Tree Based Inference (Homework 5)

1 Homework 5

[ ]:
```
conda install -c conda-forge pdpbox
```

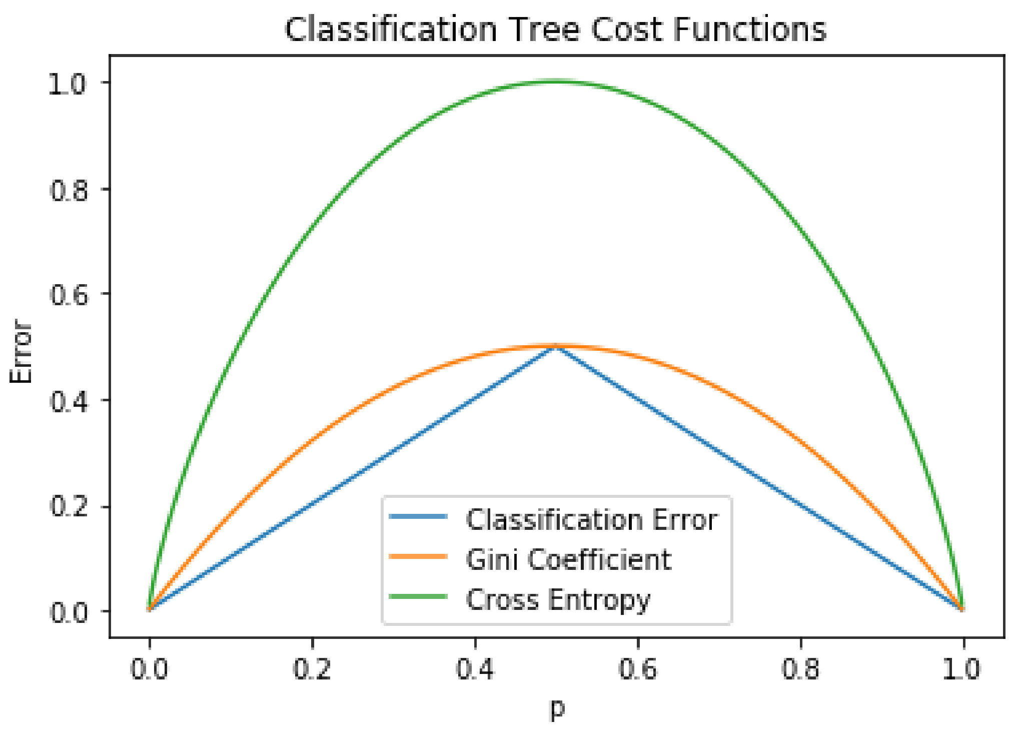
[16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score, cross_val_predict
from pdpbox import pdp, info_plots
```

1.1 Cost Functions for Classification Trees

[15]:
```
probabilities = np.linspace(0, 1, num=1000)[1:-1]
classification_error = [1 - max(i, 1-i) for i in probabilities]
cross_entropy = [-(i*np.log2(i)+(1-i)*np.log2(1-i)) for i in probabilities]
gini_index = [2*i*(1-i) for i in probabilities]

plt.plot(probabilities, classification_error, label='Classification Error')
plt.plot(probabilities, gini_index, label='Gini Coefficient')
plt.plot(probabilities, cross_entropy, label='Cross Entropy')
plt.ylabel('Error')
plt.xlabel('p')

plt.title('Classification Tree Cost Functions')
plt.legend()
plt.show()
```



We find cross entropy to be the most sensitive to changes in p, and hence node purity than the other classification trees. While growing a decision tree, the most important criterion is node purity, so the order of preference would be in descending order of this senstivity: Cross Entropy > Gini Coefficient > Classification Error.

Conversely, cost-complexity pruning is an error-minimization technique, so the previous ordering is reversed for this stage: Classification Error > Gini Coefficient > Cross Entropy.

1.2 Model Estimation

[17]:
```
training_data = pd.read_csv('data/gss_train.csv')
test_data = pd.read_csv('data/gss_test.csv')
training_data.shape, test_data.shape
```

[17]:
```
((1476, 56), (493, 56))
```

[24]:
```
kf = KFold(n_splits=10)
X_training = training_data.drop('colrac', axis=1)
y_training = training_data['colrac']

# Logistic regression
logistic_pred = cross_val_predict(LogisticRegression(solver='liblinear'),
                                          lX_training, y_training, cv = 10)

