOverSampler
res = RandomOverSampler(random_state=42)
X_res_random,y_res_random = res.fit_resample(X_train,y_train)
```



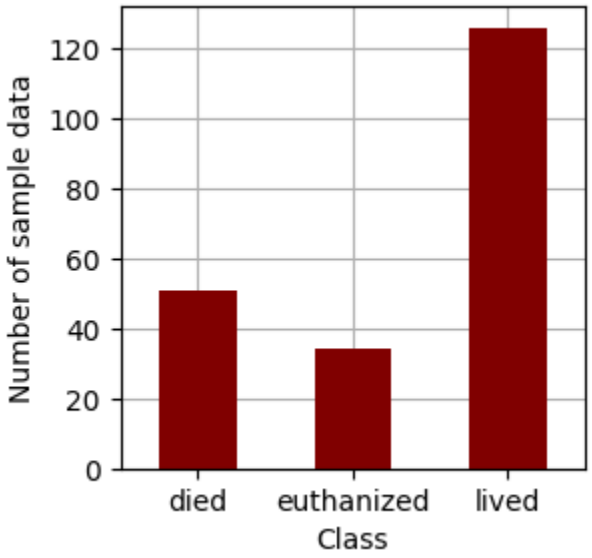
3. ชุดข้อมูลฝึกที่มีการ oversampling ด้วยเทคนิค SMOTE

```
from imblearn .over_sampling import SMOTE
sm = SMOTE()
X_resampled_smote, y_resampled_smote =
sm.fit_resample(X_train,y_train)
```



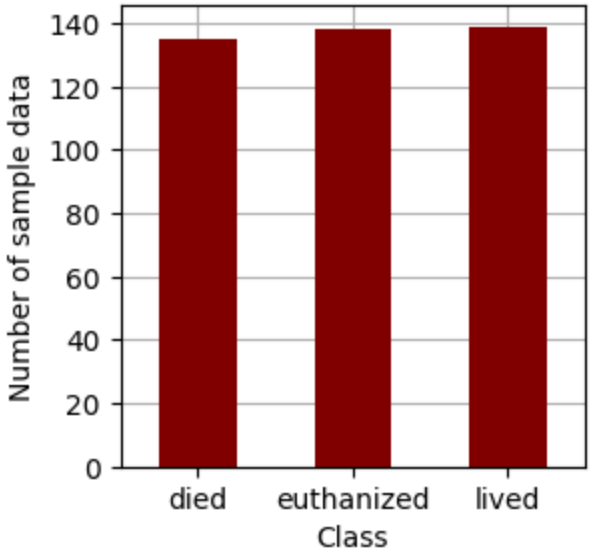
4. ชุดข้อมูลฝึกที่มีการ oversampling ด้วยเทคนิค TOMEK

```
from collections import Counter
from imblearn.under_sampling import TomekLinks
tl = TomekLinks()
X_res_tomek, y_res_tomek = tl.fit_resample(X_train, y_train)
```

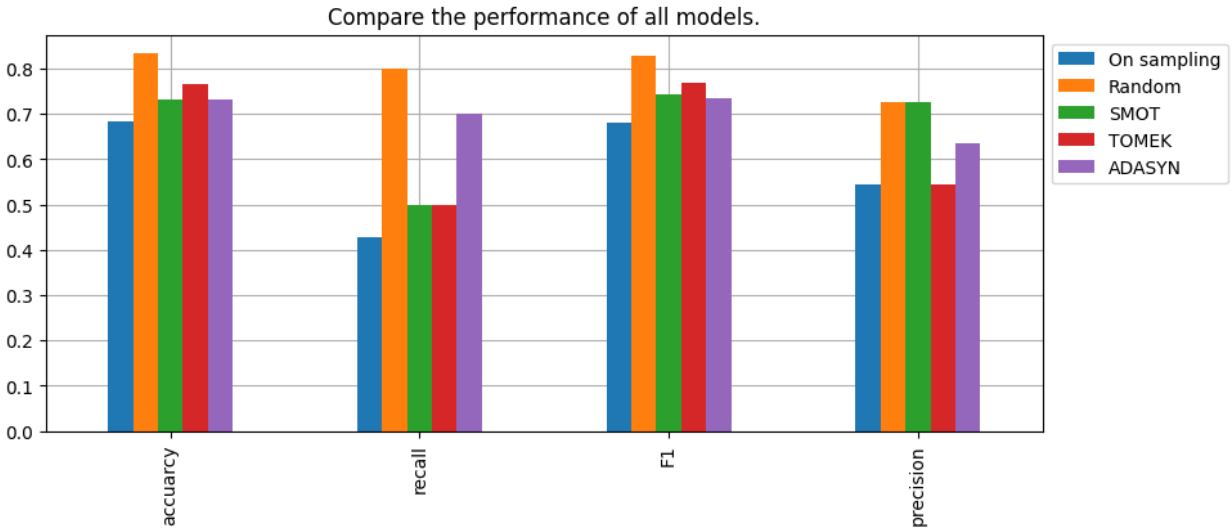


5. ชุดข้อมูลฝึกที่มีการ oversampling ด้วยเทคนิค ADASYN

