EE 460J Lab 7

Team:
Johnson Zhang - xz5993
David Rollins - Der2366
Peter Wagenaar - pjw845

Problem 1

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
import xgboost as xgb
import time
```

Use Random Forests to try to get the best possibletest accuracyon MNIST. This involves getting acquainted with how Random Forests work, understanding their parameters, andtherefore using Cross Validation to find the best settings. How well can you do? You shoulduse the accuracy metric, since this is what you used in Lab 5 – therefore this will allow you tocompare your results from Random Forests with your results from L1- and L2- RegularizedLogistic Regression. What are the hyperparameters of your best model?

In [4]:
```
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [7]:
```
forest = RandomForestClassifier(max_features='log2', max_depth=20)

param_grid = {
'n_estimators':[x for x in range(10, 80, 10)],
'criterion':['gini', 'entropy']
}

forest_CV = GridSearchCV(estimator=forest, param_grid=param_grid, cv=10)
forest_CV.fit(X_train, y_train)
print("best parameters: " + str(forest_CV.best_params_))


# prediction = []
# for i in range(X_test.shape[0]):
#     prediction.append(forest_CV.predict(X_test[i, :]))

print("Score: " + str(accuracy_score(y_test, forest_CV.predict(X_test))))
```

```
best parameters: {'criterion': 'gini', 'n_estimators': 70}
Score: 0.9636428571428571
```

Use Boosting to do the same. Take the time to understand how XGBoost works (and/orother boosting packages available). Try your best to tune your hyper-parameters. As addedmotivation: typically the winners and near-winners of the Kaggle competition are those that are best able to tune an cross validate XGBoost. What are the hyperparameters of your bestmodel?

In [8]:
```
import warnings

warnings.filterwarnings("ignore")
```

```
boosted_forest = xgb.XGBClassifier(n_jobs=-1)

param_grid_Boost = {
    'n_estimators':[x for x in range(10,40,10)],
    'max_depth':[x for x in range(1,5)],
    'learning_rate':[.05],
    'subsample':[.2]
}

forest_GS = GridSearchCV(estimator=boosted_forest, param_grid=param_grid_Boost,
forest_GS.fit(X_train, y_train, eval_metric='auc')
print(forest_GS.best_params_)

print(accuracy_score(y_test, forest_GS.predict(X_test)))
```

```
{'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 30, 'subsample': 0.2}
0.8912857142857142
```

Problem 2

In [9]:

```
X, y = fetch_openml('CIFAR_10_small', version=1, return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [10]:

```
forest = RandomForestClassifier(max_features='log2', max_depth=20)

param_grid = {
'n_estimators':[x for x in range(10, 80, 10)],
'criterion':['gini', 'entropy']
}

forest_CV = GridSearchCV(estimator=forest, param_grid=param_grid, cv=10)
forest_CV.fit(X_train, y_train)

print("best parameters: " + str(forest_CV.best_params_))
print("Score: " + str(accuracy_score(y_test, forest_CV.predict(X_test))))
```

```
best parameters: {'criterion': 'entropy', 'n_estimators': 70}
Score: 0.4056
```

In [11]:

```
boosted_forest = xgb.XGBClassifier(n_jobs=-1)

param_grid_Boost = {
    'n_estimators':[x for x in range(10,40,10)],
    'max_depth':[x for x in range(1,5)],
    'learning_rate':[.05],
    'subsample':[.2]
}

forest_GS = GridSearchCV(estimator=boosted_forest, param_grid=param_grid_Boost,
forest_GS.fit(X_train, y_train, eval_metric='auc')

print(forest_GS.best_params_)
print(accuracy_score(y_test, forest_GS.predict(X_test)))
```

```
{'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 30, 'subsample': 0.2}
0.379
```

In [932]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sci
import pandas as pd
import seaborn as sns
import matplotlib
```