# Naive Bayes
nbayes_pred = cross_val_predict(GaussianNB(), X_training, y_training, cv = 10)

# Elastic Net Regression
ENR_model = ElasticNetCV(cv=10, random_state=0).fit(X_training, y_training)
print(f'Best Alpha: ', ENR_model.alpha_)
print(f'Best l1 Ratio: ', ENR_model.l1_ratio_)
```

```
Best Alpha:  0.0038452641680228584
Best l1 Ratio:  0.5
```

[50]:
```
# Decision Tree

parameters = {
"min_samples_split": [0.1, 0.3, 0.5],
"min_samples_leaf": [0.1, 0.3, 0.5],
"max_depth": range(1,20),
}
classify_dt = GridSearchCV(DecisionTreeClassifier(),
                          parameters, cv = 10).fit(X_training, y_training)


best_classification_tree = classify_dt.best_estimator_
print(round(classify_dt.best_score_, 4), classify_dt.best_params_)
```

```
0.7656 {'max_depth': 4, 'min_samples_leaf': 0.1, 'min_samples_split': 0.1}
```

[37]:
```
# Random Forest

parameters = {
'n_estimators': range(20,50),
"max_depth": range(1,20),
}
classify_random_forest = GridSearchCV(RandomForestClassifier(),
                                      parameters, cv = 10).fit(X_training,␣
 ↪y_training)


best_random_forest = classify_random_forest.best_estimator_
print(round(classify_random_forest.best_score_, 4), classify_random_forest.
 ↪best_params_)
```

```
0.8062 {'max_depth': 8, 'n_estimators': 48}
```

[43]:
```python
# Gradient Boosting

parameters = {
"n_estimators": [20, 30, 40, 50],
"max_depth": [11, 13, 15],
"learning_rate": [0.025, 0.2]
}
classify_gboost = GridSearchCV(GradientBoostingClassifier(),
                                       parameters, cv = 10).fit(X_training,␣
 ↪y_training)


best_gboosting_model = classify_gboost.best_estimator_
print(classify_gboost.best_score_, classify_gboost.best_params_)
```

```
0.7649051490514905 {'learning_rate': 0.2, 'max_depth': 11, 'n_estimators': 40}
```

[45]:
```python
# Bagging

parameters = {
"base_estimator": [DecisionTreeClassifier()],
"n_estimators": range(20,50)
}
classify_bagging = GridSearchCV(BaggingClassifier(),
                                parameters, cv = 10).fit(X_training, y_training)


best_bagging_model = classify_bagging.best_estimator_
print(classify_bagging.best_score_, classify_bagging.best_params_)
```

```
0.7947154471544715 {'base_estimator': DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'), 'n_estimators': 38}
```

1.3 Model Evaluation

[46]:
```python
best_logistic = LogisticRegression(solver='liblinear').fit(X_training,␣
 ↪y_training)
best_nbayes = GaussianNB().fit(X_training, y_training)
best_elasticnet = SGDClassifier(alpha = ENR_model.alpha_,
                         l1_ratio = ENR_model.l1_ratio_).fit(X_training,␣
 ↪y_training)
```

1.3.1 Cross Validation

[61]:
```
error_logistic = 1 - cross_val_score(LogisticRegression(solver='liblinear'),
X_training, y_training, cv = 10).mean()
error_nbayes = 1 - cross_val_score(GaussianNB(),
                                   X_training, y_training, cv = 10).mean()
error_elasticnet = 1 - cross_val_score(best_elasticnet,
                                       X_training, y_training, cv = 10).mean()

print('Logistic', round(error_logistic, 4))
print('Naive Bayes', round(error_nbayes, 4))
print('Elastic Net', round(error_elasticnet, 4))
print('CART', round(1 - classify_dt.best_score_, 4))
print('Random Forest', round(1 - classify_random_forest.best_score_, 4))
print('Boosting', round(1 - classify_gboost.best_score_, 4))
print('Bagging', round(1 - classify_bagging.best_score_, 4))
```

```
Logistic 0.2073
Naive Bayes 0.2656
Elastic Net 0.2676
CART 0.2344
Random Forest 0.1938
Boosting 0.2351
Bagging 0.2053
```

1.3.2 ROC-AUC

[59]:
```
pred_logistic = LogisticRegression(solver='liblinear').fit(X_training,
                                                            y_training).
 ↪predict(X_training)
pred_nbayes = GaussianNB().fit(X_training, y_training).predict(X_training)
pred_elasticnet = best_elasticnet.fit(X_training, y_training).
 ↪predict(X_training)
```

[62]:
```
roc_log = roc_auc_score(pred_logistic, y_training)
roc_nbayes = roc_auc_score(pred_nbayes, y_training)
roc_elasticnet = roc_auc_score(pred_elasticnet.astype(int), y_training)
roc_CART = roc_auc_score(classify_dt.predict(X_training), y_training)
roc_random_forest = roc_auc_score(classify_random_forest.predict(X_training),␣
 ↪y_training)
roc_gboost = roc_auc_score(classify_gboost.predict(X_training), y_training)
roc_bagging = roc_auc_score(classify_bagging.predict(X_training), y_training)