```
from imblearn.over_sampling import ADASYN
ad = ADASYN()
X_res_adasyn, y_res_adasyn = ad.fit_resample(X_train, y_train)
```



6. แสดงกราฟเปรียบเทียบประสิทธิภาพโมเดลที่เรียนรู้กับชุดข้อมูลที่ sampling แตกต่างกัน ในการวัดประสิทธิภาพ ผู้ทดลองได้กำหนดให้การวัดประสิทธิภาพแบบ recall และ precision เป็นการวัดแบบที่สนใจ class died



อภิปรายและสรุปผลการทดลอง

จากกราฟผลการทดลองเปรียบเทียบประสิทธิภาพโมเดล decision tree ที่เรียนรู้ชุดข้อมูลที่ sampling แตกต่างจะพบบว่าการ sampling ด้วยเทคนิค random โมเดลที่เรียนรู้ชุดข้อมูลนี้มีประสิทธิภาพมากที่สุด ซึ่งหมายความว่าการ sampling ด้วยเทคนิค random ลดปัญหาข้อมูลไม่เท่ากันได้ดีกว่าเทคนิคอื่นๆ สำหรับชุดข้อมูลนี้

การทดลองที่ 3

เปรียบเทียบประสิทธิภาพโมเดล decision tree ที่ใช้เกณฑ์การคัดเลือกฟีเจอร์แบบ gini และ entropy โดยใช้ชุดข้อมูล sampling ที่ดีที่สุดของการทดลองที่ 2 โดยพิจารณาจากค่าเฉลี่ยของ recall score, precision score ที่สูงที่สุด

```
sampling_name = index[(data_scores.recall.values +
data_scores.precision.values).argmax(axis=0)]
print('Use sampling : ' + sampling_name)
```

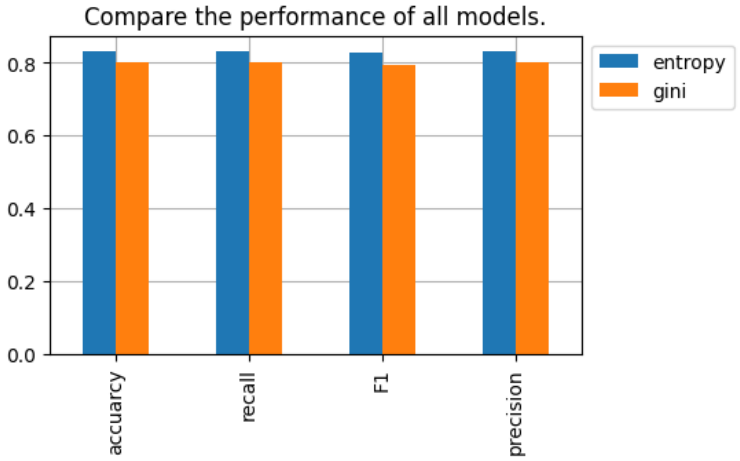
1. เกณฑ์การคัดเลือกฟีเจอร์แบบ entropy

	precision	recall	f1-score	support
died	0.73	0.73	0.73	11
euthanized	0.86	0.60	0.71	10
lived	0.86	0.92	0.89	39
accuracy			0.83	60
macro avg	0.81	0.75	0.77	60
weighted avg	0.83	0.83	0.83	60

2. เกณฑ์การคัดเลือกฟีเจอร์แบบ gini

	precision	recall	f1-score	support
died	0.67	0.73	0.70	11
euthanized	0.83	0.50	0.62	10
lived	0.83	0.90	0.86	39
accuracy			0.80	60
macro avg	0.78	0.71	0.73	60
weighted avg	0.80	0.80	0.79	60

3. เปรียบเทียบเกณฑ์การคัดเลือกฟีเจอร์ทั้ง 2 แบบ



อภิปรายและสรุปผลการทดลอง

จากกราฟผลการทดลองเปรียบเทียบประสิทธิภาพโมเดล decision tree ที่ใช้เกณฑ์การคัดเลือกฟีเจอร์แบบ entropy และ gini พบว่าเกณฑ์การคัดเลือกฟีเจอร์แบบ entropy มีประสิทธิภาพสูงกว่าแบบ gini

การทดลองที่ 4

ทดลองเพื่อหาพารามิเตอร์ที่ดีที่สุดในการสร้างโมเดล decision tree ด้วย GirdSearchCV และใช้ชุดข้อมูล sampling ที่ดีที่สุดของการทดลองที่ 2 โดยพิจารณาจากค่าเฉลี่ยของ recall score, precision score ที่สูงที่สุด

1. กำหนดพารามิเตอร์ที่ต้องการทดลองดังนี้

