Needell_Coen_HW4

February 27, 2020

1 Conceptual

1.1 1

> Consider the Gini index, classification error, and cross-entropy in simple classification settings with two classes. Of these three possible cost functions, which would be best to use when growing a decision tree? Which would be best to use when pruning a decision tree? Why?

These loss functions technically optimize for different things. Classification error optimizes for the best accuracy for each region. Gini index optimizes for the variance across all classes, and categorical cross-entropy optimizes for certainty. The Gini index is the probability of the correct item being chosen multiplied by the probability of a mistake, summed over each item.

The way I see it, when we're growing the decision tree, we want to optimize for *accuracy*, but when we're pruning the tree, we want to optimize for *parsimony*. The classification error, while naïve, is the most direct measurement of accuracy, so that should be used for growing the tree, then while pruning we can use either the Gini index or the categorical cross-entropy. Both of those losses optimize for the *certainty* of each leaf, so they can be used to combine leaves which make the model more complicated than it needs to be. In particular, I would lean towards using categorical crossentropy, because it is proportional to $-p_{mk} \log p_{mk}$, as opposed to the Gini index, which is proportional to $1 - p_{mk}^2$. This subtle difference means that the categorical crossentropy will result in a final model which is less likely to give a wrong answer by chance, as opposed to by design.

2 Application

2.1 Estimating Models

2.1.1 2

[89]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import ElasticNet, LogisticRegression
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, validation_curve
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
from tqdm import tqdm
from palettable.cartocolors.qualitative import Antique_7
```

```
[62]: train = pd.read_csv('./data/gss_train.csv')
test = pd.read_csv('./data/gss_test.csv')
x_train = train.drop('colrac', axis=1)
y_train = train.colrac
x_test = test.drop('colrac', axis=1)
y_test = test.colrac
```

```
[6]: model_set = [
    (LogisticRegression(), {'C': np.logspace(-4, 4, 10)}),
    (GaussianNB(), {}),
    (ElasticNet(), {'alpha': np.logspace(-4, 4, 10), 'l1_ratio': [.1, .5, .7, .
↪9, .95, .99, 1]}),
    (DecisionTreeClassifier(), {'criterion': ['gini', 'entropy'], 'max_depth':␣
↪range(4, 20, 2)}),
    (BaggingClassifier(), {'n_estimators': range(10, 20)}),
    (RandomForestClassifier(), {'n_estimators': range(100, 500, 25),␣
↪'criterion': ['gini', 'entropy']}),
    (GradientBoostingClassifier(), {'learning_rate': np.logspace(-4, 0, 10),
                                    'loss': ['deviance', 'exponential'],␣
↪'n_estimators': range(100, 500, 25)})
]

best_models = {}
scores = {}

for model, parameters in tqdm(model_set):
    gscv = GridSearchCV(model, parameters, cv=10, refit=True, n_jobs=-1)
    gscv.fit(x_train, y_train)
    scores[model.__class__.__name__] = gscv.best_score_
    best_models[model.__class__.__name__] = gscv.best_estimator_
```

```
100%|      | 7/7 [09:16<00:00, 79.44s/it]
```

2.1.2 3

```
[20]: scores['ElasticNet'] = accuracy_score(y_train, np.
↪greater_equal(best_models['ElasticNet'].predict(x_train), 0.5))
scores
```

```
[20]: {'LogisticRegression': 0.7988049273763559,
       'GaussianNB': 0.7344226879941166,
       'ElasticNet': 0.8136856368563685,
       'DecisionTreeClassifier': 0.7797710976282405,
       'BaggingClassifier': 0.7818257032542746,
       'RandomForestClassifier': 0.8082643868358155,
       'GradientBoostingClassifier': 0.8048676227247655}
```

The scores printed above are accuracy for all models. Sklearn's gridsearchcv implements the built in .score() method for the model, and all of these use direct accuracy, or MSE in the case of ElasticNet. Because of this, we need to do the first line in the above block to binarize ElasticNet's linear function.

ElasticNet performs the best on this measure, which places the cutoff at 0.5. Let's try optimizing the error rate's and comparing again.

```
[95]: err_optis = {}
p_tars = np.linspace(0, 1, 1000)
for mname, model in best_models.items():
    if mname == 'ElasticNet':
        y_pred = model.predict(x_train)
    else:
        y_pred = model.predict_proba(x_train)[:, 1]
    accs = []
    for p in p_tars:
        acc = accuracy_score(y_train, np.greater_equal(y_pred, p))
        accs.append(acc)
    bestrun = np.argmax(accs)
    err_optis[mname] = (accs[bestrun], p_tars[bestrun])