print('Logistic Regression', round(roc_log, 4))
print('Naive Bayes', round(roc_nbayes, 4))
print('Elastic Net', round(roc_elasticnet, 4))
print('CART', round(roc_CART, 4))
print('Random Forest', round(roc_random_forest, 4))
print('Boosting', round(roc_gboost, 4))
print('Bagging', round(roc_bagging, 4))
```

```
Logistic Regression 0.8166
Naive Bayes 0.7425
Elastic Net 0.7845
CART 0.7847
Random Forest 0.9237
Boosting 1.0
Bagging 0.9994
```

On the cross-validated error rate, random forest, bagging, and logistic regression perform the best. Bagging and Random Forest both have among the highest AUC scores, so it's not clear which is the best model by these two metrics. I'll also run tests on boosting due to its very good AUC score. Although logistic regression has better interpretability properties, it does not do well enough on the AUC metric to lead me to choose it.

1.3.3 Test Data Evaluation

[80]:
```
X_testing = test_data.drop('colrac', axis=1)
y_testing = test_data['colrac']

# Random Forest
predicted_rf = classify_random_forest.predict(X_testing)
error_rf = sum(predicted_rf != y_testing) / len(y_testing)
AUC_rf = roc_auc_score(predicted_rf, y_testing)

# Bagging
predicted_bag = classify_bagging.predict(X_testing)
error_bag = sum(predicted_bag != y_testing) / len(y_testing)
AUC_bag = roc_auc_score(predicted_bag, y_testing)

# Boosting
predicted_boost = classify_gboost.predict(X_testing)
error_boost = sum(predicted_boost != y_testing) / len(y_testing)
AUC_boost = roc_auc_score(predicted_boost, y_testing)

print('Random Forest Classification Error: ' , error_rf)
print('Bagging Classification Error: ' , error_bag)
print('Boosting Classification Error: ' , error_boost)
print('Random Forest AUC: ', AUC_rf)
print('Bagging AUC: ', AUC_bag)
print('Boosting AUC: ', AUC_boost)
```

```
Random Forest Classification Error:  0.20689655172413793
Bagging Classification Error:  0.20689655172413793
Boosting Classification Error:  0.2231237322515213
Random Forest AUC:  0.7994445598006645
```

```
Bagging AUC:  0.7965784982935153
Boosting AUC:  0.7782213050979331
```

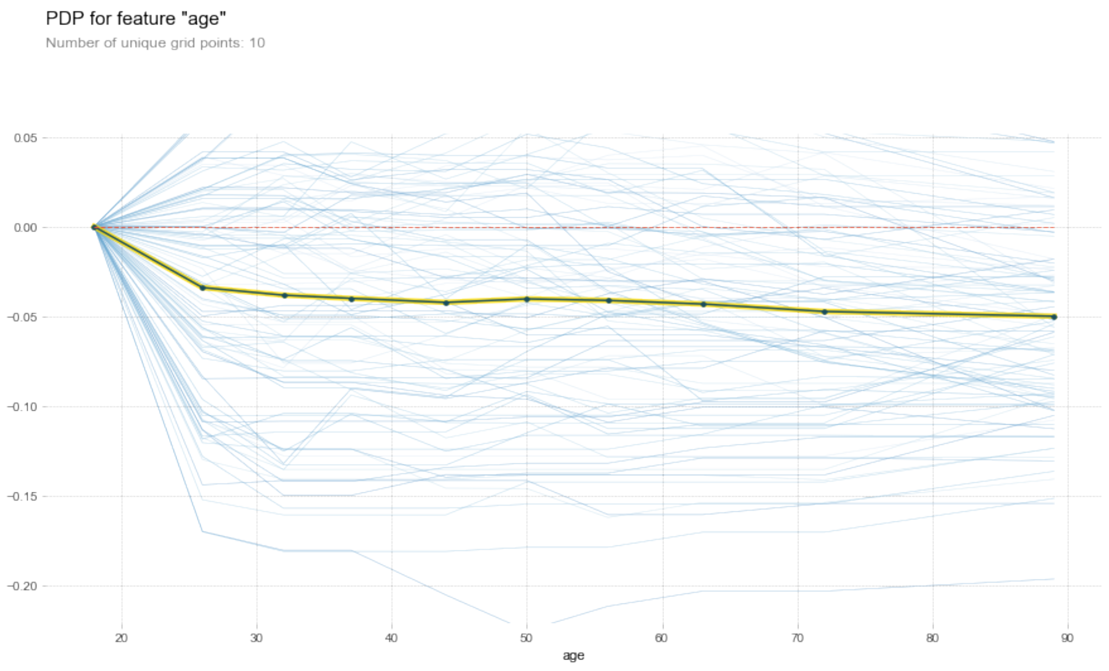
[ ]:
```
predicted = classify_random_forest.predict(X_testing)
n_error = sum(predicted != y_testing) / len(y_testing)
AUC = roc_auc_score(predicted, y_testing)