```
param_grid = { 'criterion':['gini','entropy'], 'max_depth':
np.arange(3, 100)}
cv = 10
```

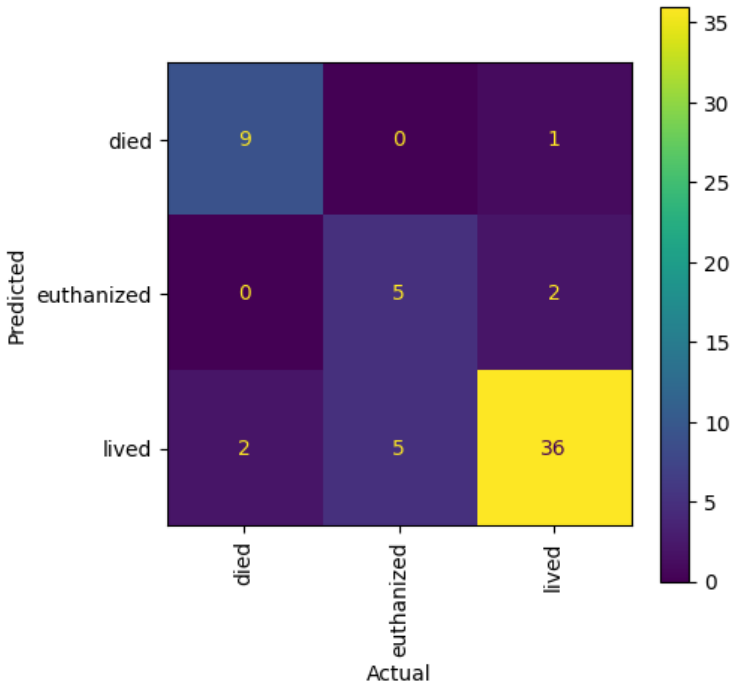
2. สร้างฟังก์ชันสร้างโมเดลและเรียนรู้ชุดข้อมูลฝึก

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

def gridSearchCVbulidAndTrainModel(X_train, y_train) :
    dtree = DecisionTreeClassifier()
    dtree_gscv = GridSearchCV(dtree, param_grid, cv=cv,
scoring='accuracy')
    dtree_gscv.fit(X_train, y_train)
    print('best params : {}'.format(dtree_gscv.best_params_))
    print('best score : {}'.format(dtree_gscv.best_score_))

    return dtree_gscv.best_estimator_
```

```
dtree_cv =
gridSearchCVbulidAndTrainModel(sampling[sampling_name][0],
sampling[sampling_name][1])
>>> best params : {'criterion': 'gini', 'max_depth': 31}
>>> best score : 0.9069686411149824
```

3. แสดงผลการทดลอง

	precision	recall	f1-score	support
died	0.90	0.82	0.86	11
euthanized	0.71	0.50	0.59	10
lived	0.84	0.92	0.88	39
accuracy			0.83	60
macro avg	0.82	0.75	0.77	60
weighted avg	0.83	0.83	0.83	60