In [933]:
```
train = pd.read_csv("house-prices-advanced-regression-techniques/train.csv")
test = pd.read_csv("house-prices-advanced-regression-techniques/test.csv")
```

In [934]:
```
data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'], test.loc[:,'MSSub
Class':'SaleCondition']))
```

In [935]:
```
train["SalePrice"] = np.log1p(train["SalePrice"])

#log transform skewed numeric features:
indeces = data.dtypes[data.dtypes != "object"].index

skewed_feats = train[indeces].apply(lambda x: sci.stats.skew(x.dropna())) #com
pute skewness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

data[skewed_feats] = np.log1p(data[skewed_feats])
```

In [936]:
```
data = pd.get_dummies(data)
```

In [937]:
```
#filling NA's with the mean of the column:
data = data.fillna(data.mean())
```

In [938]:
```
#creating matrices for sklearn:
X_train = data[:train.shape[0]]
X_test = data[train.shape[0]:]
Y_train = train.SalePrice
```

In [939]:
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import cross_val_score
```

In [940]:
```
model_ridge = Ridge(alpha=.1)
model_ridge.fit(X_train, Y_train)
```

Out[940]:
```
Ridge(alpha=0.1)
```

In [941]:
```
preds = model_ridge.predict(X_test)
preds = np.expm1(preds)
```

In [942]:
```
prediction = pd.DataFrame({"id":test.Id, "SalePrice":preds})
prediction.to_csv("ridge_sol.csv", index = False)
```

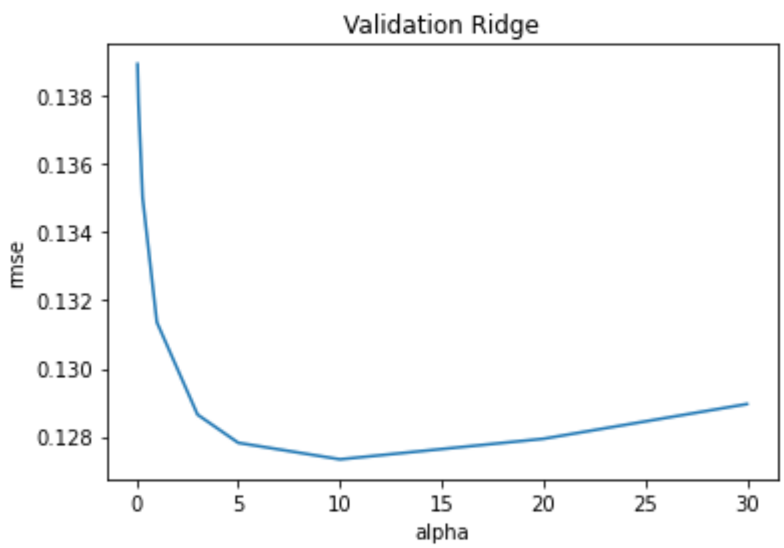
In [943]:
```
#After submitting to Kaggle, we get a RMSE of .1377
```

In [944]:
```
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, Y_train, scoring="neg_mean_squared_error", cv = 5))
    return(rmse)
```

In [945]:
```
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 20, 30]
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]
```

In [946]:
```
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation Ridge")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