err_optis
```

```
[95]: {'LogisticRegression': (0.825880758807588, 0.4094094094094094),
       'GaussianNB': (0.7506775067750677, 0.052052052052052055),
       'ElasticNet': (0.8163956639566395, 0.4854854854854855),
       'DecisionTreeClassifier': (0.7981029810298103, 0.33433433433433435),
       'BaggingClassifier': (0.997289972899729, 0.47147147147147145),
       'RandomForestClassifier': (1.0, 0.36336336336336333),
       'GradientBoostingClassifier': (0.8726287262872628, 0.5495495495495496)}
```

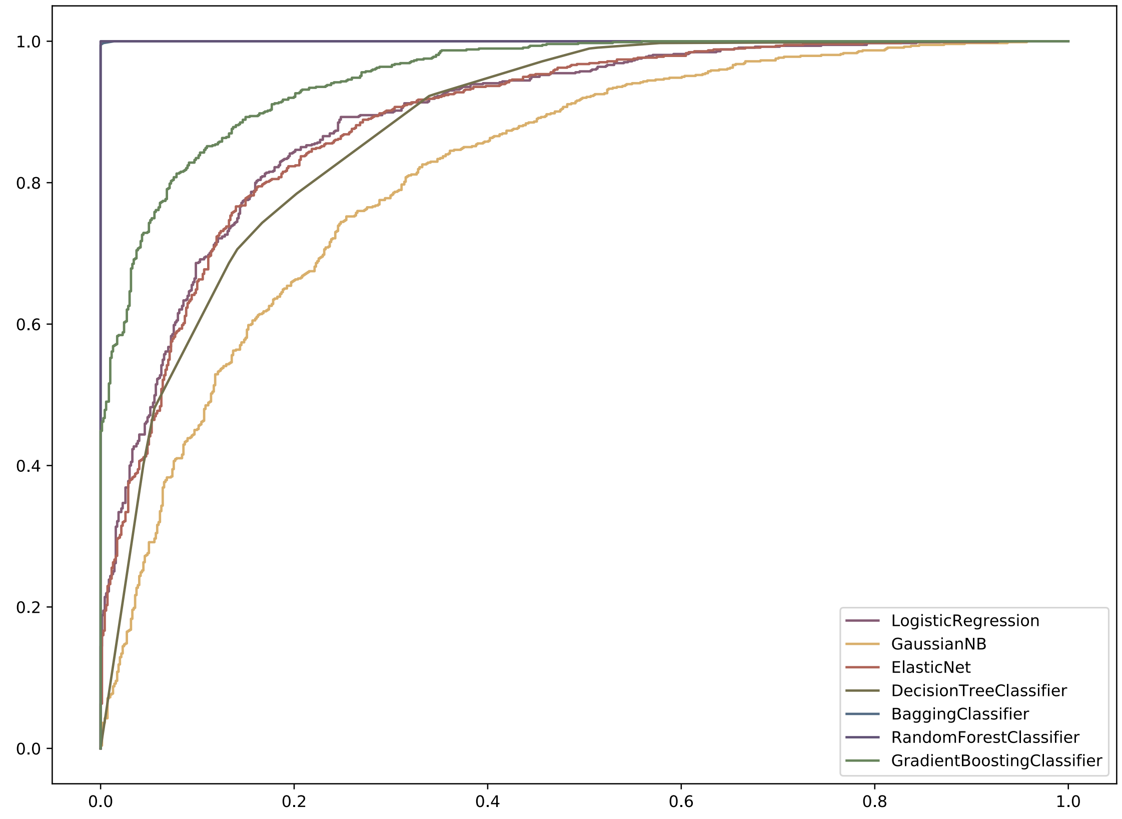
If we optimize the cutoff, random forest classifier performs perfectly, with the cutoff set to .36. The bagging classifier get's quite close too. Chances are good this is overfitting though, which we can see from the roc/aucs below

```
[97]: aucs = {}
sns.set_palette(Antique_7.mpl_colors)

plt.figure(figsize=(12, 9))
```

```
for mname, model in best_models.items():
    if mname == 'ElasticNet':
        y_pred = model.predict(x_train) # Recall elasticnet isn't a normally a␣
 ↪binary classification algorithm.
    else:
        y_pred = model.predict_proba(x_train)[:, 1]
    aucs[mname] = roc_auc_score(y_train, y_pred)
    fpr, tpr, thresholds = roc_curve(y_train, y_pred)
    plt.plot(fpr, tpr, label=mname)
    plt.legend()
print(aucs)
```

```
{'LogisticRegression': 0.893455432331692, 'GaussianNB': 0.816411577930146,
'ElasticNet': 0.889766692743086, 'DecisionTreeClassifier': 0.8799825134600341,
'BaggingClassifier': 0.9999742303621554, 'RandomForestClassifier': 1.0,
'GradientBoostingClassifier': 0.9505296580921265}
```



By the metrics above, one would be tempted to pick Random Forest as the best model. But don't be fooled, this is probably because of overfitting. Considering this, I would choose gradient boosting, it yielded the best non-perfect roc, and the best non-perfect accuracy. Considering this, we'll look at both random forest and gradient boosting in problem 5, to make this clearer. ### 5

[98]:
```python
# Random Forest:
rfc = best_models['RandomForestClassifier']
y_pred = rfc.predict_proba(x_test)[:, 1]
cutoff = err_optis['RandomForestClassifier'][0]
rfcacc = accuracy_score(y_test, y_pred > cutoff)
rfcauc = roc_auc_score(y_test, y_pred)
print(f'The accuracy for Random Forest was {rfcacc}, the AUC was {rfcauc}')

# Gradient Boosting:
gbc = best_models['GradientBoostingClassifier']
y_pred = gbc.predict_proba(x_test)[:, 1]
cutoff = err_optis['GradientBoostingClassifier'][0]
gbcacc = accuracy_score(y_test, y_pred > cutoff)
gbcauc = roc_auc_score(y_test, y_pred)
print(f'The accuracy for Gradient Boosting was {gbcacc}, the AUC was {gbcauc}')
```

```
The accuracy for Random Forest was 0.46247464503042596, the AUC was
0.8705561072492551
The accuracy for Gradient Boosting was 0.6835699797160243, the AUC was
0.8739655743131414
```

While it's no surprise that the predicted accuracy was lower for the Random Forest model, and the ROC-AUC was worse for it overall, it is a little surprising that it performs almost-as-well as gradient boosting going by ROC-AUC. The Gradient boosting model generalizes *fairly* well, especially in comparison to random forest, but still not as well as we'd hope. Since the train and test metrics are so different from each other, we can assume that random forest overfit really heavily, even when we employ cross-validation.