สรุปอภิปรายผลการทดลอง

จากผลการทดลอง หาพารามิเตอร์ที่ดีที่สุดในการสร้างโมเดล decision tree ด้วย GridSearchCV ได้ค่าพารามิเตอร์ที่ดีที่สุดดังนี้ เกณฑ์การเลือกฟีเจอร์แบบ gini และ max depth เท่ากับ 31 ซึ่งโมเดลที่ได้มีคะแนน recall และ precision ใน class died เป็น 0.82 และ 0.9 ตามลำดับ นอกจากนี้ยังมีคะแนน accuracy เป็น 0.83 หมายความว่าโมเดลที่กำหนดพารามิเตอร์ตามที่ได้กล่าวข้างต้นมีการเรียนรู้และทำนายค่าได้ดี

Deep Neural Networks (DNN)

Pre-processing

ในการใช้ DNN จำเป็นต้องทำการเตรียมข้อมูลเพิ่มเติมโดยการเข้ารหัสด้วย 1 เพื่อใช้หาค่า loss ในชั้นสุดท้ายของโมเดลได้

สร้างโมเดล

อินพุตเป็นข้อมูลนำมาใช้จะมีสองประเภทคือข้อมูลเชิงตัวเลข และข้อมูลจำแนกประเภท โดยที่ข้อมูลจำแนกประเภทจะนำไป embed ขนาด 128 หน่วยก่อนเพื่อให้มีมิติข้อมูลมากขึ้น จากนั้นก็นำข้อมูลจากทุกอินพุตมาต่อกัน (concatenate) ตามมาด้วยชั้น Dense และจากนั้นไปหาค่าเฉลี่ยระหว่างอินพุตเดิมกับข้อมูลชั้นปัจุบัน และต่อด้วย 2 ชั้น Dense โดยที่ชั้นสุดท้ายมีจำนวนโหนดเป็น 3 หน่วยตรงกับจำนวนประเภทของวัวคือ มีชีวิต ตาย และฉีดยาให้ตาย

```
def _build_input_layer(feature_name):
   if col2types[feature_name] == 'category':
       inputs = tf.keras.layers.Input(shape=(1,), name=feature_name)
       n = np.unique(X[:, x_label2idx[feature_name]]).size
       x = tf.keras.layers.Embedding(n, input_size, input_length=1)(inputs)
       x = tf.keras.layers.Flatten()(x)
       x = tf.keras.layers.Dense(input_size)(inputs)
       return inputs, x
   else:
       inputs = tf.keras.layers.Input(shape=(1,), name=feature_name)
       x = tf.keras.layers.Dense(input_size)(inputs)
       return inputs, x
```

โค้ดแสดงฟังก์ชั่นการสร้างชั้นอินพุตของโมเดล

หากว่าฟีเจอร์เป็นข้อมูลจำแนกประเภท ให้ผ่านชั้น Embed ก่อน

การทดลอง

ในการทดลองใช้โมเดลแบบ DNN กำหนดให้โมเดลมีโครงสร้างแบบเดียวกัน มีจำนวนรอบ (epoch) เท่ากัน ใช้ optimizer เหมือนกัน แต่กำหนดให้จำนวน features ที่ใช้ต่างกัน โดยมีการทดลองทั้งสิ้น 3 การทดลองคือ

1. ใช้ features ทั้งหมด (ทั้งสิ้น 26 features)
2. คัดเลือก features ที่ 50 เปอร์เซ็นไทล์ (ทั้งสิ้น 13 features)

   ตัวอย่างโค้ด

   ```
   import sklearn.feature_selection as fs
   fs.SelectPercentile(fs.f_classif, percentile=50)
   ```
3. คัดเลือก features ที่ 10 เปอร์เซ็นไทล์ (ทั้งสิ้น 3 features)

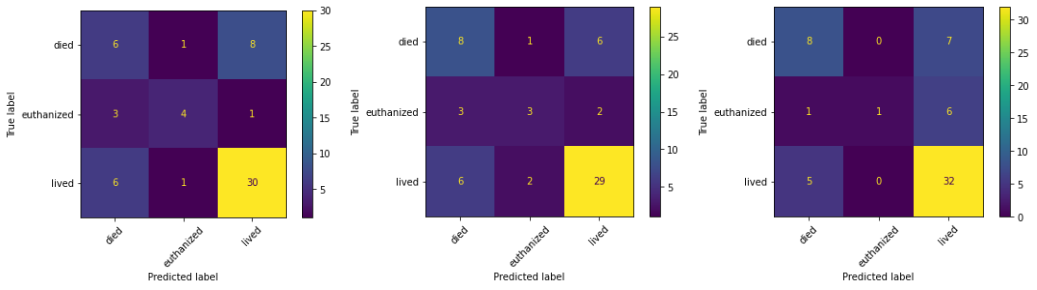
   ตัวอย่างโค้ด

   ```
   import sklearn.feature_selection as fs
   fs.SelectPercentile(fs.f_classif, percentile=10)
   ```