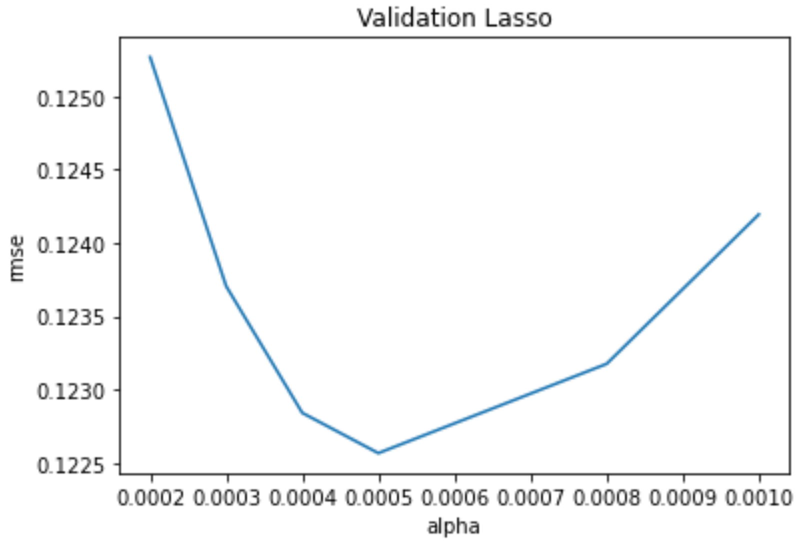
Out[946]:
```
Text(0, 0.5, 'rmse')
```



In [947]:
```
alphasl = [.001, .0008, .0005, .0004, .0003, .0002]
cv_lasso = [rmse_cv(Lasso(alpha = alpha)).mean()
            for alpha in alphasl]
```

In [948]:
```python
cv_lasso = pd.Series(cv_lasso, index = alphasl)
cv_lasso.plot(title = "Validation Lasso")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

Out[948]: Text(0, 0.5, 'rmse')



In [949]:
```python
# For a single LASSO Model, we can get to a RMSE of ~.138
# For a single Ridge Model, we can get to a RMSE of ~.141
```

In [950]:
```python
models_lasso = [Lasso(alpha = alpha).fit(X_train, Y_train) for alpha in alphas
l]
```

In [951]:
```python
coefs = [pd.Series(models_lasso[i].coef_, index = X_train.columns) for i in ra
nge(0, len(alphasl))]
```

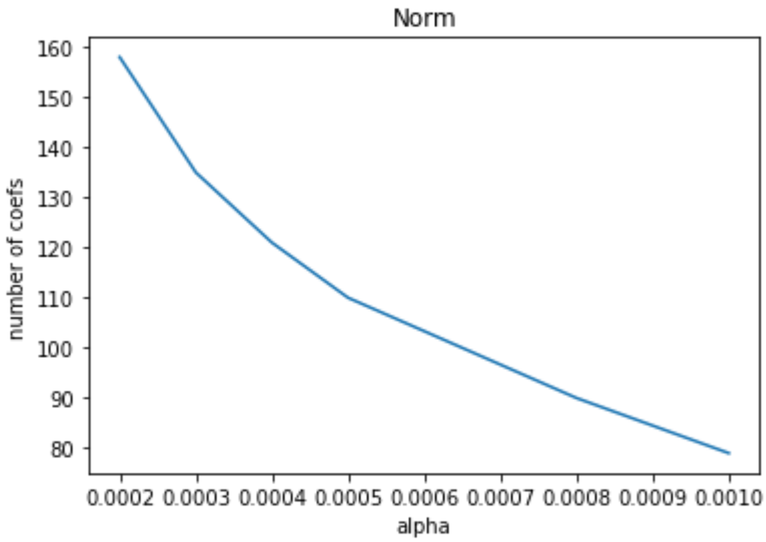
In [952]:
```python
l0 = np.zeros_like(alphasl)
for i in range(0, len(alphasl)):
    l0[i] = sum(coefs[i] != 0)
```

In [953]:
```python
print(l0)
```