print('Classification Error: ' , n_error)
print('AUC: ', AUC)
```

Random forest generalizes a bit better than bagging or boosting. Classification error on the testing data is 20.690%. ROC/AUC decreased from 0.924 to 0.800. The model may have suffered from some overfitting, not effectively pruning the branches of the tree.

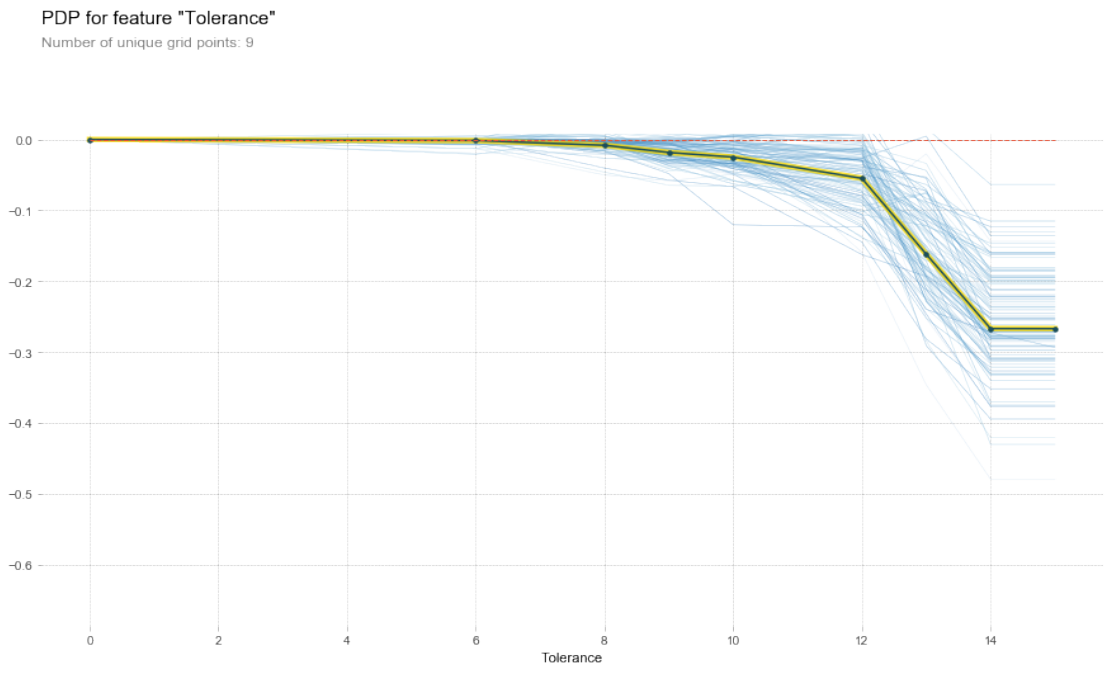
1.4 Partial Dependence Plots

[82]:
```
pdp_age = pdp.pdp_isolate (model=classify_random_forest,
                           dataset=X_training, model_features=X_training.
 ↪columns, feature = 'age')
fig, axes = pdp.pdp_plot(pdp_age, 'age', plot_lines=True, frac_to_plot=100)
```



[83]:
```
pdp_tol = pdp.pdp_isolate (model=classify_random_forest,
                           dataset=X_training, model_features=X_training.
 ↪columns, feature = 'tolerance')
```

```
fig, axes = pdp.pdp_plot(pdp_tol, 'Tolerance', plot_lines=True,␣
↪frac_to_plot=100)
```



The PDP plot displays the marginal effect of a feature on the predicted outcome. The effect of age is limited, leveling out for individuals over the age of 25. However, tolerance has a much more significant effect on predicted outcome. For those with tolerance above 8, and especially above 12, the marginal effect of each individual unit of tolerance makes individuals significantly more likely to object to a racist professor.