ผลการทดลอง

การทดลอง	Precision			Recall			จำนวนพารามิเตอร์
	มีชีวิต	ตาย	ฉีดยาตาย	มีชีวิต	ตาย	ฉีดยาตาย	
ใช้ features ทั้งหมด	0.78	0.5	**1.00**	**0.81**	0.60	0.25	441,219
50 เปอร์เซ็นไทล์	**0.82**	0.45	**1.00**	0.76	**0.67**	**0.50**	224,899
10 เปอร์เซ็นไทล์	0.76	**0.57**	0.75	0.76	0.60	0.38	75,139

ตารางแสดงคะแนน Precision และ Recall ของแต่ละการทดลอง



กราฟเมทริกซ์ความสับสนของโมเดลที่ใช้ features ทั้งหมด ใช้ features ที่ 50 เปอร์เซ็นไทล์ และใช้ 10 เปอร์เซ็นไทล์ ตามลำดับจากซ้ายไปขวา

จากการทดลองพบว่าการเลือกใช้ features เยอะไม่ได้หมายความว่าจะดีกว่าการใช้ feature น้อยเสมอไป ดังการทดลองที่ใช้ features ที่ 50 เปอร์เซ็นไทล์ได้ผลลัพท์การทำนายที่ดีที่สุดในหลายมาตราวัด นอกจากนี้ผลการยังบอกได้อีกว่าการใช้ features ทั้งหมดยังให้ผลลัพธิ์ที่ใกล้เคียงกับการใช้ feature ที่ 10 เปอร์เซ็นไทล์หรือ 3 features ตามการทดลองนี้

Support Vector Machine (SVM)

การทดลอง

ในการทดลองผู้ทดลองหา hyperparameter ที่เหมาะสมที่สุด โดยกำหนด hyperparameter ที่ต้องการหามีดังนี้คือ

1. **Loss**: ฟังก์ชั่นหาค่า loss โดยมี Hinge ที่เป็นค่าตั้งต้นและ Squared Hinge ที่จะทำการยกกำลังสอง Hinge
2. **Multi class**: เป็นกลวิธีการคำนวณเมื่อมีประเภทของการทำนายมากกว่าสอง ในพารามิเตอร์นี้มีให้เลือก OVR คือ หนึ่งเทียบกับทั้งหมด และ Crammer singer
3. **C**: เป็นค่าการควบควม โดยที่จะควบคุมผกผันกับค่า C อย่างเป็นสัดเป็นส่วน เราเลือกให้มี 1 และ 2

Hyperparameters			**precision**			**recall**		
loss	**Multi class**	**C**	**died**	**euthanized**	**lived**	**died**	**euthanized**	**lived**
Hinge	OVR	1	0.00	0.00	0.00	0.00	0.55	**1.00**
Hinge	OVR	2	0.62	**0.83**	0.75	0.33	**0.84**	0.82
Hinge	Crammer singer	1	0.60	0.50	0.43	**0.67**	**0.84**	0.79
Hinge	Crammer singer	2	0.65	**0.83**	**1.00**	0.11	0.78	0.85

Squared hinge	OVR	1	0.69	0.61	0.00	0.00	0.68	0.91
Squared hinge	OVR	2	0.00	0.00	0.00	0.00	0.55	**1.00**
Squared hinge	Crammer singer	1	0.00	0.00	0.00	0.00	0.55	**1.00**
Squared hinge	Crammer singer	2	**0.89**	0.44	0.57	0.44	0.75	**1.00**

ตารางแสดงคะแนน Precision และ Recall ของแต่ละการทดลอง

สรุปผลการทดลอง

จากผลการทดลอง พบว่า เมื่อวัดประสิทธิภาพด้วย precision hyperparameter loss=Hinge Multiclass=Crammersinger C=2 สามารถให้ประสิทธิภาพได้ที่สุด และเมื่อวัดประสิทธิภาพด้วย recall score พบว่า hyperparameter loss=Hinge Multiclass=Crammersinger C=1 สามารถให้ประสิทธิภาพได้ที่สุด