```
[ 79.  90. 110. 121. 135. 158.]
```

In [954]:
```
l0 = pd.Series(l0, index = alphasl)
l0.plot(title = "Norm")
plt.xlabel("alpha")
plt.ylabel("number of coefs")
```

Out[954]: Text(0, 0.5, 'number of coefs')



In [955]:
```
predictions_lasso = [models_lasso[i].predict(X_train) for i in range(0, len(al
phasl))]
```

In [956]:
```
X_train.loc[:]["lasso1"] = predictions_lasso[0]
X_train.loc[:]["lasso2"] = predictions_lasso[1]
X_train.loc[:]["lasso3"] = predictions_lasso[2]
X_train.loc[:]["lasso4"] = predictions_lasso[3]
X_train.loc[:]["lasso5"] = predictions_lasso[4]
X_train.loc[:]["lasso6"] = predictions_lasso[5]
```

In [957]:
```
model_ridge_es = Ridge(alpha=10)
model_ridge_es.fit(X_train, Y_train)
```

Out[957]: Ridge(alpha=10)

In [958]:
```
preds = model_ridge_es.predict(X_test)
preds = np.expm1(preds)
```

In [959]:
```
prediction = pd.DataFrame({"id":test.Id, "SalePrice":preds})
prediction.to_csv("ridge_es_sol.csv", index = False)
```

In [960]:
```
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, Y_train, scoring="neg_mean_
squared_error", cv = 5))
    return(rmse)
```

In [961]:
```
cv_ridge_es = rmse_cv(model_ridge_es).mean()
print(cv_ridge_es)
```

```
0.1273373466867077
```

In [962]:
```
# We can get down to a RMSE score of .134 which is better than both the LASSO
 and the Ridge Models
```

In [963]:
```
from xgboost import XGBRegressor
```

In [964]:
```
data_dmatrix = xgb.DMatrix(data=X_train,label=Y_train)
```

In [965]:
```
model = XGBRegressor(learning_rate=1, n_estimators=1000, max_depth=6,
                    min_child_weight=.8, gamma=0, subsample=0.8,
                    colsample_bytree=.8, nthread=4, objective='reg:squarederro
r'
                     )
model.fit(X_train, Y_train)
```

Out[965]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=1, max_delta_step=0, max_depth=6,
             min_child_weight=0.8, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=4, nthread=4, num_parallel_tree=1,
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             subsample=0.8, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [966]:
```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = np.expm1(y_pred)
```

In [967]:
```
params = {"objective":"reg:squarederror",'colsample_bytree': .8,'learning_rat
e': .1,
                'max_depth': 6, 'alpha': 0}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=100,
                    num_boost_round=5000,early_stopping_rounds=10,metrics="rms
e", as_pandas=True, seed=123)
```

In [968]:
```
cv_results.head(1000)
```

Out[968]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	10.380132	0.000874	10.379370	0.096206
1	9.344456	0.000787	9.343646	0.096331
2	8.412309	0.000718	8.411510	0.092148
3	7.573347	0.000648	7.572970	0.088340
4	6.818244	0.000580	6.817646	0.085559
...	...	...	...	...
216	0.021461	0.000843	0.119023	0.045331
217	0.021341	0.000825	0.119003	0.045352
218	0.021213	0.000821	0.119005	0.045363
219	0.021083	0.000809	0.118982	0.045401
220	0.020969	0.000806	0.118971	0.045372

221 rows × 4 columns

In [969]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
220    0.118971
Name: test-rmse-mean, dtype: float64
```

In [970]:
```
prediction = pd.DataFrame({"id":test.Id, "SalePrice":predictions})
prediction.to_csv("xgb_sol.csv", index = False)
```

In [ ]:

Lab 7 Problem 3: XGBoosting

We scored a RMSE of .1247 in the Kaggle Competition. XGBoost did not end up getting us a better score than our Linear Regression with Ensemble Lasso and Ridge Estimators. I found a useful document that helped me implement and understand XGBoost in python. It was helpful in seeing their use of one hot encoding for their variables and being able to relate that to how I used the pandas function of get dummies. There was also another helpful code that showed the use of removing skewness from our data. We could see from our results that some features were very skewed and we were able to counteract this by taking the log of these skewed features. Using XGBoost, we were alone unable to get a better RMSE. However, combined with K-fold cross validation, I was able to see large improvements in our score. We also found that higher k-values resulted in a better score. A k value of 100 was significantly better than that of 10. I tried treating the problem as a classification with many leaf nodes to separate different price ranges through label encoding. This worked decently well but was not as good as the Regression Approach.